UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ALEX GLIESCH

Heuristic algorithms for districting problems

Ph.D. thesis

Advisor: Prof. Marcus Ritt

Porto Alegre
June 2023

CIP - Catalogação na Publicação

Gliesch, Alex
Heuristic algorithms for districting problems / Alex Gliesch. -- 2023.
189 f.
Orientador: Marcus Ritt.

Tese (Doutorado) -- Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2023.

1. Distritamento. 2. Heurística híbrida. 3. Partições conexas. 4. Busca alternada. 5. p-medianas. I. Ritt, Marcus, orient. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da UFRGS com os dados fornecidos pelo(a) autor(a).

“I am, somehow, less interested in the weight and convolutions of Einstein’s brain than in the near certainty that people of equal talent have lived and died in cotton fields and sweatshops.”

— Stephen Jay Gould

ACKNOWLEDGEMENTS

This thesis, in its completed form, is the product not just of my work, but also of the people who helped to make this journey a reality. To everyone who has played a part, however large or small, I owe my deepest thanks, but extend a special thank you to the following people.

I begin by acknowledging my parents Werner and Ursula, and my stepmother Lena for their encouragement and support. You fostered in me a deep intellectual curiosity and a passion for questioning, which have led me on the path of research.

Second, a heartfelt thank you to my Ph.D. advisor, Marcus. Your dedication to intellectual rigor and high academic standards have shaped my way of approaching problems, and were instrumental in my formation as a researcher.

Lastly, to my loving partner, Micheli, thank you for your enduring patience, empathy, and love. You've seen me at my best and at my worst through this journey, providing sustenance, comfort, and a steadfast belief in my abilities when I needed them most.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

LP	Linear Programming
IP	Integer Programming
MIP	Mixed-Integer Programming
MaxDP	Maximum Dispersion Problem
PMDP	p-Median Districting Problem
PCDP	p-Center Districting Problem
DDP	Diameter Districting Problem
TSP	Traveling Salesperson Problem
VRP	Vehicle Routing Problem
CVRP	Capacitated Vehicle Routing Problem
RCL	Restricted Candidate List
VNS	Variable Neighborhood Search
LKH	Lin-Kernighan Heuristic

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABSTRACT

In this thesis we study districting, a general class of optimization problem which asks for the grouping of small geographical units into disjoint clusters, called districts. Districting problems appear in a wide variety of applications, ranging from political districting for elections, to service districting for the distribution of commercial products or the provision of urban services, to the allocation of parcels of farmland into lots. Across the many domains three core requirements are almost always present: that districts be contiguous, geometrically compact, and mutually balanced with respect to attributes associated to the units. Given the broad range of applications, however, these and other planning criteria have been modelled mathematically in different ways, which has led to many distinct optimization problems. In the first part of this thesis, we review in detail the different formulations and common solution methods found in the districting literature.

Since the core problem of finding balanced and connected partitions is NP-hard, solution methods for districting have so far focused on heuristics. However, upon review we find that methods are typically developed independently, with a focus on applications. In the second part of this thesis we look at districting from an application-independent point of view. We propose an extendable heuristic framework that can solve a large set of districting problem variants, and apply it to the three versions of the problem that consider different objective functions for minimizing dispersion, namely a p-median, a p-center and diameter function. In separate analyses, we further extend it towards domain-specific criteria, like routing costs and redistricting. Our heuristic is competitive when compared to other methods which are tailored to a single variant, and improves best known bounds in many cases.

One particular problem which originates in the design of waste collection territories is called the Maximum Dispersion Problem. Differently from classical districting problems, it asks for maximally dispersed rather than compact partitions. The third part of this thesis focuses on solving the Maximum Dispersion Problem. We propose a hybrid heuristic for it which combines a number of components with proven effectiveness in the literature, as well as a new upper bounding scheme. Despite being a heuristic our method finds optimal solutions more often than a current exact approach, and can handle much larger instances.

Keywords: Districting. Territory design. Compactness. Hybrid heuristic. Connected

partitions. Alternating search. Maximum Dispersion Problem. p-Median Problem. p-Center Problem. Metaheuristics..

RESUMO

Algoritmos exatos e heurísticos para problemas de distritamento

Nesta tese nós estudamos problemas de distritamento, que são uma classe geral de problemas de otimização que pedem o agrupamento de pequenas unidades geográficas em clusters disjuntos, chamados distritos. Este tipo de problema aparece numa variedade de aplicações, que vão desde o distritamento político para eleições, ao distritamento de serviços para a distribuição de produtos comerciais ou a prestação de serviços urbanos, até à atribuição de parcelas de terras agrícolas em lotes. Dentre os muitos domínios estão quase sempre presentes três requisitos fundamentais: que os distritos sejam contíguos, geometricamente compactos, e mutuamente equilibrados no que diz respeito a atributos associados às unidades. Dada a vasta gama de aplicações, estes e outros critérios de planejamento têm sido modelados matematicamente de diferentes maneiras, o que leva a vários problemas de otimização distintos. Na primeira parte desta tese apresentamos de maneira sistemática as diferentes formulações de problemas distritamento, e os métodos de solução mais comuns encontrados na literatura.

Dado que o problema central de encontrar partições equilibradas e conectadas é NP-difícil, os métodos de solução para problemas de distritamento têm sido, até agora, heurísticos. Contudo, ao revisar a literatura pode-se observar que estes métodos são tipicamente desenvolvidos de forma independente por pesquisadores, com foco nas aplicações. Na segunda parte desta tese nós olhamos para distritamento sob um ponto de vista independente de aplicação. Nós propomos uma framework heurística extensível que pode lidar com um conjunto grande de variantes de problemas de distritamento, e a aplicamos às três variantes mais comuns que consideram differentes funções objetivo para minimizar a dispersão: a p-median, p-center e diameter. Além disso, em análises separadas nós estudamos a sua extensão para critérios específicos de domínio, como custos de roteamento e redistritamento. A nossa heurística é competitiva quando comparada com outros métodos que são desenvolvidos com foco em uma única variante, e melhora os melhores bounds conhecidos em muitos casos. Um problema particular que tem origem no planejamento de territórios de recolhimento de resíduos chama-se o Maximum Dispersion Problem. Diferentemente dos problemas de distritamento clássicos, ele pede partições com dispersão máxima em vez de

compactas. A terceira parte desta tese foca na resolução do Maximum Dispersion Problem. Propomos uma heurística híbrida que combina uma série de componentes que se mostraram eficazes na literatura, e um novo esquema para obter upper bounds. Apesar de ser heurístico, nosso método encontra mais frequentemente soluções ótimas do que métodos exatos da literatura, e pode lidar com instâncias muito maiores.

Palavras-chave: Distritamento. Planejamento de territórios. Compacidade. Heurística híbrida. Partições conectadas. Busca alternada. Maximum Dispersion Problem. Branch-and-price. Geração de colunas. Problema p-Median. Problema p-Center. Metaheurísticas..

1 INTRODUCTION

Given a connected planar graph $G = (V, E)$ of $n = |V|$ vertices and a set $P = [p]$[1] of p points, a districting problem seeks a partition $S_1 \cup S_2 \cup \cdots \cup S_p = V$ of the vertices into p groups called *districts*. Districting is also sometimes called *territory design*, and in this thesis we use both terms interchangeably. The vertices V, commonly called *basic units* or *geographical units*, represent geographical entities such as city blocks or neighborhoods, which must be grouped into districts. The p-partition of V given by $S = (S_1, \ldots, S_p)$ defines a solution, also commonly known as a *districting plan*.

Districting formulations have been used to model several real-world problems [111]. The most common such problem is that of partitioning an urban area (e.g. a city) into regions, to which service providers will be independently allocated by a central planning body. Examples of such services are police patrolling, waste collection, salt spreading, home care provision, or product deliveries, to name a few. A second kind of problem is political or electoral districting, where in a by-district election system urban areas must be divided into politically unbiased voting sections of equal population. Moreover, we also find districting problems in rural domains, where small parcels of farmland, possibly with different geographical characteristics such as soil quality, elevation or access to water, must be grouped together into lots of equal productivity. An example of a districting plan for the distribution of commercial goods is shown in Figure 1.1.

Multiple definitions exist as to what makes districting plans feasible or acceptable, and for which planning criteria they should be optimized towards. This is mainly determined by the application at hand, and can vary significantly from domain to domain. The following three criteria however are nearly always present, and go back to the work of Hess et al. [100] in political districting during the 1960s.

Balance: districts should be balanced with respect to attribute (also called *activity*) vectors $w_v^a \in \mathbb{R}^+$, $a \in A$ associated with the units, for a given set A of attribute

[1] For positive integer i, we use the notation $[i] = \{1, \ldots, i\}$.

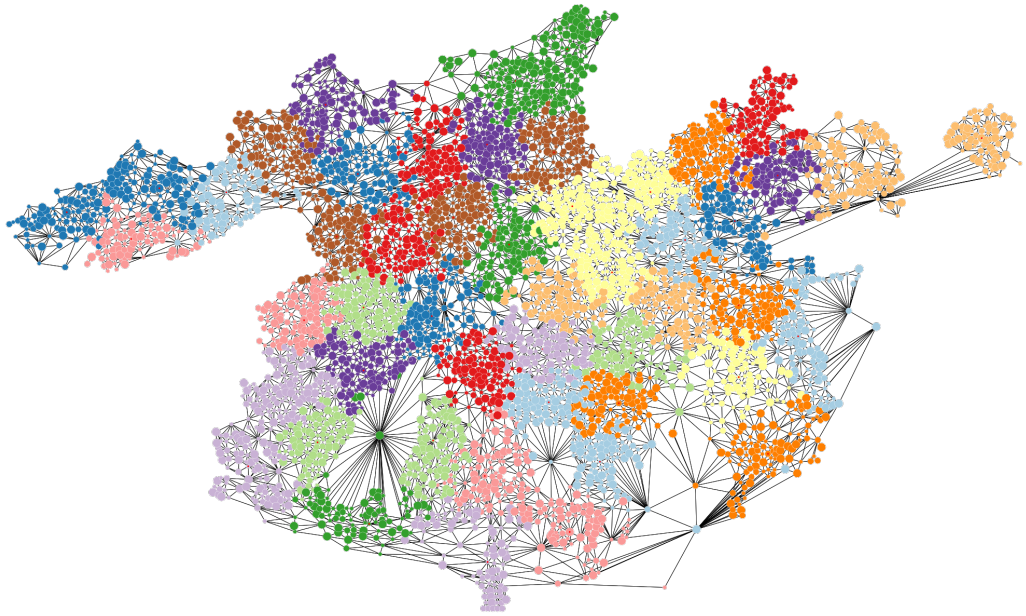


Figure 1.1: Example of a districting plan with $n = 5000$ basic units and $p = 50$ districts, in the city of Monterrey, Mexico (instance from Ríos-Mercado and López-Pérez [182]).

types. These attributes may represent different aspects of the units depending on the application. For example, in service districting domains it is common to use attributes related to the cost or time of servicing a unit, to avoid idle or overworked service agents, and in electoral districting it is desired that districts have an equal number of voters, for reasons of fairness. A district S_i is said to be *balanced* if, for each attribute $a \in A$, its total value $w_a(S_i) = \sum_{u \in S_i} w_u^a$ is within given lower and upper limits U_a and L_a.

Connectivity: each district should be a connected subgraph of G. In some domains, connectivity is also called *contiguity*. It serves to facilitate traversal along the units of a district, and to avoid the need to leave the district's borders to access parts of it.

Compactness: districts should form geometrically compact shapes, ideally convex and neither too long nor too narrow. Computationally, this often translates to minimizing some dispersion function $C(S)$ of distances $d \in \mathbb{R}_+^{V \times V}$ between the units, given by the problem input. Compact districts facilitate traversal by service agents, as paths over territories with compact shapes tend to be shorter, and in political domains can provide visual clues as to whether districts have been manipulated to gain partisan advantage, a tactic known as *gerrymander-*

ing [90]. Despite the existence of several compactness measures, there currently is no consensus as to which one best captures human preference [27, 116]. Because of this, applications rarely specify how $C(S)$ should be computed, and districting plans tend to be evaluated visually in a case-by-case basis.

The usual way of representing these three requirements is through the following mathematical model:

$$\begin{aligned}
\underset{S \in \mathcal{S}}{\textbf{minimize}} \quad & C(S) && && \text{(1.1a)}\\
\textbf{subject to} \quad & L_a \leqslant w_a(S_i) \leqslant U_a, && \forall i \in P, a \in A, && \text{(1.1b)}\\
& G[S_i] \text{ is connected}, && \forall i \in P, && \text{(1.1c)}
\end{aligned}$$

where $\mathcal{S} = \left\{ {V \atop p} \right\}$ denotes the set of all p-partitions of V.

Model (1.1) constitutes the basic building block which the vast majority of districting problems extend. Besides balance, compactness and connectivity, domain-specific criteria are extremely common. Some examples include minimizing differences to prior districting plans, in so-called *redistricting* problems [82, 22] which are common in political districting; forbidding enclaves, i.e. districts within districts [120, 80]; satisfying conflict relations between basic units, where some pairs of units cannot be grouped together because of local rivalries [182]; minimizing the sum of intra- or inter-district routing costs, computed e.g. as optimal Traveling Salesman Problem (TSP) tours over districts [28, 83, 152], a common criterion in service or product distribution contexts; or guaranteeing physical access by all districts to some resource [80, 213], such as e.g. a road network or highway. Many of these criteria appear in several domains, but are often modeled mathematically in different ways.

This combination of requirements makes districting particularly challenging, as most requirements translate into NP-hard subproblems whose interaction creates difficult optimization problems. For example, finding balanced partitions is akin to the Bin Packing Problem [70]; compactness functions are usually modeled as discrete location problems (p-medians and p-centers being the two most common dispersion measures [115]), as geometric partition problems [117], or as minimum k-partitions [3]; conflict constraints between the units combined with the disjoint partition requirement equates to the Vertex Coloring Problem [109]; routing constraints often require solving variants of the Vehicle Routing Problem [125]; and the combination of balancing and connectivity defines the Balanced Connected k-Partition Problem [14]. In fact, to find balanced district partitions alone is NP-hard: the case with $p = 2$ districts,

$A = \{1\}$ and $U_1 = L_1$ generalizes the Subset Sum Problem [69]. Therefore, it follows that districting as described by Model (1.1) in general is NP-hard.

Because districting is difficult, the focus of computational solutions thus far has been on heuristics. Given a problem formulation that minimizes a discrete dispersion function (e.g. p-median or p-center function) subject to connectivity and balancing constraints, current heuristic methods can consistently find feasible solutions for instances with up to 10^4 basic units [77, 182, 184]. Some exact solutions based on branch-and-bound over integer programming (IP) models have also been proposed, but with current computational capabilities they are limited to instances of 500 to 700 basic units [187, 192, 211], with Validi et al. [212] recently reporting results for some instances with up to 1000 units, but with considerable computational effort.

1.1 Motivation and contributions of this thesis

In this thesis we propose computational methods to solve districting problems. We make three main contributions. In the following sections we present our motivation and a short summary of each. Then, we outline the structure of this thesis.

1.1.1 Contribution 1: A generic heuristic for districting

Because applications are so diverse, the literature on districting problems is noticeably fragmented, despite early efforts to propose unified models [113, 114]. As put by Kalcsics et al. [114] about the districting literature, "the tendency to separate the model from the application and establish the model itself as a self-contained topic of research cannot be observed". Upon review, we find few published comparisons between methods across different domains, and it is common for new algorithms to be proposed even if similar or identical formulations have been studied in the past. Also notable are a lack of standard instance sets with best-known values, with many works considering only one or two case studies particular to the domain at hand, and the fact that many implementations and data sets are unavailable because they are proprietary. This poses a major barrier of entry for new researchers, especially ones who are not in contact with a real-world application, since it makes it difficult to assess the effectiveness of a new algorithm or whether a new instance is hard or trivial.

With the above in mind, our first contribution is a heuristic for districting that can handle a large combination of different criteria found in the literature, and thus can be compared to most existing methods. Our focus here are single-objective formulations.

Our aim is for this heuristic to serve as a baseline of comparison, and to promote the study of districting problems from an algorithmic standpoint, independently of applications. The proposed method uses a novel strategy that alternates between two neighborhood searches, one that optimizes the objective and another that minimizes constraint violations. This is embedded into a randomized multistart framework.

In an extensive experimental work we show that our heuristic can handle the three most common dispersion measures found in the literature (p-median, p-center and diameter), as well as other popular criteria (balancing multiple attributes, similarity to previous plans and routing costs), and that it is reasonably competitive when compared to other methods. In many cases, we improve over best-known solutions. To foster its reuse our implementation has been made available in public repositories online, and can be extended to include new optimization criteria without changing the main algorithm.

The details of our heuristic are explained in Chapter 3, which combines our work published in the following conference papers:

- A. Gliesch, M. Ritt, and M. C. Moreira. A multistart alternating tabu search for commercial districting. In A. Liefooghe and M. López-Ibáñez, editors, *European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 10782 of *Lecture Notes in Computer Science*, pages 158–173, Cham, Switzerland, 2018. Springer.
- A. Gliesch and M. Ritt. A generic approach to districting with diameter or center-based objectives. In M. López-Ibáñez, editor, *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 249–257, Prague, Czech Republic, July 2019. ACM.
- A. Gliesch, M. Ritt, A. H. S. Cruz, and M. C. O. Moreira. A heuristic algorithm for districting problems with similarity constraints. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, Glasgow, United Kingdom, July 2020. IEEE.
- A. Gliesch, M. Ritt, A. H. S. Cruz, and M. C. O. Moreira. A hybrid heuristic for districting problems with routing criteria. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–9, Glasgow, United Kingdom, July 2020. IEEE.

1.1.2 Contribution 2: A hybrid heuristic for the Maximum Dispersion Problem

Our second contribution is a new heuristic for the Maximum Dispersion Problem (MaxDP) [61, 78]. Given a set of weighted units, the MaxDP aims to partition them

into groups such that each group is as close as possible to a target weight, and the dispersion, defined as the maximum distance between two objects in the same group, is maximal. This problem is closely related to districting, with some authors also classifying it as a districting problem [179, 185], even though it maximizes a dispersion function rather than minimizing it, and does not require groups to be connected.

The heuristic we propose combines several components which we show to be independently effective. They include an ejection chain-based heuristic for improving dispersion, and an iterated, truncated branch-and-bound algorithm for balancing pairs of districts. We also propose better upper bounds for the problem, which help to terminate the heuristic early and prove optimality by matching it to lower bounds. When compared to the existing exact method in the literature [61], our heuristic finds solutions significantly faster, and more often provably optimal.

The details of our method are shown in Chapter 4, which presents the following published article:

- A. Gliesch and M. Ritt. A hybrid heuristic for the maximum dispersion problem. *European Journal of Operational Research*, 288(3):721–735, 2021.

Moreover, some of the methods we proposed for the MaxDP were used to develop a new heuristic for finding critical subgraphs in the context of the Vertex Coloring Problem. This has led to the following publication, which we did not include in this thesis since it is not related to districting:

- A. Gliesch and M. Ritt. A new heuristic for finding verifiable k-vertex-critical subgraphs. *Journal of Heuristics*, 28:61–91, 2022.

1.2 Structure of this thesis

This thesis is organized as follows. In Chapter 2 we provide a literature review on the most common districting problems, with a focus on models and solution methods, as well as the notation used in the rest of this thesis. Next, in Chapter 3 we present our general heuristic for districting problems. Last, in Chapter 4 we present the proposed hybrid heuristic for the MaxDP. We conclude in Chapter 5 with an overview of our contributions.

2 BACKGROUND

In this chapter we present the main districting models considered in this thesis and provide a general background on districting from a technical standpoint, in terms of the most common optimization criteria, formulations, and solution methods. We also introduce some mathematical notation. Since our focus here are the technical aspects of districting from a computational perspective, we provide only a brief overview of real-world applications in the next section, and then shift our attention to the formal definitions of optimization criteria and to solution methods. For a more in-depth discussion on the applications of districting, we refer the reader to the surveys in Kalcsics and Ríos-Mercado [111] and Ríos-Mercado [178].

Since the scope of this chapter are classical districting problems which follow the general structure of Model (1.1), we leave out the Maximum Dispersion Problem, and review its related literature in Chapter 4.

2.1 Applications

Kalcsics and Ríos-Mercado [111] classify districting applications in 4 major areas: political districting, sales territory design, service districting and distribution districting. As we shall see, there is some overlap particularly for the latter three, and some applications could fit into multiple areas. In the following we give a brief overview of each of these four classes, then mention a fifth area of application we believe is relevant: land allocation.

2.1.1 Political districting

Political districting concerns the design of constituencies (districts) from which political representatives are elected in a by-district election system. The basic units to be grouped are usually *census tracts*: polygonal regions of an urban area that are used by national censuses. In the United States, census tracts average about 4000

inhabitants [26]. To ensure that the voting power, and therefore the political representativeness among voters is equivalent, governments generally adhere to a "one-person, one vote" principle that requires districts to have as equal a population as possible [145]. The strictness of this requirement varies from country to country. In the United States, for example, the year 2000 census identified that districts differ in population by no more than 1% [215], whereas in Germany a more lenient limit of 15% difference from the average population is imposed [87].

As the population of census tracts changes over time, political districts must be periodically redrawn lest they become imbalanced in the number of voters. This process is called *redistricting*. To avoid significant changes to districts, a typical requirement in redistricting is that a minimum degree of spatial similarity be maintained between the old and new districts.

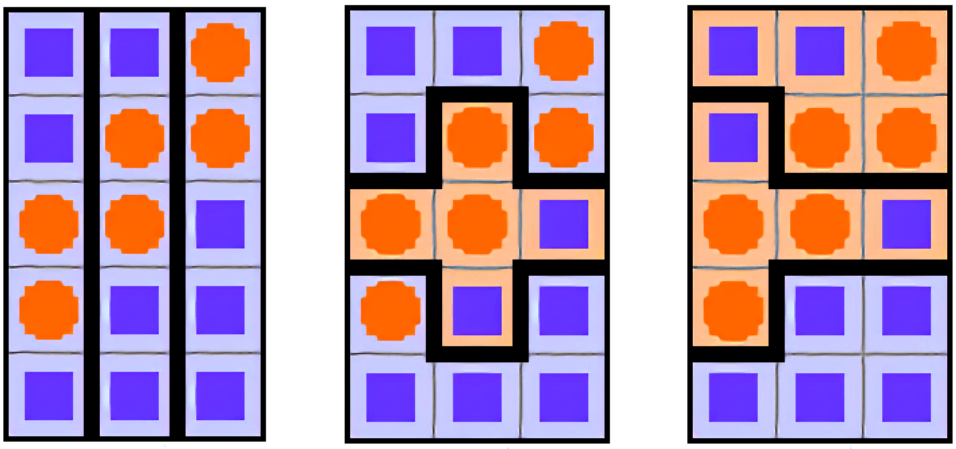
Because the political party currently in power is usually in charge of redistricting, district shapes are sometimes manipulated in order to concentrate the opposition's voting power in just a few districts, while simultaneously diluting its remaining voters over a larger number of districts. This tactic, illustrated in Figure 2.1, is called "cracking and packing" [196] and dramatically increases the chances of reelection to the party currently holding office, as it maximizes the number of "wasted" votes for the opposing party, i.e. votes over the minimum amount needed to win the district.

The general technique of manipulating districts to obtain partisan advantage is known as *gerrymandering* [90]. The term originated from a satirical cartoon published in 1812, which we show in Figure 2.2, depicting a drawing of a salamander over a highly manipulated districting plan for the state of Massachusetts, whose governor at the time was Elbridge Gerry. The practice of gerrymandering famously produces districts with bizarre and distorted shapes, and criteria such as contiguity and compactness have been historically used as indicators of whether a district has been gerrymandered [215].

2.1.2 Service districting

Service districting includes various districting applications concerning the provision of municipal services. Both Butsch [27] and Kalcsics and Ríos-Mercado [111] mention two distinct subclasses within service districting.

The first concerns applications that design fixed service facilities, typically located centrally in their designated district to facilitate access to them. These facilities can be either pre-existing, a planning decision, or a mixture of both, e.g. if one wants to



(a) (b) (c)

Figure 2.1: An example of how districting plans can be manipulated to disproportionally favor either the orange (circles) or blue (squares) party. In 2.1a, blue wins 3-0 despite the ratio of 9:6 voters for blue:orange. In 2.1b, blue wins 2-1, a result which reflects the voter distribution. In 2.1c, orange wins 2-1, despite being a minority in the general population. Source: User "cmglee" at English Wikipedia [210].

Figure 2.2: "The Gerry-Mander", by E. Tisdale, which coined the term *gerrymandering*, showing a highly manipulated districting plan for the state of Massachusetts, United States. Originally published in the Boston Gazette, March 26th 1821.

build a new facility to accommodate increase demand, given that a set of facilities already exist. The three main applications are the planning of school [59, 194], health care [201, 58], and police districts [30]. In all three, each facility (school, hospital or police station) is associated with an "area of influence" (i.e. a district) such that basic units, usually city blocks, are serviced preferably or exclusively by the facility they are assigned to. To avoid imbalanced districts with respect to the amount of resources and personnel needed, which is often limited by the size of the facility itself, districts should deviate as little as possible in the amount of service provided. Districts that minimize travel times from units to the facility (in the case of school or hospital districts) or from the facility to units (in the case of police districts) are generally sought, and often the maximum travel time is limited by a constraint. This translates indirectly to compactness.

The second subclass covers applications that provide service to individual streets of a city. These include e.g. postal services [193], waste and recycling collection [94, 135], police patrolling [39, 133], electric or hydraulic meter reading [42], and salt spreading and winter gritting operations [140, 159, 160, 36]. In this case the basic units are street segments, and the intersections between them define the adjacencies. This makes them edge-based districting problems, which are different in essence from districting problems which are based on grouping the vertices of a graph. If it is not possible to visit both sides of a two-way street on the same trip, each side of a street segment can be considered as a separate basic unit. The goal is to divide a city into districts, each of which will be assigned to a shift or a fixed service provider, e.g. a delivery agent or a truck. To ensure balanced workloads districts must have similar visiting costs, and for maximum efficiency the overall visiting costs should be as low as possible. When a fixed cost is associated with visiting each street, the cost of a district is typically be modelled as a Chinese Postman Tour, which can be computed in polynomial time, and particularly efficiently if the district is Eulerian. With this in mind, some authors optimize for the "Eulerian-ness" of a district, i.e. the number of units with even degree in the induced subgraph [28, 68]. Most authors, however, use contiguity and compactness as proxy functions for the tour costs.

2.1.3 Distribution districting

Distribution districting concerns problems where basic units must be visited periodically over a planning horizon. It appears chiefly in domains related to logistics where a number of available delivery agents (e.g. trucks), originating from a central

depot, must deliver products to clients. Differently from services like postal or waste collection, however, the set of units to be visited at each time step is not known in advance.

The visiting of clients at each time step is usually modeled by a set of TSP tours bounded by a maximum distance, which is determined by time availability, fuel or vehicle storage capacity. One way to tackle the distribution problem is therefore to solve multiple Capacitated Vehicle Routing Problems [209], one in each time step. Unfortunately this can be costly, depending on the number of units, and most importantly results in consecutive solutions that are substantially spatially dissimilar.

A common alternative is to use a "cluster-first, route-second" strategy [76] where a partition is first made that matches parts (territories) to delivery agents. At each time step agents then compute TSP tours only over units in their assigned territory. In distribution districting, the "cluster" part is modeled as a districting problem. Demands are usually modeled as stochastic variables [97, 127], since they are priorly unknown, and so clusters minimizing expected costs are sought. There are many advantages to this approach. Computationally, it shifts the cost of partitioning to a prior step and amortizes it, as it is generally easier to compute, over a time limit of T steps, Tp independent TSP tours over n/p units than to solve T Capacitated Vehicle Routing Problems over n units. Moreover, in practice drivers become familiar with their assigned territories and over time become more efficient in them [111], which helps to reduce travel costs and improve service quality, such as through developing customer relations, for example.

Although the focus of distribution districting is to minimize routing costs, balanced, compact and contiguous districts are still generally desired. Balancing is used to avoid overworked or idle agents, while connectivity and compactness combined are a good proxy objective for low routing costs [83, 76, 28], and are easier to optimize by computational methods.

2.1.4 Sales districting

Sales districting is a mix between service and distribution districting, but in principle closer to the latter. In sales districting clients, each representing a basic unit, are allocated to the salespersons of a company. The goal is to achieve an allocation that is fair to the salespersons with respect to both their workload per period of time, which is proportional to the total travel distance plus visitation time for the assigned clients, and to the expected profit margin, which is influenced by both the number of ac-

counts and their size. To quote Kalcsics and Ríos-Mercado [111] “territories with low sales potential, intense competition, or too many small accounts lead to low morale, poor performance, a high turnover rate, and an inability to assess the productivity of individual territories”. Besides fairness the total profit, given by the total income minus the visitation costs, should generally be as high as possible [50, 44].

While the potential income per client is typically fixed, as in distribution districting district workloads are ideally modeled by TSP tours over the set of client locations. Some authors approximate tour costs by considering a fixed visiting cost to each unit. Also in the same spirit of distribution districting, most solutions use compactness and contiguity as proxies in place of routing costs. Moreover, in the sales domain contiguity has a secondary benefit: preventing salespersons from intruding into one another’s territory to reach clients. Contiguity can be hard to define, however, since adjacencies between clients are sometimes unclear [111].

2.1.5 Land allocation

Lastly, we cite here a fifth domain we believe falls into the scope of districting, but that is seldom mentioned. It concerns the grouping of small parcels of farmland, typically given as a rectangular grid, into larger territories called *lots*. Lots are then assigned either to different owners, or to a specific type of crop which will be planted there. Here, each parcel corresponds to a basic unit and each lot to a district.

To ensure access to every part of a lot by the farmer, lots must be contiguous and sometimes need to have access to a road network. Commonly it is required that lots be fairly balanced with respect to their expected crop yield over time, which is determined by attributes of parcels such as soil type, elevation, or how close they are from a water source [151, 80]. In other cases, each lot simply has a predefined target area [21, 75]. Moreover, lots should have compact shapes resembling regular polygons with few edges [46, 47], since this facilitates both the use of farming machinery and the physical delimitation of the lots with e.g. fences [62]. This relates directly to compactness.

In this discrete case we observe clearly the three districting pillars of balance, connectivity and contiguity. Therefore, this is a classical districting problem. On the other hand, some authors also consider a continuous version where no basic units are given, and lots are defined as polygons on the plane [21, 46, 47]. In this case, it is debatable whether this should be considered a districting problem. We note, however, that the geometric method of Kalcsics et al. [114] for districting seems particularly well-suited

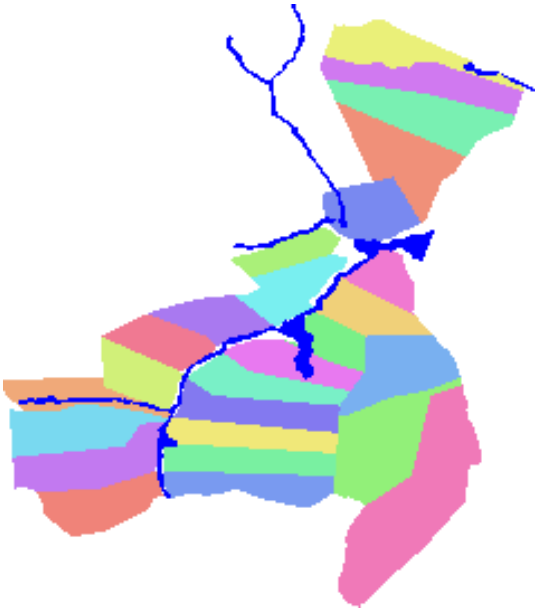


Figure 2.3: Example of a solution to a districting problem in a land allocation domain. Figure from Gliesch et al. [80].

to tackle the continuous version.

Figure 2.3 shows an example solution to a problem originating in land redistribution in Brazil [80]. This was the original problem that motivated our study of districting.

2.2 Modeling domain-specific criteria

As we have just seen, districting problems are often accompanied by domain-specific criteria other than compactness, balance and connectivity. Besides the many criteria cited in the previous section, they can also include variations such as a second compactness function, if compactness with respect to different criteria is desired (see e.g. Garfinkel and Nemhauser [71]), or to treat the balancing of a certain attribute as an objective rather than a constraint (see e.g. Salazar-Aguilar et al. [188]).

In the context of our base model (1.1) additional criteria are typically modeled in one of three ways: i) as a soft objective to be minimized, or as a hard constraint where ii) local (per district) or iii) global (comprising all districts) maximum limits are imposed. Let us denote by $Z^G \subseteq \mathbb{R}^S$ a set of functions which encode optimization criteria defined over entire solutions, i.e. *global criteria*, and $Z^L \subseteq \mathbb{R}^{\mathcal{P}(V)}$ be a set of functions which encode criteria defined over single districts, i.e. *local criteria* (here, $\mathcal{P}(V)$ denotes the power set of V). Often global criteria are simply aggregates of

local criteria using some linking function f, i.e. $z^g(S) = f(z^l(S_1), \ldots, z^l(S_p))$ for some $z^g \in Z^G$ and $z^l \in Z^L$, with common linking functions being the maximum and the sum.

Formally, the inclusion of additional planning criteria results in the general formulation below, which reduces to Model (1.1) if $Z^G = Z^L = \emptyset$:

$$\begin{aligned}
\underset{S \in \mathcal{S}}{\textbf{minimize}} \quad & C(S) + \sum_{z \in Z^G} \lambda^z z(S) && && \text{(2.1a)}\\
\textbf{subject to} \quad & L_a \leqslant w_a(S_i) \leqslant U_a, && \forall i \in P, a \in A, && \text{(2.1b)}\\
& G[S_i] \text{ is connected}, && \forall i \in P, && \text{(2.1c)}\\
& z(S_i) \leqslant U^z, && \forall i \in P, z \in Z^L, && \text{(2.1d)}\\
& z(S) \leqslant U^z, && \forall z \in Z^G. && \text{(2.1e)}
\end{aligned}$$

Upper limits $U^z \in \mathbb{R}^+$ are imposed for hard criteria $z \in Z^G \cup Z^L$, such that no feasible plan can exceed them. Here we assume, without loss of generality, that all criteria are to be minimized. For each global criterion $z \in Z^G$, a weight value $\lambda^z \in \mathbb{R}^+$ is given that defines the objective function as a weighted sum of "soft" criteria, including compactness. This is a common approach for representing districting problems with many soft criteria as single-objective models, since in many cases it can be difficult to define limits U^z that work well on every instance of a domain. Note that in the above we use z as a index to both U and λ, thus viewing them as functions of type $(\mathcal{S} \to \mathbb{R}) \to \mathbb{R}$. As we shall see in Section 2.3.2, this is also why compactness is usually handled as an objective, rather than as a constraint.

A common alternative is to consider a multiplicative weight λ^C to the compactness term C such that $\lambda^C + \sum_{z \in Z^g} \lambda^z = 1$, in order to have the objective function be a convex combination of the different criteria that is easier to manipulate according to the desired output. The same effect could be obtained, however, by appropriately setting λ^z, $z \in Z^G$, hence to maintain the basic model (1.1) as a subproblem we did not include λ^C in Model (2.1).

More generally, the extended model above could even include balancing, connectivity or compactness as generic functions in Z^L or Z^G, instead of explicitly. As we shall see in the next sections, some authors do handle these criteria in different ways, such as by considering balancing as a soft objective, compactness as a constraint, or relaxing connectivity by minimizing the number of connected components per district. This would however make the model too generic, as it could represent many other problems than districting. Moreover, these variants are not the norm. Therefore, in

this thesis we limit our scope to Model (2.1).

Finally, we note that although our focus here are single-objective formulations, as an alternative to the weighted sum approach of (2.1a) many authors have modeled districting as a multiobjective problem with two [190, 188, 191, 42, 43], three [44, 40, 41, 174, 201, 94] or even four [87] global criteria as minimization objectives. Multiobjective formulations are naturally relevant in districting: since soft criteria such as compactness are often ambiguous or ill-defined, it can be desirable to have decision makers select plans by examining a set of Pareto-optimal solutions. All multiobjective formulations we are aware of minimized $C(S)$ along with a set of global criteria, which always included some balancing term defined as the sum of violations to a subset of constraints (2.1b). Some remove balance constraints (2.1b) altogether, in favor of minimizing imbalance as an objective [42, 43, 201, 87].

2.3 Typical districting optimization criteria

In this section we review in detail the most common requirements in districting, including balance, compactness, connectivity, with a focus on mathematical definitions. When applicable we explain the domains where each requirement is relevant, and how it could be included in Model (2.1).

2.3.1 Balance

To balance some type of resource among the districts is a goal shared by all districting problems. Most of the time, these resources are aggregates of numerical attributes associated with basic units, which must be evenly distributed over the districts. Most commonly, limits L and U are defined as maximum relative deviations from the mean attribute over all districts $\mu = \sum_{v \in V} w_v / p$, and a tolerance term $\tau \geqslant 0$ is given such that $L = (1-\tau)\mu$ and $U = (1+\tau)\mu$. Tolerance τ is usually between 0.01 and 0.1, depending on how strictly balancing should be enforced.

The classical attribute from the political districting domain is population: in the United States, for example, a strict "one-person, one-vote" principle is sought [145], such that the voting power of two voters in the same state should be roughly equivalent. This is correlated with districts having equal populations; consequently, tolerances τ in political districting tend to be stricter [212]. Another common attribute from the service and sales districting domains is workload: the time or cost of servicing a basic unit by a service provider or salesperson assigned to each district. Having homogeneous workloads helps to avoid overworking or idling. In sales districting

one also finds attributes such as product demand, number of customers or expected profits, where fairness with respect to potential profits of individual salespersons is desired [220]. Other examples come from land partitioning [80, 46], where parcels of land (basic units) can be associated with area, soil quality or monetary value, and must be evenly divided into lots (districts). The number of units assigned to a district, called district size, can also be viewed as a resource to be balanced. In this case, each unit has an attribute value of 1. For example, some authors [201, 40, 41, 166] impose lower and upper limits on district sizes, while others [39, 80] impose a limit on the ratio between the largest and smallest districts.

The requirement to balance multiple attributes simultaneously, which Model (1.1) considers, is not uncommon. As we have seen this happens often in sales districting domains where both workload and expected profits per salesperson must be balanced [29, 183, 184, 220, 102]. Some works even consider three attributes: workload, number of customers, and product demand [181, 180]. In most cases, the balancing of multiple attributes is enforced by constraints (1.1b), although a few works considering two attributes choose to constrain one and optimize the other [188, 190, 191]. For ease of exposition, unless specifically stated in this chapter we shall assume a single attribute, and thus omit subscripts a.

Besides attributes, some works also seek to balance district routing costs, which dynamically depend on the current solution. For example, Lei et al. [128, 129] minimize, among other criteria, the relative deviation to the mean of the sum of unit attributes (in this case, revenue per unit) minus the cost of a TSP tour over the district, while Lin et al. [136] and Moreno et al. [152] treat routing costs as a hard constraint where an upper limit to it is imposed. On the same note, Muyldermans et al. [159, 160] seek to balance Eulerian tour costs, but as fixed attributes of units which are determined by pre-processing the input graph. This is for an edge-based districting problem variant, however, which is fundamentally different from vertex-based districting.

According to Kalcsics and Ríos-Mercado [111], there is no clear trend on whether to consider balancing as a hard constraint or as an objective to be optimized, and some works even do both (see Table 2.1). When treated as a constraint, it is almost always through two-sided constraints of type (1.1b), although some authors [39, 18] use constraints $w(S_i) \leqslant U \mid \forall i \in P$ for upper limit U, and do not impose lower limits. If $p - \lfloor K \rfloor \leqslant 1$ for $K = p\mu/U$, however, then a lower limit of $L = U(1 + (K - \lfloor K \rfloor - 1)(p - \lfloor K \rfloor))$ is automatically guaranteed, which can result in balance if U is appropriately chosen. The same reasoning applies when setting only lower limit constraints, and in

Constraint	[68, 100, 99, 92, 91, 112, 113, 114, 188, 190, 191, 65, 212, 211, 146, 103, 71, 220, 33, 130, 45, 63, 9, 144, 29, 183, 187, 189, 182, 139, 54, 184, 40, 181, 180, 177, 81]
Objective	[28, 19, 67, 74, 102, 46, 159, 160, 87, 30, 12, 42, 43, 174, 22, 23]
Both	[17, 199, 16, 41, 40, 80, 188, 190, 191, 94, 201]

Table 2.1: Publications that consider balancing as a constraint, objective, or both.

our experience optimal solutions usually have either tight U- or L-constraints, but not both.

When treated as an objective, individual district imbalances have been aggregated in different ways. Let $b(S_i) = |(w(S_i) - \mu)|/\mu$ define the *imbalance* of district i relative to the mean μ. Some authors minimize the sum of deviations $\sum_{i \in P} \max\{0, b(S_i) - \tau\}$ of the imbalance with respect to tolerance τ [19, 67, 74, 23, 22], while others simply minimize the sum of imbalances $b^{sum}(S) = \sum_{i \in P} b(S_i)$ [188, 201, 199, 174, 102, 46, 160, 42], regardless of τ. Another common approach is to minimize the maximum imbalance $b^{max}(S) = \max_{i \in P} b(S_i)$ [16]. Butsch et al. [28] discuss the drawbacks of both *sum* and *max* approaches: the sum allows for highly imbalanced districts to be compensated by balanced ones, while the maximum does not take into account the imbalance of all districts. Therefore, Butsch et al. [28] choose to minimize a convex combination $\alpha b^{sum}(S) + (1-\alpha) b^{max}(S)$, $\alpha \in [0,1]$, of both; Camacho-Collados et al. [30] adopt the same approach. Another mixed formulation, by Goderbauer and Winandy [87], minimizes both b^{sum} and $|\{b(S_i) > 0.15 \mid i \in P\}|$ in a multiobjective problem, where the second term represents the number of districts with imbalance larger than $\tau = 0.15$.

2.3.2 Compactness

The main reason for optimizing towards compact districts comes from political districting, since as we have seen in Section 2.1.1 highly compact districts are less likely to have been manipulated for partisan advantage. Besides political districting, compactness also has beneficial properties in other applications and is generally desired. For example, in distribution districting when delivery routes are to be computed over the districts, it has been shown that compactness can serve as a proxy for routing costs [83, 76]. Moreover, some compactness functions are correlated with connectivity [187].

There exists some ambiguity in the literature concerning the terms “compactness” and “dispersion”. The confusion arises due to the inverse relationship between them:

minimizing compactness leads to dispersed districts, whereas minimizing dispersion yields compact districts. Despite this, the term "minimizing compactness" is often used where "minimizing dispersion" would be more accurate. Since compact districts are generally preferred in the context of districting, this inconsistency has not negatively impacted research, as the meaning remains clear. Therefore, in this thesis we use the terms “compactness” and “dispersion” interchangeably, trusting that the reader will interpret these terms like in the established convention, where compact districts are preferred.

Despite its prevalence, compactness as an optimization criterion has been noticeably hard to define. According to Butsch [27], a compact district shape can loosely be said to “[be] nearly round-shaped or square, undistorted, without holes, and [have] a smooth boundary”. In practice, however, there are often no clear guidelines on how to distinguish compact from non-compact districts other than by inspection by a human expert, commonly called the “eyeball test”.

Historically several quantitative compactness measures have been proposed in attempts to mirror the eyeball test. In his review of compactness, Butsch [27] describe 29 different measures, while Horn et al. [106] show no fewer than 32 measures that were used to evaluate political districting plans in the United States. Most of these measures have been thoroughly criticized in the literature, however, since pathological examples are easy to construct, yielding solutions which are compact in theory but deemed unacceptable by the subjective eyeball test [217, 11]. In a recent survey with public officials and judges in the United States, Kaufman et al. [116] show that none of 7 commonly used compactness measures accurately predicts human preference, but that nonetheless there seems to be a general consensus among the community as to what constitutes compactness: the so-called “you know it when you see it” concept.

Compactness measures generally fall into two categories: contour (or geometric), and distance (or discrete) [52]. Broadly speaking, contour measures take into account geometric elements of the polygon that defines the district on the plane, such as perimeter length, area, or angles of intersection, and assume units are associated with coordinates on the plane. Distance measures, on the other hand, are computed as discrete functions over an arbitrary distance matrix $d \in \mathbb{R}^{V \times V}$ defined over the basic units. In most cases d is Euclidean (and therefore symmetric), but not always, especially in urban domains where tunnels, overpasses and one-way streets are common. In practice, often multiple compactness measures are used in combination to assess a district before it can be considered acceptable [161, 52, 22, 28].

According to Barnes and Solomon [11], the three most frequently used contour

measures are the Polsby-Popper score [167], given by

$$C_{pp}(S_i) = 4\pi \frac{\text{area}(S_i)}{\text{perimeter}(S_i)^2}; \tag{2.2}$$

the Reock score [171], given by

$$C_{reock}(S_i) = \frac{\text{area}(\text{minimumEnclosingCircle}(S_i))}{\text{area}(S_i)}; \tag{2.3}$$

and the Convex Hull score [161], given by

$$C_{ch}(S_i) = \frac{\text{area}(\text{convexHull}(S_i))}{\text{area}(S_i)}. \tag{2.4}$$

Here functions area, perimeter, minimumEnclosingCircle and convexHull return the area, perimeter, minimum enclosing circle and convex hull, respectively, of the 2D shape of a district. Besides these three, other less used measures in computational studies include diameter/area^2 [39], diameter^2/area [71], perimeter^2/area [67], $(2\pi\sqrt{\text{area}/\pi})$/perimeter [22, 128], or combinations of several elements such as edge lengths, intersection angles, and number of boundary points [46, 47]. These measures are local, i.e. defined over single districts, and aggregate into global measures by summing: $C(S) = \sum_{i \in P} C(S_i)$. As discussed at length by Duchin and Tenner [52], however, contour measures are often problematic, since they are prone to:

1. coastline paradox problems, where districts with boundaries produced by natural features such as coastlines are heavily penalized,
2. resolution instability, where the map resolution used to compute the score can have a drastic impact in its value,
3. numerical issues caused by the choice of map projection and coordinate system,
4. the "empty-space problem", where sparsely-populated areas are given the same importance as densely-populated ones.

As a consequence, although contour measures appear very frequently in the districting applications literature, the literature on computational methods heavily favors distance-based measures, since they are much easier to compute and optimize over graph-based representations.

The two most common distance-based measures are the p-median

$$C_{pm}(S) = \sum_{i \in P} \min_{c \in S_i} \sum_{j \in S_i} d_{jc} \tag{2.5}$$

and the p-center

$$C_{pc}(S) = \max_{i \in P} \min_{c \in S_i} \max_{j \in S_i} d_{jc}, \tag{2.6}$$

which are derived from the well-known discrete location problems of the same names [126]. They are *centroid-based* measures, meaning they implicitly define center units for each district S_i: $c^{pm}(S_i) = \operatorname{argmin}_{c \in S_i} \sum_{j \in S_i} d_{jc}$ for the p-median, and $c^{pc}(S_i) = \operatorname{argmin}_{c \in S_i} \max_{j \in S_i} d_{jc}$ for the p-center. These centers turn out to be useful in a number of ways, for example in iterative algorithms such as location-allocation, as we shall see in Section 2.4.1.1, in defining depots for routing applications, or as root vertices for enforcing connectivity in MIP formulations.

The p-median function is far more prevalent in the literature than the p-center, but often appears disguised as variants. Instead of distances d_{ij}, many authors compute p-medians over different pairwise weight functions. One example is the weighted moment of inertia [100, 220, 212], which considers non-symmetric weights $\omega_{ij} = w_j d_{ij}^2$ that take into account attribute values w_j. Other weight functions exist which differ on whether attribute w_j or distance d_{ij} are present and whether they are squared, and Kalcsics and Ríos-Mercado [111] provide a thorough discussion on the advantages and drawbacks of each variant. Note, however, that any weight function ω can substitute d in (2.5) without loss of generality, hence all such problems can be seen as variations of the p-median problem.

Besides centroid-based measures, two other discrete measures stand out. One is the discrete diameter

$$C_{dm}(S) = \max_{i \in P} \max_{j,k \in S_i} d_{jk}, \tag{2.7}$$

defined as the maximum pairwise distance among units in a district, and the other is the sum of edge distances within each district

$$C_{spe}(S) = \sum_{i \in P} \sum_{\{j,k\} \in E(S_i)} d_{jk}, \tag{2.8}$$

which is to be maximized. Here, $E(V')$ denotes the induced edge set of $V' \subseteq V$.

Another measure minimizes the number of cut edges C_{cut} between districts, i.e. edges in E whose endpoints are in different districts. This can be loosely seen as minimizing the districts' "discrete perimeters", and has been shown to have several advantages over other measures in political domains [52, 211]. Minimizing C_{cut} is equivalent to maximizing C_{spe} when $d_{ij} = 1 \ \forall i,j \in V$: in this case, $C_{cut} = |E| - C_{spe}$, since counting unit distances for every edge within a district leaves out only the cut edges. Therefore, throughout this thesis we refer to both C_{cut} and C_{spe} interchange-

p-median	[102, 166, 18, 74, 77, 82, 83, 17, 212, 199, 220, 12, 99, 100, 103, 146, 65, 184, 139, 182, 189, 191, 190, 188, 187, 113, 112, 34, 91, 68]
Diameter	[77, 81, 180, 33, 30, 136, 16, 42, 43, 183, 71, 39]
Cut edges	[211, 45, 63, 9, 144, 52, 31, 53, 19, 104]
p-center	[154, 177, 181, 54, 187, 29]
Others	[174, 22, 23, 127, 128, 129, 201, 71, 132, 47, 46, 41, 40, 67, 107, 28, 152, 39]

Table 2.2: References for the most common compactness measures in the literature.

ably. Moreover, some authors minimize the total cut weighted by d [104, 19, 53], in which case $C_{cut} = |E| - C_{spe}$ holds for any d.

Distance-based measures that group local compactness values into global compactness by taking the maximum, rather by summing, such as p-center and diameter, have the property that there are only $O(n^2)$ possible values for the compactness. This allows for very efficient strategies in heuristic [81, 77] and exact [192] methods based on bounding and binary search, which we discuss in Section 2.4.1.5. However, it also allows a high degree of freedom for assignments below the maximum, and lets non-compact districts be compensated by compact ones, which is generally undesired [111]. On the other hand, sum-based have the property of being "responsive", meaning that a change in a single basic unit has an immediate impact on the compactness value [184]. This fits well the criterion proposed by Horn et al. [106] that a compactness measure should be easy to understand, because decision makers are more likely to trust easy-to-understand measures.

Nearly all optimization models for districting consider compactness as an objective, hence our decision to write Model (2.1) as such. This is mainly due to the fact that upper or lower limits for compactness constraints are hard to specify, since compactness is subjective and values vary significantly across instances. As put by Butsch [27], "the definition of a threshold is actually impossible since the transition from non-compact to compact is fuzzy". This also corroborates previous observations that a hard limit does not work well in practice [219, 161]. Still, some authors have worked with hard limits for the discrete diameter [136, 16, 71, 39], which may be easier to specify as the diameter allows only $O(n^2)$ values. Yet another type of constraints does not use thresholds, but rather requires that the convex hulls of districts must not overlap [152, 114].

Table 2.2 summarizes references in the literature for the most common compactness functions. Although this is not an exhaustive list, a general trend can be seen towards the four distance-based measures discussed above. To our knowledge, none

among the measures classified as “Others” have been reused by multiple research groups. Finally, we note that there is a vast literature on compactness, and in this section have provided an overview of it. For an in-depth discussion, we refer the interested reader to the overview in Butsch [27].

In the rest of this thesis we call the classical problems that result from setting objective C in Model (1.1) to C_{pm}, C_{pc} and C_{diam} the p-*Median Districting Problem* (PMDP), p-*Center Districting Problem* (PCDP), and *Diameter Districting Problem* (DDP), respectively. Previously, Salazar-Aguilar et al. [187] have called the PMDP and PCDP the Median-Based and Center-Based Territory Design Problems. We have renamed them here to frame them in a districting perspective, and to more explicitly relate them to the classical p-Median and p-Center Problems.

2.3.3 Connectivity

As we briefly discussed above, some compactness functions are related to connectivity. One example is the p-median, which, as shown by Salazar-Aguilar et al. [187], tends to produce optimal solutions that are already connected. This is the case particularly in artificial instances such as in rectangular grids or Delaunay triangulations, where distances d are Euclidean or near-Euclidean, and satisfy the triangle inequality. The intuition behind this is not hard to spot: units which are close together in space are more likely to be connected. A much smaller effect has been observed for the p-center and cut edges measures [187, 211].

Because many compactness functions are correlated with connectivity, many authors do not enforce it explicitly [54, 107, 154, 65, 16, 18, 103], although some state that this is a desired property. Others simply repeatedly restart a randomized method until a connected solution is found [100]. In heuristic search, Salazar-Aguilar et al. [190] note that enforcing connectivity as a hard constraint can greatly restrict the search space and pose a hurdle to further improving solutions. To mitigate this, many authors penalize disconnectivity by minimizing the number of disconnected districts as an objective, and discard disconnected solutions at the end [190, 188, 191, 12, 33, 94]. Another approach has been to repair disconnected solutions using post-processing methods when the algorithm stops [212, 28, 24], or after a recombination operator in a genetic algorithm [67, 40, 80]. In MIP formulations, as we will see connectivity constraints lead to very large models and are usually treated by lazy constraints [187, 212] or by restricting the solution space, thereby forgoing optimality [146, 65].

Some methods find connected solutions by construction, and thus the corresponding models do not impose connectivity directly. Two examples are the algorithms of Ricca et al. [175] and Brieden et al. [24] based on Voronoi and shortest path diagrams, respectively. The same is true for greedy constructive heuristics which grow districts incrementally by assigning neighbor units, although these are almost always followed by metaheuristics that do treat connectivity explicitly [136, 81, 53, 30, 181, 191].

So far we have considered connectivity as given by input graph G. Some authors [114, 108, 152], however, use a geometric notion independent of G, whereby basic units are associated with coordinates on the plane and a solution is considered "connected" if the intersection areas of the convex hulls of the districts are zero. This can also be seen as a form of compactness requirement, and helps to guarantee the absence of enclaves [114, 152]. In the context of districting and routing (see Section 2.3.5), it further prevents routes from crossing a different district [111].

2.3.4 Similarity to existing or previous plans

In practice, districting is usually applied several times over a planning horizon. The need to reoptimize districts, also called *redistricting*, arises when due to evolving attributes, current districts get increasingly inefficient or even infeasible. A typical example are electoral districts of roughly the same population, which after changes in its distribution (e.g. by migrations between regions), become imbalanced. Since creating entirely new districting plans can be costly, redistricting problems seek to find an improved or optimal solution that accommodates the attribute changes while also maintaining a high similarity to the current plan.

Similarity measures quantify by how much districts have changed over time [215]. As noted by Williams, Jr. [217], there is comparatively little literature on similarity measures, and these are often not formally defined. For example, according to Goderbauer and Winandy [87], in Germany there exist no clear legal guidelines for how much electoral districts may deviate from existing solutions, but state that it should be as little as possible, as long as balancing constraints are satisfied.

A typical measure for the similarity of a pair of districts is the overlap distance

$$o_a(S_i, S_j) = w_a(S_i \cap S_j) / w_a(S_i \cup S_j), \tag{2.9}$$

defined in terms of overlaps with respect to some attribute $a \in A$. The choice of attribute a depends on the application and can be, for example, area [22, 23, 182],

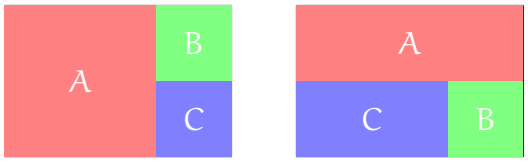


Figure 2.4: A simple example with $p = 3$ districts and different similarity measures. Similarities are 0.5, 1/3, and 2/3, for matching-based mapping, center-based mapping, and the mapping of Bozkaya et al. [22], respectively.

such that $w_a(S_i) = |S_i|$, or population [24]. When a reference solution S^0 is given which corresponds the existing districting plan, a similarity measure usually requires a mapping $m : P \to P$ of old to new districts to evaluate changes in individual units. Given such a mapping, the local similarity of district S_i could be defined e.g. as $\sigma_L(S_i) = o_a(S_i, S^0_{m(i)})$. Global similarity measures usually aggregate by the average, i.e. $\sigma_G(S) = \sum_{i \in P} \sigma_L(S_i)/p$, and thus have range $[0, 1]$, with $\sigma_G(S) = 0$ for totally dissimilar solutions and $\sigma_G(S) = 1$ for identical ones.

Most redistricting problems in the literature fit into this model. A natural choice for mapping m is a bijection, which can be computed by a perfect matching maximizing the sum of pairwise district similarities. However, mapping m is not always bijective. Bozkaya et al. [22, 23] map each district to the counterpart in the original plan with the most overlap, i.e. $m(i) = \operatorname{argmax}_{j \in P} \sigma_L(S_i, S^0_j)$. An advantage of this approach is that it also works if S and S^0 have a different number of districts. A simpler choice for m in problems where centers are defined, such as when local routing depots or p-median compactness are present, can be to use the fixed mapping defined by the district centers of the initial solution. This approach is used, for example, by Brieden et al. [24].

Figure 2.4 illustrates the effect of these different similarity measures. Observe that, when we require a high similarity (0.8 or more) these measures produce the same mapping m, and consequently the same similarity.

If similarity is considered a hard constraint, typically a lower limit $\overline{\sigma^G} \leqslant \sigma_G(S)$ is imposed on the global similarity [42, 43]. Some authors, however, consider it a soft objective and maximize σ_G [22, 23]. Ríos-Mercado and López-Pérez [182] consider similarity both as a constraint and as an objective, but in the objective function a different overlap measure $o'(S_i, S_j) = 0.5 \sum_{v \in S_i \mid v \notin S_j} d_{v c^{pm}(i)}$ is used to penalize dissimilarity in terms of p-median compactness, where $c^{pm}(i)$ is the p-median of S_i. Another approach by both Brieden et al. [24] and D'Amico et al. [39] does not consider similarity explicitly, but uses S^0 as a starting point for its optimization method,

which naturally results in a certain degree of similarity.

2.3.5 Routing criteria

Routing criteria in districting arise when the basic units in a district need to be attended periodically, but there is a cost to do so. A route attending district S_i is typically modeled by a tour which visits all units in S_i in an order such that the total tour cost $R(S_i)$ is minimal. These tours represent, for example, routes used by delivery trucks in the distribution of commercial products to businesses. Combining districting and routing can be useful when immutable districts are desired, but routes must be optimized periodically over a planning horizon. In this case, usually districts are assigned to fixed service providers, who attend any demand arising within it. As we have mentioned in Section 2.1.3, doing so allows providers to become familiarized with the region, which helps to reduce travel costs and improve service quality, such as through developing customer relations, for example. Such demands are typically modeled as stochastic variables [97, 127].

Real-world routes are often constrained by time availability, fuel or vehicle storage capacity, and so frequently budget constraints $R(S_i) \leqslant \overline{R} \mid \forall i \in P$ on the length of each route are imposed [183, 136], for some budget $\overline{R}$. Alternatively, some authors minimize the total length of routes $R(S) = \sum_{i \in P} R(S_i)$ as a soft objective [94, 127, 129, 160, 28]. Often routes are required to start and end at a specified *depot* $h_i \in V$. Depots can be different for each district [159, 160], or the same [152, 136], in which case a global depot $h^G = h_i \; \forall i \in P$ is given. Still, some problems do not use depots and consider open routes (e.g. in waste collection where routes finish at a dump [94]). In Figure 2.5, we show an example of a districting plan with routes connecting to a global depot.

Two overlapping problems here are vehicle routing and location routing. In vehicle routing, clients need to be attended from a central depot by vehicles of a limited capacity, minimizing the total travel distance [209]. For a fixed fleet of p vehicles the routes partition the clients into p regions that are attended by the same vehicle. Differently from our problem, there are no constraints concerning connectivity or compactness of the resulting regions, and no balancing requirements except from the demand upper bound given by the capacity of the vehicle. In location-routing problems, on the other hand, the focus lies on facility location: one must open a set of facilities and assign clients to them. In addition, routes must be constructed for the clients each facility attends. Vehicles are usually capacitated, and thus several routes

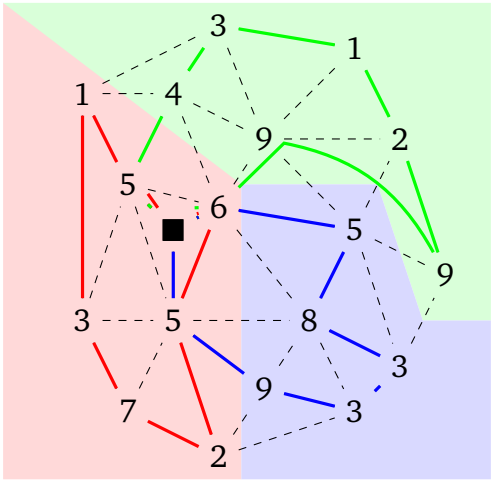


Figure 2.5: An example of a districting solution with routes. There are $p = 3$ districts; units v are labeled with their single attribute w_v^1. The districts are connected and balanced with a tolerance $\tau = 2.5\%$. The central depot is marked by a black square, edges are dashed, routes are shown in different colors. Note how the routes from the central depot pass over units from different districts and the curved connection is the shortest intra-district path going over the green unit labeled 2.

may be required to attend the clients assigned to a facility [169, 2]. In contrast, in combined districting and routing opening costs are not usually considered, and it is assumed that the clients of each district can always be visited with a single tour.

Most commonly, routes are modeled as Traveling Salesman Problem tours. Usually edge costs are given by the instance, and distances between other unit pairs in $\binom{V}{2} \setminus E$ are defined as shortest path distances in G. This means that, in practice, the resulting routes may visit some units more than once, or even make a detour outside their assigned district. This is sometimes undesirable in the distribution of goods, as it can be annoying to customers if e.g. a delivery truck passes by them without stopping. A problem variant may exclude such detours by requiring routes to stay within district boundaries, in which case distances vary according to each particular districting plan. This variant is rare, however, since maintaining shortest path matrices for each district is computationally expensive [208].

Since the TSP is NP-hard [69], it is usually too costly to keep optimal routes while optimizing the districts, despite the existence of high-quality heuristics [137] and exact methods based on branch-and-cut algorithms [5]. To this end, as we have seen many authors consider a “cluster-first, route-second” approach, which partitions the units first without accounting for routing costs, and computes routes after [76]. A selection mechanism (e.g. a multistart or evolutionary heuristic) then typically filters

solutions by their routing costs. In the case of districting, compactness functions can be a proxy objective for routing costs in the first phase [83].

Another approach, used for example in Lei et al. [127, 128, 129], has been to approximate routing costs during optimization with the Beardwood-Halton-Hammersley [13] formula, which states that the length of an optimal TSP tour on n uniformly-distributed vertices on the surface of an area A, with Euclidean distances, converges to $H = b_d/\sqrt{An}$ for large n and some constant b_d (empirically $b_d \approx 0.714$ [6]). Moreno et al. [152] note, however, that uniform distributions may not always be safely assumed in urban scenarios, and propose to use the approximation method of Çavdar and Sokol [222], which does not depend on the distribution of the units and can be further extended to allow a global depot. Nonetheless, some authors have opted to compute exact tours during optimization [183], which leads to a higher precision at the cost of running time.

In service districting domains which service the streets of a city, such as postal services or salt spreading operations, the consideration of routing costs induce an arc routing subproblem. In this case the basic units are rather the edges of a dual graph, and adjacencies are defined by vertex incidences, which makes the problem different from vertex-based districting. Peculiar to this variant is that, if the number of edges in a district is even, routes can be Eulerian tours, which can be computed in polynomial time [101]. Because of this, many authors focus on obtaining Eulerian districts. Butsch [27], for example, optimize towards districts with an even number of edges within a neighborhood search heuristic, while García-Ayala et al. [68] set an upper limit on the number of units with even degree in the districts' induced subgraphs, and solve the problem by branch-and-cut. Another approach has been to preprocess the input graph so that basic units aggregate edge-disjoint cycles, which guarantees Eulerian districts. Bodin and Levy [20], for example, match each odd-degree arc with an opposite arc of the same length, while Muyldermans et al. [159, 160] decompose the input graph into cycles forming a checkerboard pattern.

2.3.6 Number of districts

Although we have limited our scope to problems with a fixed number of districts, some problems minimize p as an objective. This is the case, for example, in service or sales districting domains when service-providing agents have a limited capacity in the number of basic units they can visit, or if the inclusion of a new district is costly [57, 221], meaning a trade-off exists between opening a new district or assigning some

units to districts that are further way.

When p is to be minimized, balancing is always enforced through constraints $B(S_i) \leqslant U \mid \forall i \in P$ for some upper limit U and local balancing criterion $B \in Z^L$. Some authors minimize p as the sole objective [136, 152], while others minimize p among other soft objectives such as balancing and compactness [154, 53]. Another approach by Steiner et al. [201] does not minimize p, but constrains it to a range $[p_{min}, p_{max}]$, while in Lei et al. [127] p affects the costs of global criteria in the objective, but is not minimized or constrained.

We note that an algorithm to solve Model (2.1) could be embedded in a binary search if one wishes to minimize the number of districts, assuming that there exists some p' such that every $p < p'$ result in feasible problems, and every $p \geqslant p'$ result in infeasible ones. This is usually true, due to balancing upper limit constraints. We are not aware of any works that do this, however.

2.3.7 Other criteria

Besides the major criteria above, many optimization models for districting consider problem-specific criteria which have not been widely used outside the application they were proposed for. In this section we review the most relevant.

2.3.7.1 Efficiency gap and responsiveness

A recent trend in political districting has been to optimize a fairness criterion, rather than compactness. This is due to many existing compactness measures still being susceptible to partisan manipulation, as we saw in Section 2.3.2. One fairness criterion is the *efficiency gap* [202, 92, 15], defined as the difference between the number of "wasted" votes for the two parties in a two-party political system. Here, a vote is considered wasted if it was cast for a losing party, or if it exceeds the minimum amount of votes needed to win a particular district. In order to compute it, vote share distributions of the units are obtained using historical results from previous elections. Another, less used criterion is *responsiveness* [214], defined as the derivative of a party's expected seat share with respect to the number of votes it receives.

2.3.7.2 Joint and disjoint assignments

Some authors have considered *disjoint assignment* constraints [182, 16], whereby a set $K \subseteq V^2$ of conflicting or incompatible unit pairs is given such that, if $\{i, j\} \in K$, units

i and j cannot be placed in the same district. Units can be incompatible for several reasons, such as belonging to different administrative districts, being separated by geographical obstacles that are difficult to traverse like bodies of water or dirt roads, or requiring resources or equipment which cannot be carried simultaneously in a single trip. We note that disjoint assignment constraints could be transformed into balancing constraints by creating, for each $k = \{i,j\} \in K$, a new attribute such that $w_i^k = w_j^k = 1$, $w_v^k = 0\ \forall v \in V \setminus \{i,j\}$, $L_k = 0$ and $U_k = 1$. This can increase the number of attributes, however, and may pose difficulties for some algorithms. We are not aware of any works that do this. A related variant is *joint assignment* [29], in which each unit pair in K must be placed together.

2.3.7.3 Accessibility

Sometimes it is desirable that all districts have access to some kind of resource. Here, a given set $R \subset V$ denotes units containing a resource, and $R(S_i) = |S_i \cap R|$ quantifies the degree of access of district S_i with respect to R. This type of requirement is common in problems of land allocation and redistribution. For example, Demetriou et al. [46, 47] optimize, among other things, the number $\sum_{i \in P}[R(S_i) > 0]$ of lots (districts) with access to roads, while Gliesch et al. [80] require that lots with access to water be smaller than lots without access to water, i.e. $R(S_i) > 0 \wedge R(S_j) = 0 \Rightarrow |S_j| > |S_i|\ \forall i,j \in P$. In another application in parliament seating assignment, Vangerven et al. [213] require each seating arrangement (district) to have a minimum number of seats (units) in the front row, i.e. $R(S_i) \geqslant \bar{R}\ \forall i \in P$ for some lower limit $\bar{R}$.

2.3.7.4 Administrative boundaries

In political districting having districts that do not overlap with boundaries of administrative regions such as cities, counties or provinces can be beneficial to the organization and management of elections [175]. This criterion is commonly called *administrative conformity*. According to Goderbauer and Winandy [87], German political redistricting laws state that "where possible, the boundaries of administrative subdivisions should be respected", but do not specify how this should be quantified.

Ricca and Simeone [174] treated conformity by minimizing the number of edges within districts with endpoints belonging to different administrative regions, while Bozkaya et al. [22, 23] maximized the ratio of the largest population in a district with respect to some administrative region and the district's population. Yet other authors do not optimize conformity explicitly, but force units belonging to the same

city to be in the same district, which can be done in a pre-processing step [162, 8].

2.3.7.5 Absence of enclaves

Another common requirement concerns the absence of enclaves, which happen when a district neighbors exactly one other district, and does not touch the outer fringe of the instance. This arises when having to cross another district in order to access a location or resource is undesirable. Note that the notion of enclave here only applies when outer fringes are clearly defined, such as state or city limits, or natural boundaries. Gliesch et al. [80] handle this problem by discarding solutions with enclaves in their genetic algorithm, since they do not occur often, while Steiner et al. [201] merge the enclave with the enclosing district.

In neighborhood search heuristics, one way to address enclaves could be to disallow moves that produce them, since the number of neighbors of a district can be maintained in amortized constant time, and these moves are rare enough that they would likely not constrain the search.

2.3.7.6 Representativeness of minority groups

In political districting, there is often a significant divide in the voting patterns of different social or racial groups. For example, in the United States voters of lower income households tend slightly towards Republican candidates [72, 147], whereas people of African and Native American descent overwhelmingly vote Democratic [216, 147]. As a consequence, depending on how districts are laid out it is possible that minority groups cannot elect any representatives to advocate for their interests, since by being a minority they are not able to win any districts.

One way to mitigate this issue is to give preference to homogeneous districts with respect to minority groups. Doing so increases the likelihood (or even guarantees) that candidates preferred by these minorities will win, since by grouping minority voters together they become the majority in some districts. This may come at the price of lowered compactness, however [49]. In the automated districting literature, we cite two works that do this. Bozkaya et al. [22] considers the grouping of voters by income class. To ensure that districts comprise units of similar income, they minimize as a soft objective the sum, over all districts, of the standard deviation of income among the units of a district. Meanwhile, Arredondo et al. [8] consider indigenous communities. In their work, a basic unit is considered to be "indigenous" if over τ% of its population is of indigenous descent, for some parameter τ. Then, their proposed

mathematical formulation ensures that the number of districts with more indigenous than non-indigenous units is larger than some threshold $p_I < p$.

2.4 Solution methods

In this section we give an overview of the most common solution methods for districting problems. We begin by reviewing MIP-based approaches, both exact and heuristic, in Section 2.4.1; then, in Section 2.4.2 we review metaheuristic approaches.

2.4.1 MIP-based approaches

The first mathematical model for districting, which we shall call the *Hess model*, was proposed by Hess et al. [100] for a p-Median Districting Problem variant in the context of political districting. It is defined as follows.

$$\textbf{minimize} \quad \sum_{i,j \in V} x_{ij} d_{ij} \tag{2.10a}$$

$$\textbf{subject to} \quad \sum_{j \in V} x_{jj} = p, \tag{2.10b}$$

$$\sum_{j \in V} x_{ij} = 1, \qquad \forall i \in V, \tag{2.10c}$$

$$L x_{jj} \leqslant \sum_{i \in V} x_{ij} w_i \leqslant U x_{jj}, \qquad \forall j \in V, \tag{2.10d}$$

$$G[\{ i \mid x_{ij} = 1 \}] \text{ is connected}, \qquad \forall j \in V, \tag{2.10e}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i,j \in V. \tag{2.10f}$$

Semantically, variables $x \in \{0,1\}^{V \times V}$ encode

$$x_{ij} = \begin{cases} 1, & \text{if unit } i \text{ is assigned to the district with center in } j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.11}$$

The objective function (2.10a) minimizes the p-median compactness, and thus a value $x_{jj} = 1$ denotes that a district exists which has j as its p-median. In the original model, distances between units were in fact moment-of-inertia weights $\omega_{ij} = w_i d_{ij}^2$ for given Euclidean distances $d \in \mathbb{R}^2$, but for generality we assume here that distances d are arbitrary and given by the problem input. Constraints (2.10b) ensure exactly p medians (districts) are selected. Partition constraints (2.10c) ensure that

each unit is assigned to exactly one district. Constraints (2.10d) ensure districts are balanced; in the original paper, L and U were defined with respect to a tolerance τ around the average attribute value for all districts, as we saw in Section 2.3.1. Hess et al. [100] did not treat connectivity explicitly in their model, and instead solved this issue by discarding disconnected solutions and restarting their method. Since connectivity is one of the three defining criteria of districting problems, here we include constraints (2.10e) explicitly, and in Section 2.4.1.4 we discuss several ways to implement them.

To improve LP bounds, nearly all authors add valid inequalities of the type

$$x_{ij} \leqslant x_{jj} \quad \forall i, j \in V \tag{2.12}$$

to the Hess model [176, 212, 187]. They enforce that unit i cannot be assigned to center unit j if j itself is not selected as median.

Without constraints (2.10d) and (2.10e), the Hess model corresponds to the classical p-Median Problem where the set of possible medians constitutes the set of basic units. Moreover, if only upper balance (knapsack) constraints are enforced, the model becomes the Capacitated p-Median Problem. The Hess model can be easily modified towards the p-Center Districting Problem (see e.g. Salazar-Aguilar et al. [187] for a formulation), and so the same observations above are true for the p-Center and Capacitated p-Center Problems. A complete definition of the (Capacitaded) p-Median and p-Center Problems can be found in Laporte et al. [126].

Since computers at the time were not sufficiently powerful to solve Model (2.10) with branch-and-bound algorithms, even on smaller instances, Hess et al. [100] proposed a heuristic to solve it based on location-allocation [35], which we explain in the next section. Later, as computer capabilities grew and commercial MIP solvers became widespread, solving the Hess model by branch-and-bound algorithms became a standard in exact methods [187, 212]. Moreover, several authors have extended the Hess model with additional constraints in order to model domain-specific requirements [16, 199, 19, 102, 182].

2.4.1.1 Location-allocation

The location-allocation heuristic, originally proposed by Cooper [35] and first used in districting by Hess et al. [100], is one of the most used heuristics for the p-Median Districting Problems and its variants [103, 65, 184, 212, 113, 218]. It is closely related to the well-known k-means algorithm for clustering [142], and consists of

two phases: *location* and *allocation*, over which it iterates until convergence, i.e. until the results from consecutive iterations are the same.

The location phase consists of heuristically fixing the center variables x_{jj}, $j \in V$ in the Hess model, so that p center variables are fixed to 1 and $n-p$ center variables are fixed to 0. This is enough to satisfy constraints (2.10b). Partition constraints (2.10c) then enforce that, if x_{jj} is fixed to 0, all $x_{ij}, i \in V \setminus \{j\}$ must be fixed to 0, and that if x_{jj} is fixed to 1, all $x_{ji}, i \in V \setminus \{j\}$ must also equal 0. This ultimately reduces the number of variables in the model from n^2 to $(n-p)p$. Let the set $\mathcal{C}$ include the unit indices $j \in V$ for which the corresponding center variables x_{jj} are fixed to 1. The simplified model, here illustrated with a p-median objective, can then be written as:

$$\textbf{minimize} \quad \sum_{i \in V} \sum_{j \in \mathcal{C}} x_{ij} d_{ij} \tag{2.13a}$$

$$\textbf{subject to} \quad \sum_{j \in \mathcal{C}} x_{ij} = 1, \qquad \forall i \in V, \tag{2.13b}$$

$$L \leqslant \sum_{i \in V} x_{ij} w_i \leqslant U, \qquad \forall j \in \mathcal{C}, \tag{2.13c}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in V, j \in \mathcal{C}. \tag{2.13d}$$

The allocation phase solves Model (2.13) to obtain a districting plan. Because of its reduced number of variables, today's MIP solvers can handle this model for instances of up to 3000 units within reasonable time [212]. However, in the past, when computer resources were limited, the allocation model was solved heuristically. The most common heuristic is to compute its LP relaxation, and then to use a *split resolution* rule for rounding fractional (split) variables in the LP solution. This is particularly effective, since Fleischmann and Paraschis [65], Hojati [103] have shown that optimal LP solutions to Model (2.13) admit no more than $p-1$ split variables when only one balancing attribute is considered, and when the p-median function is minimized. For multiple attributes no theoretical bound on the number of split variables is known, but Ríos-Mercado et al. [184] show empirically that for two attributes it is usually less than $2p$. Several split resolution rules have been proposed [100, 65, 103, 113, 24, 184], some of which have the property of being optimal with respect to some desired property of districting plans, e.g. balance [113, 195]. Moreover, if L and U are heuristically set such that $L = U$, which can be done by setting tolerance $\tau = 0$ or by assigning $L = U = (U+L)/2$, the LP relaxation of the allocation problem can be computed in polynomial time using network flow algorithms [65], which are much faster than solving Model (2.13) with the Simplex method.

Given solution S obtained from the allocation phase, the location phase reassigns $\mathcal{C}$ by computing, for each district S_i, its center in the next iteration as the optimal p-median $c^{pm}(S_i)$. To obtain the centers for the first iteration, multiple methods have been proposed. A common one is to simply select a random subset of V [100, 212]. As observed by Resende and Werneck [173] and Validi et al. [212], this simple rule works surprisingly well if location-allocation is executed with several (10 or more) random restarts. A second approach has been to set the initial centers as the optimal p-medians or p-centers of a heuristic solution, obtained e.g. by solving the Hess model with a MIP solver until a feasible solution is found, by running an existing metaheuristic [184], or by a nearest-neighbor allocation starting from randomly-chosen starting points [184].

Note that Model (2.13) did not include connectivity constraints. This is because they usually make the model too difficult to solve in practical time. To mitigate this, some location-allocation methods simply discard disconnected solutions at the end and restart the method with different initial centers [100]. Because optimally compact solutions are often connected, as we discussed in Section 2.3.3, this strategy is often sufficient for smaller instances. Another approach has been to use a split resolution rule which gives preference to repairing disconnectivity [184]. More recently, Validi et al. [212] proposed a post-processing method that fixes units in the interior of districts (i.e. units without neighbors assigned to another district), and then solves a reduced model with full connectivity constraints using a MIP solver to obtain the remaining assignments.

2.4.1.2 Labeling models

Although the Hess model has been the standard for problems with centroid-based objectives, it can be too hard to solve for compactness functions which do not depend on centers such as the (weighted) cut edges or the diameter [211]. For such functions, it is common to use *labeling* variables $x \in \{0,1\}^{V \times P}$ such that

$$x_{ip} = \begin{cases} 1, & \text{if unit } i \text{ is assigned to the district with label } p, \\ 0, & \text{otherwise.} \end{cases} \tag{2.14}$$

With labeling variables, the Diameter Districting Problem can be written as:

$$\begin{aligned}
\textbf{minimize} \quad & \mathcal{D} && && (2.15a)\\
\textbf{subject to} \quad & \mathcal{D} \geqslant d_{ij}(x_{ip} + x_{jp} - 1), && \forall i,j \in V, p \in P, && (2.15b)\\
& \sum_{p \in P} x_{ip} = 1, && \forall i \in V, && (2.15c)\\
& L \leqslant \sum_{i \in V} x_{ip} w_i \leqslant U, && \forall p \in P, && (2.15d)\\
& x_{ip} \in \{0,1\}, && \forall i \in V, p \in P, && (2.15e)\\
& G[\{i \mid x_{ip} = 1\}] \text{ is connected}, && \forall p \in P, && (2.15f)\\
& \mathcal{D} \in \mathbb{R}_+. && && (2.15g)
\end{aligned}$$

Here, constraints (2.15b) ensure that diameter $\mathcal{D}$ is lower bounded by the distance between units i and j assigned to the same district p: the right-hand side is only positive if both i and j are selected in p.

For the Minimum Cut Districting Problem, additional variables $y_{ep} \in \{0,1\}^{E \times P}$ representing whether edge e was cut in district p are also needed. The following model, due to Hojny et al. [104], again omitting connectivity constraints, formulates the problem:

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{e \in E} \sum_{p \in P} w_e y_{ep} && && (2.16a)\\
\textbf{subject to} \quad & x_{ip} - x_{jp} \leqslant y_{ep}, && \forall e = \{i,j\} \in E, i < j, p \in P, && (2.16b)\\
& \text{(2.15c) to (2.15f)}, && && \\
& y_{ep} \in \{0,1\}, && \forall e \in E, p \in P. && (2.16c)
\end{aligned}$$

Constraints (2.16b) link variables x and y, and indicate that edge $e = \{i,j\}$ is cut by district p if unit i is assigned to p, but j is not, with a convention that $i < j$. These constraints are sufficient for correctness. Hojny et al. [104] also propose to use additional constraints

$$\begin{aligned}
& x_{jp} - x_{ip} \leqslant y_{ep} && \forall e = \{i,j\} \in E, i < j, p \in P, && (2.17)\\
& x_{ip} + x_{jp} + y_{ep} \leqslant 2 && \forall e = \{i,j\} \in E, p \in P, && (2.18)
\end{aligned}$$

for strength, but Validi and Buchanan [211] report that in practice they do not yield much improvement. Note that Model (2.16) could also be written with a reduced variable set $y_e \in \{0,1\}^E$ representing only whether an edge was cut; however, as

observed by Validi and Buchanan [211] this leads to poor LP bounds.

Labeling-based models have a high degree of symmetry in variables x. This is because district labels P can be permuted arbitrarily to obtain equivalent solutions. One way to mitigate this issue is to introduce constraints

$$\sum_{i \in V} x_{ip} w_i \leqslant \sum_{i \in V} x_{iq} w_i \qquad \forall p, q \in P, q < p, \tag{2.19}$$

which induce an ordering of districts with respect to attribute weights [148]. Another way has been to use extended partitioning orbitope formulations [211, 104], which have shown to be very effective in similar problems.

2.4.1.3 Set partitioning/covering approaches

Shortly after the work of Hess et al. [100], Garfinkel and Nemhauser [71] proposed a set partitioning formulation over the set $\mathcal{F}$ of all feasible districts with respect to balancing constraints (1.1b) and connectivity constraints (1.1c). It considers variables $z_i \in \{0,1\}^{\mathcal{F}}$, such that

$$z_i = \begin{cases} 1, & \text{if district } i \in \mathcal{F} \text{ is selected in the solution,} \\ 0, & \text{otherwise,} \end{cases} \tag{2.20}$$

and is defined as:

$$\textbf{minimize} \quad \sum_{j \in \mathcal{F}} c_j z_j \tag{2.21a}$$

$$\textbf{subject to} \quad \sum_{j \in \mathcal{F}} z_j = p, \tag{2.21b}$$

$$\sum_{j \in \mathcal{F}} \delta_{ij} z_j = 1, \qquad \forall i \in V, \tag{2.21c}$$

$$z_j \in \{0,1\}, \qquad \forall j \in \mathcal{F}. \tag{2.21d}$$

Here c_j denotes the objective cost of district $j \in \mathcal{F}$, and $\delta_{ij} = 1$ if unit i is included in j, and 0 otherwise. Constraints (2.21b) ensure that exactly p feasible districts are selected, and partition constraints (2.21c) ensure that each unit appears exactly once among the selected districts.

One advantage of this formulation is that any type of objective function can be used, as long as costs c_j are appropriately set. Garfinkel and Nemhauser [71] used

$C(S_i) = \text{diameter}(S_i)/\text{area}(S_i)^2$. As noted by Moreno et al. [152], constraints (2.21c) can usually be substituted by covering constraints $\sum_{j\in\mathcal{F}} \delta_{ij} z_j \geqslant 1 \; \forall i \in V$ without affecting LP bounds. This often results in easier LPs and has the advantage that the corresponding dual variables are always positive. The original formulation by Garfinkel and Nemhauser [71] also had additional constraints which specified upper bounds on the compactness and diameter of districts in $\mathcal{F}$, which we omit here for generality, but did not include connectivity constraints.

The method proposed by Garfinkel and Nemhauser [71] exhaustively enumerates $\mathcal{F}$ and then solves Model (2.21) by a tree search with custom pruning rules. As expected, due to the sheer size of $\mathcal{F}$ their method could only solve small instances. Later, Mehrotra et al. [146] proposed to use the LP relaxation of Model (2.21) as lower bound in a branch-and-bound algorithm for the PMDP. This bound is tighter than the one obtained from linear relaxation of the Hess model, since balancing and connectivity constraints are convexified, and the former, being knapsack constraints, do not have the integrality property [143]. Since $\mathcal{F}$ is exponentially-sized, Mehrotra et al. [146] solved the relaxation by column generation. The corresponding pricing subproblem to obtain new columns is defined over variables $y_i \in \{0,1\}^n$, such that

$$y_i = \begin{cases} 1, & \text{if unit } i \in V \text{ is in the selected column,} \\ 0, & \text{otherwise,} \end{cases} \tag{2.22}$$

and is defined as follows:

$$\underset{u\in V}{\textbf{minimize}} \quad -\pi^0 + \sum_{i\in V\setminus\{u\}} y_i \bar{c}_{iu} \tag{2.23a}$$

$$\textbf{subject to} \quad L \leqslant \sum_{i\in V} w_i y_i \leqslant U, \tag{2.23b}$$

$$y \text{ induces a connected subgraph of } G, \tag{2.23c}$$

$$y_u = 1, \tag{2.23d}$$

$$y_i \in \{0,1\} \qquad \forall i \in V\setminus\{u\}. \tag{2.23e}$$

Here, $\pi^0 \in \mathbb{R}$ is the dual variable associated with constraint (2.21b), $\pi^1 \in \mathbb{R}^n$ is the dual variable vector associated with constraints (2.21c), and $\bar{c}_{iu} = d_{iu} - \pi^1_i$ are the reduced costs. This decomposes into n independent subproblems, one for each center u. To improve convergence, Mehrotra et al. [146] added to the restricted master problem all subproblem solutions found with negative reduced cost. Since these subproblems were still too difficult to solve optimally, however, Mehrotra et al. [146] solved

them heuristically, making their branch-and-bound algorithm an inexact method.

In the realm of heuristic solutions, some authors have proposed to solve Model (2.21) with a non-exhaustive set of columns generated by a randomized heuristic [152, 92, 213]. One issue with this approach is that generating enough districts to guarantee multiple feasible partitions can be challenging. Gurnee and Shmoys [92] propose a way to mitigate this problem by generating districts according to a binary partition tree, such that any pair of district sets originating from two sides of a branch always "fit" together in the partition.

2.4.1.4 Connectivity

Many ways to handle graph connectivity in MIP models have been proposed over the years, and for districting problems no method seems to be a clear winner when it comes to solution speed [211]. Hence, we have thus far refrained from specifying exactly how connectivity should be modeled, and in this section present the most relevant formulations in the districting literature to do so. For the definitions that follow, let $N(v) = \{u \mid \{u,v\} \in E\}$ denote the neighbor set of v, and let the *boundary* $\partial(S) = \bigcup_{v \in S}(N(v) \setminus S)$ of a set of units S be the set of all units neighboring S that outside of S.

The first formulation, due to Drexl and Haase [50] and later used by others [187, 68, 182, 192], uses subtour elimination constraints of the form

$$\sum_{i \in \partial(S)} x_{ij} - \sum_{i \in S} x_{ij} \geqslant 1 - |S| \qquad \forall j \in V,\ S \subseteq V \setminus (N(j) \cup \{j\}) \tag{2.24}$$

over the Hess model variables. They read: "for any subset S of units assigned to center j that does not contain j or a neighbor of j, at least one neighboring unit to S must be assigned to j". Since there are exponentially many such constraints, solution methods typically add them during branch and bound as they are violated in integer solutions. The separation problem is to find connected components for each center j by breadth-first or depth-first search, and can be solved in $O(n)$; a cut is then added for each component that does not contain j. The downside of this formulation is that it requires root vertices to be defined for each district. This is trivial for the Hess model where the center x_{jj} is naturally required in a district centered at j, but would require additional root variables in a Labeling-based formulation.

A similar formulation with an exponential number of constraints is based on cutset

inequalities [163, 32, 212, 211]. It uses constraints of the form

$$x_{ij} \leqslant \sum_{c \in C} x_{cj} \qquad \forall i, j \in V,\ C \in C_{ij},\ C \neq \emptyset, \tag{2.25}$$

where each *separator* $C \in C_{ij}$ is a subset of units such that there exists no path between i and j in $G \setminus C$. Since any separator can be extended by more units without losing its property, C_{ij} are considered to be inclusion-wise minimal. Like constraints (2.24), since C_{ij} can have exponential size, in practice cutset constraints are added lazily during branch-and-bound. The separation problem consists of finding, for each pair of non-adjacent units i and j, a minimum cost vertex cut in the subgraph induced by units $\{u \in V \mid x_{uj} = 1\}$: if a cut exists which costs less than x_{ij}, then a constraint is violated. These cuts could be found efficiently e.g. with Gomory-Hu trees [89] or algorithms specific for planar graphs [123]. As noted by Validi et al. [212], however, this tends to be computationally too expensive and in practice does not provide much improvement over the simple heuristic of Fischetti et al. [64], which finds non-minimal violated separators. In the same vein, Miyazawa et al. [148] propose to contract edges having endpoints assigned to the same district, which generates larger cuts but makes the separation problem considerably smaller.

With respect to formulations based on lazy cuts, Sandoval et al. [192] note that, in most cases, disconnected components identified in the separation problem are singletons, i.e. they have only one unit. They therefore consider inequalities

$$x_{ij} \leqslant \sum_{k \in N(i)} x_{kj}, \qquad \forall i, j \in V \tag{2.26}$$

which force that, for each unit, at least one of its neighbors must be assigned to the same district. Including these inequalities improves linear bounds and significantly reduces the number of cuts added, but make the linear relaxation more difficult to solve, and the experience of Sandoval et al. [192] was inconclusive on whether they ultimately improve the method. In the context of finding minimum-weight arborescences, Ritt and Pereira [186] propose to include them fully for smaller instances and, for larger instances, to add them lazily as they are violated in fractional solutions.

Another well-known class of connectivity constraints are based on network flow. Shirabe [199] and later others [163, 212, 211] use a multi-commodity flow approach for the Hess model, which uses additional flow variables f^j_{uv} for each $j \in P$ and $(u, v) \in A$, where A is the directed version of E. Each f^j_{uv} denotes the amount of flow of type

j passing through arc (u,v). The formulation uses constraints

$$\sum_{v \in N(u)} (f^j_{vu} - f^j_{uv}) = x_{uj}, \qquad \forall j, u \in V, j \neq u, \tag{2.27a}$$

$$\sum_{v \in N(u)} f^j_{vu} \leqslant M x_{ij}, \qquad \forall j, u \in V, j \neq u, \tag{2.27b}$$

$$\sum_{v \in N(j)} f^j_{vj} = 0, \qquad \forall j \in V, \tag{2.27c}$$

$$f^j_{uv} \geqslant 0, \qquad \forall (u,v) \in A, j \in V, \tag{2.27d}$$

where M is a sufficiently large constant, typically $n-1$. Validi and Buchanan [211] note that a better value for M for a given center $j \in V$ center is the maximum number of reachable units, starting from j, using a path of total weight at most U. This can be obtained in $O(n + m \log n)$ with a weighted graph search. To avoid using big-M constraints (2.27b) altogether, Validi et al. [212] proposed an alternative formulation which uses binary variables f^{ab}_{uv} denoting whether arc (u,v) is in a path connecting a to b; however, in their experiments they found this formulation to be ineffective.

Related to the formulation above, recently Hojny et al. [104] and Validi and Buchanan [211] used a single-commodity flow formulation over the Labeling variables:

$$\sum_{u \in V} r_{uj} = 1 \qquad \forall j \in P, \tag{2.28a}$$

$$r_{uj} \leqslant x_{uj} \qquad \forall u \in V, j \in P, \tag{2.28b}$$

$$\sum_{v \in N(u)} (f_{uv} - f_{vu}) \leqslant 1 - M \sum_{j \in P} r_{uj} \qquad \forall u \in V, \tag{2.28c}$$

$$f_{uv} + f_{vu} \leqslant M(1 - y_e) \qquad \forall e = \{u,v\} \in E, \tag{2.28d}$$

$$f_{uv} \geqslant 0 \qquad \forall (u,v) \in A, \tag{2.28e}$$

$$r_{uj} \in \{0,1\} \qquad \forall u \in V, j \in P. \tag{2.28f}$$

Here, f_{uv} denotes the amount of flow passing through arc (u,v). Since labeling formulations do not use district centers, additional variables r_{ij} are required that indicate whether unit u is the root of district j in the flow tree.

Finally, some authors [65, 146, 92] have used a tree-like formulation over Hess

variables:

$$x_{ij} \leqslant \sum_{k \in N(i) \mid s_{kj} < s_{ij}} x_{kj} \qquad \forall i, j \in V, i \neq j. \tag{2.29}$$

It forces the district of center j to be a subtree of a shortest-path tree rooted at j. Here s_{ij} denotes the shortest path from i to j, and the constraints read "unit i can only be assigned to center j if some neighbor of i closer to j is also selected". This formulation is much more efficient than the others presented above, but may exclude some feasible districts and is therefore not optimal.

2.4.1.5 Extended MIP-based methods

Problems where the set of possible objective values is small often admit tailored solution techniques. Two examples in districting are diameter and p-center, whose sets of possible values are the $\binom{n}{2}$ distinct distances between units. Let $D = \{d^1, \ldots, d^R\}$ be the set of distances between units such that $d^1 < \cdots < d^R$, and consider, for example, the p-Center Districting Problem. A simple way to take advantage of the reduced number of objective values is to perform a binary search between d^1 and d^R, where at each intermediate step d^k a feasibility problem $\mathcal{F}(k)$:

$$\begin{aligned}
&\textbf{exists} && x && && (2.30a)\\
&\textbf{subject to} && x_{ij} = 0, && \forall i, j \in V, d_{ij} > d^k, && (2.30b)\\
& && \text{(2.10b) to (2.10f)}
\end{aligned}$$

is solved. Here, constraints (2.30b) disable all assignments with distances larger than d^k. If $\mathcal{F}(k)$ is infeasible, then no solution to the original p-center problem of value less than or equal to d^k exists and d^{k+1} is a lower bound; otherwise, d^k is an upper bound. This follows from the observation that, for all $k \in [R-1]$, the solution set of $\mathcal{F}(k+1)$ is a superset of the solution set of $\mathcal{F}(k)$. Once a k is found such that d^k is both an upper and lower bound, then by definition d^k is the optimal p-center solution value.

Sandoval et al. [192] propose to repeat this process 4 times with increasing levels of relaxation, where each level starts from an improved lower bound (the optimal solution to the last level) rather than simply d^1. This reduces the amount of effort at the last level, where the full model is solved. In their method, a binary search is only used on the first level, and the last 3 levels iterate over the distances linearly from the

lower bound. In some cases, however, a linear search starting down from an upper bound may be more efficient [61], since MIP solvers often take significantly longer to prove infeasibility than to find a feasible solution.

Elizondo-Amaya et al. [54] use a similar approach to compute lower bounds. For fixed $k \in [R]$, the problem

$$\begin{aligned}
\textbf{maximize} \quad & \sum_{i,j \in V} x_{ij} w_i && && \text{(2.31a)}\\
\textbf{subject to} \quad & \sum_{j \in V} x_{jj} \leqslant p, && && \text{(2.31b)}\\
& \sum_{j \in V} x_{ij} \leqslant 1, && \forall i \in V, && \text{(2.31c)}\\
& \text{(2.10d), (2.10f) and (2.30b)}
\end{aligned}$$

is considered. It allows some units and districts to be unassigned, but solutions which cover V are preferred by the objective, which maximizes the total attribute of the assigned units. The authors prove that, if any upper bound to Model (2.31) is smaller than $\sum_{i \in V} w_i$, then d^{k+1} must be a lower bound, as no feasible solution of value d^k or less exists. They use a Lagrangean relaxation of constraints (2.31c) for an upper bound, and find the optimal k via binary search.

Besides the p-center objective, this type of strategy was used also by Fernández et al. [61] for the Maximum Dispersion Problem, and could be adapted for the diameter objective, although we are not aware of any published works that do so.

A different approach is due to Validi et al. [212] for the p-Median Districting Problem. It considers the following Lagrangean relaxation of the Hess model, which relaxes partition constraints (2.10c) and balancing constraints (2.10d), and drops connectivity constraints:

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{i,j \in V} x_{ij} d_{ij} + \sum_{i \in V} \alpha_i \Big(1 - \sum_{j \in V} x_{ij}\Big) + \\
& \sum_{j \in V} |\lambda_j| \Big(x_{jj} - \sum_{i \in V} x_{ij} p_i / L\Big) + \sum_{j \in V} |\nu_j| \Big(\sum_{i \in V} x_{ij} p_i / U - x_{jj}\Big) && \text{(2.32a)}\\
\textbf{subject to} \quad & \text{(2.10b) and (2.10f).}
\end{aligned}$$

Here, $\alpha \in \mathbb{R}^n$, $\lambda \in \mathbb{R}^n$ and $\nu \in \mathbb{R}^n$ are the Lagrangean multipliers corresponding to the dualized constraints. This relaxation decomposes into p subproblems which can be solved combinatorially in $O(n)$ time, and so the optimal multiplier vectors for α, λ and ν can be efficiently found by gradient descent. The authors then show that the

reduced costs of the x variables in the Lagrangean optimal solution can be computed in constant time. By comparing these reduced costs to an upper bound UB obtained by a heuristic, the x variables may be fixed to either 0 or 1, as follows. Let x^*_{ij} be the optimal solution to the Lagrangean with value z^*, and let c_{ij} be the reduced cost of negating variable x^*_{ij} (i.e. setting it to 0 if $x^*_{ij} = 1$, or setting it to 1 if $x^*_{ij} = 0$). Then, if $z^* + c_{ij} > UB$, variable x_{ij} can be safely fixed to x^*_{ij}. This approach fixes, on average, over 95% of variables in political districting instances of 1000 or fewer units, leading to many instances being solved during the solver's presolve phase.

2.4.2 Metaheuristic approaches

Besides methods based on linear or integer programming, districting has mainly been solved by heuristics which systematically explore the solution space using combinatorial operators. There are many metaheuristic frameworks in the literature [73], but for districting three categories are prevalent: pure constructive heuristics, neighborhood search-based heuristics, such as local search, tabu search, or GRASP (which combines local search with a greedy constructive heuristic), and evolutionary algorithms. In this section we outline the main strategies within these categories.

2.4.2.1 Initial solutions and constructive heuristics

Optimization methods must be initialized either by solutions built computationally, or by existing solutions extracted from real-world applications. In districting, the latter approach has the advantage that existing plans are usually readily available and often satisfy implicit criteria which may have not been formulated, since they were drawn by human experts. (This may not always true in the case of gerrymandered electoral districts, however, where non-compact plans are purposefully drawn for political advantage.) On the other hand, because typically only one solution (or a series of related solutions evolved over a time period) is available, starting from existing plans does not provide the solution variability needed for multistart or evolutionary heuristics, which rely on generating a large number of distinct solutions. Hence, only a few authors use this approach [94, 39, 33].

Among computational methods, the most widely used are constructive heuristics. Starting from a seed unit defining a singleton district, a constructive heuristic iteratively assigns neighboring units to it according to a greedy, sometimes randomized criterion. This process is done either simultaneously for all p districts, or sequentially, such that districts are built one at a time. During construction solutions are allowed

to be partial functions, i.e. to not cover V, and so lower balancing constraints are relaxed. Non-partial solutions are called *complete*. For a partial solution S let us define its set of *unassigned* units as $\bar{S} = V \setminus \bigcup_{i \in P} S_i$, and denote by $S[u \to i]$ the solution resulting from the assignment of unit $u \in \bar{S}$ to district S_i. Generally only assignments $u \to i$ where $N(u) \cap S_i \neq \emptyset$ are allowed, so S remains connected, although some authors allow disconnected districts under a penalty to the objective function, as we saw in Section 2.3.3.

A few methods consist purely of a constructive heuristic [159, 160, 136], but in most cases constructive heuristics are embedded within a larger metaheuristic framework. For example, they may be followed by a neighborhood search procedure such as local search, used to generate several solutions in the initial population of a genetic algorithm, or in destroy/repair heuristics where an operator deletes part of a solution and then reconstruct it [67, 190, 80, 42, 43].

Constructive heuristics generally differ in three aspects:

1. how seeds are selected,
2. how the assignment order is determined,
3. whether districts are built simultaneously or sequentially (i.e. one at a time).

When districts are built simultaneously, seeds are pre-selected and the heuristic iteratively executes the feasible assignment a with minimum $\phi(S[a])$, for some fitness function ϕ, until S is complete. Function ϕ is usually a weighted sum of the compactness objective and constraint violations [159, 160, 29, 181, 183, 191, 180, 28, 42, 43, 30], and some authors use $\phi(S[u \to i]) = w_u d_{us_i}$, where s_i is the seed of district i, which amounts to building discrete weighted Voronoi regions over the initial seeds [175, 130, 46, 80]. In order to determine the next assignment all feasible assignments must be considered, and, depending on the choice of ϕ, caching techniques can be used to save effort [132, 94]. For very large instances where even updating a cache is costly, however, a common technique is to execute the best b assignments in a batch, where b is a parameter [80]. Another approach is to select the next assignment in two stages: first, a district $i \in P$ is greedily selected according to some fitness function ϕ_1 (e.g. balancing constraint violations), and then, among all neighboring units to S_i, one is selected according to a different function ϕ_2 (e.g. compactness) [94, 191].

When districts are built one at a time, the first seed is selected randomly and units are assigned to the current district until no feasible assignment exists, or until a "closing" criterion is met, usually when $w(S_i) \geqslant L$ [53, 136, 181, 183, 180, 42, 43, 127]. Next, a new seed is chosen from which a new district is "opened". As above, assign-

ments are made greedily with respect to fitness ϕ. Since new districts can be added as long as there are unassigned units left, this sequential approach is more common in problems where p is not fixed or is to be optimized [53, 127, 201]. However, some authors also use it for fixed p followed by a repair heuristic that merges or splits districts [181, 42, 43, 23, 136].

Regarding seed selection, most simultaneous construction methods simply sample the p seeds uniformly from V [30, 53, 132, 23, 22, 201, 12, 46, 127, 41]. This provides a high degree of variability without requiring a randomized assignment strategy, but, as in location-allocation, may depend on further optimization or multiple restarts to avoid poor initial random seed configurations. Another approach, originally proposed by Erkut et al. [56] and used by several authors [183, 191, 180, 190], is to choose the first seed randomly from V and each next seed as the unit whose minimum distance to the other seeds is maximal, i.e. $s_i = \operatorname{argmax}_{u \in \bar{S}} \min_{j \in [i-1]} d_{us_j}$ for $1 < i \leqslant p$. This ensures the initial seeds are better dispersed over the topology. In routing domains with local depots, it is also common to select the depots themselves as seeds [159, 160]. Yet another approach is taken by Gliesch et al. [80], who first expand discrete weighted Voronoi regions starting from random seeds, and then set the initial seeds as the weighted centroids of these regions. This equates to executing a single step of k-means clustering [142].

When districts are built sequentially, the selection of new seeds often takes into account the current partial solution by choosing the next seed as the unit that would have next been greedily assigned to the previous district [127, 128, 129], although some methods simply pick the next seed randomly from $\bar{S}$ [53, 132, 22, 23, 201, 67]. Also common is to select the next seed as the unassigned unit of smallest degree [42, 43, 181], since this supposedly favors contiguous districts [181].

Besides constructive heuristics, less-used initial solution approaches are through location-allocation [184], geometric partitioning methods [28, 113], or other heuristics which do not fit into the constructive framework above. Of these, we note two that stand out. The first is based on the hierarchical agglomerative algorithm for clustering [158], and was used by Caballero-Hernández et al. [29]. Differently from constructive heuristics, it starts with every basic unit being a district in itself, and iteratively merges neighboring districts until only p districts remain; each next pair of districts to be merged is selected by a greedy mechanism. The second, proposed by Ricca and Simeone [174], works on a decomposition of the input graph into trees. First, a random spanning tree of G is generated using graph search. Then, p seed units are greedily chosen. Next, while there exist two or more seeds within the same

tree, select two such seeds and delete some edge on the unique path between them, thus splitting the tree. At the end p trees remain, each inducing a district.

2.4.2.2 GRASP and multistart heuristics

A common way to randomize constructive heuristics is to use a semi-greedy or α-greedy approach [172]. At each iteration, instead of greedily executing assignment a_{best} that minimizes fitness ϕ, it selects uniformly at random an assignment $a \in RCL$ for some restricted candidate list $RCL \subseteq \mathcal{A}$, where $\mathcal{A}$ denotes the set of all feasible assignments. The most common choice for RCL is

$$RCL = \left\{ a \in \mathcal{A} \mid \phi(S[a]) \in [\phi_{best}, (1-\alpha)\phi_{best} + \alpha\phi_{worst}] \right\}, \tag{2.33}$$

for $\phi_{best} = \phi(S[a_{best}])$, $\phi_{worst} = \max\{\phi(S[a]) \mid a \in \mathcal{A}\}$, and $\alpha \in [0,1]$. Parameter α dictates how much variability is desired compared to a greedy solution, with $\alpha = 0$ leading to a pure greedy solution and higher values of α favoring more diverse solutions at the expense of quality. In districting, nearly all randomized constructive heuristics use this definition of an RCL. A less used option for the RCL is to select the best K elements from $\mathcal{A}$, for some parameter K [80].

Multistart algorithms have been extremely popular for districting [132, 174, 29, 181, 183, 191, 180, 42, 43, 30] and are the main use case for randomized constructive heuristics. They consist of repeatedly generating a new semi-greedy solution and attempting to improve it through a neighborhood search algorithm, while recording the best solution seen so far. This is usually repeated for a fixed number of iterations or until a time limit is reached. When the neighborhood search is a local search, this equates to the well-known GRASP heuristic [172]. Ideally, the blend between randomness and solution quality provided by the semi-greedy construction should allow each iteration to explore a different part of the search space, while also reducing the burden on the neighborhood search since it starts from a relatively high-quality solution.

Within the GRASP framework, several extended techniques have been proposed [172]. We note three in the districting literature.

First, Ríos-Mercado and Fernández [181] use a filtering mechanism to discard unpromising initial solutions. The rationale is that, since the local search is the computational bottleneck, only solutions which are likely to improve the incumbent S_{best} should be submitted to it. To this end, they maintain the average improvement $\bar{\beta}$ in fitness yielded by the local search relative to the initial solution, and only run the lo-

cal search for solutions S with $\psi(S)\bar{\beta}(1-\bar{\beta}) < \psi(S_{best})$, where ψ is the local search's fitness function.

A second approach, called *reactive* GRASP, is to use a self-adjusting α. It considers a set $A = \{\alpha_1, \ldots, \alpha_m\}$ of possible values for α with probabilities $p_1, \ldots, p_m$ associated with value. Iteratively after a fixed number of moves, probabilities are updated so that values of α which lead to better solutions are favored. Ríos-Mercado and Fernández [181] use this approach for districting and set $p_i = q_i / \sum_{j \in [m]} q_j$, where $q_i = A_i^{-8}$ and A_i is the average fitness for solutions obtained with $\alpha = \alpha_i$.

Third, Ríos-Mercado and Escalante [180] combine GRASP with path relinking. Path relinking consists of transforming a source solution into a target solution through a sequence of modifications, and returning the best intermediate solution. In this case, path relinking is done by iteratively shifting boundary units from one district to another, such that the Hamming distance between some pair of matched districts decreases. Given source S^1 and target S^2, such a matching is obtained by computing a maximum weight matching in the complete bipartite graph $K_{p,p}$, where the weight of edge $\{i,j\}$ is the distance between the initial seeds for S^1_i and S^2_j. The authors propose two approaches to use this: i) to generate a fixed number B of multistart solutions and execute path relinking between all pairs from B, and ii) an evolutionary GRASP (see Section 2.4.2.4 for details).

2.4.2.3 Neighborhood search heuristics

Neighborhood search heuristics iteratively modify a solution using moves chosen from some set N of operators over solutions, called a *neighborhood*. Ultimately, the goal is to find an improved solution with respect to some fitness function ψ. Most methods define $\psi = \phi$, but not all. In districting methods the most common neighborhood search heuristics are local search [53, 94, 181, 191, 190, 207, 130, 42, 43, 184], tabu search [175, 28, 22, 23, 19, 128, 127, 94], and simulated annealing [94, 53, 39, 132, 175, 17]. Below we give a brief overview of the three. For an in-depth discussion on them, see Gendreau and Potvin [73].

Local search iteratively selects the best improving move from N with respect to ψ, and stops when no more improving moves exist. This strategy is also commonly called *best-improvement* local search. A variant is *first-improvement* local search, which immediately executes the first improving move encountered while iterating over N. In this case, usually one iterates over N in a random order, or in a round-robin fashion over multiple executions.

Tabu search, originally proposed by Glover [85], always executes the best move from N, even if it does not improve the incumbent. To avoid cycling, i.e. moving to a worse neighbor from a local optimum and then immediately going back, a *tabu list* is used to exclude the most recent moves. More specifically, when a move is made its arguments (e.g. a district or a basic unit) are declared *tabu* and cannot take part in another move for a certain number of iterations, called the *tenure*. The search typically stops after a predefined maximum number of iterations without improvement.

Simulated annealing considers moves from N in random order, and each move is executed with probability $p = e^{\frac{\Delta}{T}}$, where Δ is the change in ψ induced by the move, and T is a temperature variable. Temperature T starts at a fixed $T = T_{init}$ and is multiplied by a cooling rate $\alpha < 1$ after each iteration. The search stops once T reaches a predefined value T_{max}.

The central component in these strategies is the neighborhood N. In districting the typical neighborhood used by nearly all heuristics is *shift* [29, 181, 180, 191, 183, 94, 42, 43, 53, 30, 132, 28, 22, 23, 174, 190, 207]. In the following, we overload notation and also consider solutions $S : V \to \mathcal{P}(V)$ as functions mapping units to districts, such that $S(u) = S_i$ if $u \in S_i$. A shift $u \to i$ moves unit u to the district with index i, and removes it from its current district. Given solution S, we denote by $S[u \to i]$ the solution that results from shift $u \to i$, i.e. from setting $S_i \leftarrow S_i \cup \{u\}$ and $S(u) \leftarrow S(u) \setminus \{u\}$. Therefore, the shift neighborhood is defined as

$$N_{shift}(S) = \{u \to i \mid i \in P, u \in \partial(S_i), S(u) \setminus \{u\} \text{ is connected}\}. \tag{2.34}$$

Because of the connectivity requirement, a shift $u \to i$ is only feasible when u is on the boundary of district S_i. Since the perimeter of a planar region generally scales relative to the square root of its area, when distances are Euclidean and unit locations are uniformly distributed on the plane, experimentally we have observed that $|N_{shift}| \approx \sqrt{np}$ [81].

Although shifts are sufficient to transform a solution into any other, a local search limited to improving shifts may miss promising parts of the search space which require passing through worse intermediate solutions. To this end, composite moves are sometimes used.

Among composite moves, the perhaps natural extension to shifts are *swaps*. Swaps have been used extensively in similar partition problems [25, 95, 179], but less so for districting [94, 22]. A swap $u \leftrightarrow v$ exchanges the districts assigned to units u and

v. Specifically, given solution S, $S[u \leftrightarrow v]$ denotes the solution obtained by setting $S(u) \leftarrow (S(u) \cup \{v\}) \setminus \{u\}$ and $S(v) \leftarrow S(v) \cup \{u\}) \setminus \{v\}$, and the swap neighborhood is thus defined as

$$\begin{aligned} N_{swap}(S) = \{ & u \leftrightarrow v \mid u, v \in V, u \in \partial(S(v)), v \in \partial(S(u)), \\ & \text{both } (S(u) \cup \{v\}) \setminus \{u\} \text{ and } (S(v) \cup \{u\}) \setminus \{v\} \text{ are connected} \}. \end{aligned} \tag{2.35}$$

If we maintain the assumption that $|N_{shift}| \approx \sqrt{np}$, for a given pair of districts the maximum number of possible swaps between them is $\binom{\sqrt{n/p}}{2} = O(n/p)$, in the worst case where the districts only border each other. Consider now a graph G' obtained from any feasible solution by substituting each district by a vertex, and the adjacencies between districts by edges. Because G is planar, G' must also be, and therefore it can have at most $3p-6$ edges. This means the number of pairs of adjacent districts is bounded by $3p-6$. Thus, the size of the swap neighborhood is approximately $(3p-6)\binom{\sqrt{n/p}}{2} = O(n)$.

Other, less-used composite moves are double shifts [28], where two feasible shifts are realized simultaneously, 3-chains [25, 149], where, given units u, v and w and solution S, u is shifted to $S(v)$, v to $S(w)$, and w to $S(u)$, and whole-boundary moves [151], where, for districts i and j, every unit of S_i neighboring some unit of S_j is moved to S_j. The main drawbacks of elaborate composite moves is that their neighborhoods tend to grow exponentially with the number of units involved, and designing efficient dynamic update functions gets increasingly difficult.

Shifts and composite moves are often combined in a variable neighborhood search (VNS) fashion [95]. The idea is to consider composite moves only if no improving shifts are found; otherwise, the best shift is executed. This can save a considerable amount of effort since composite neighborhoods are significantly larger than N_{shift} and, in the majority of cases, if an improving composite move exists an improving shift exists as well. As we shall see in Chapter 3, this happens in up to 80% of cases when using swaps. Despite this, due to their asymptotically larger neighborhoods composite moves usually remain the bottleneck.

Another bottleneck can be to check whether a move violates connectivity constraints (e.g. by removing a bridge unit from a district). The naïve implementation executes a breadth-first or depth-first search at an average cost of $O(n/p)$ on the district, which may easily dominate the running time. Surprisingly, very few authors mention this cost or how it could be mitigated. To our knowledge, prior to this thesis the only works on this topic were by King et al. [118, 119, 120], who proposed a sophisticated data structure that checks for disconnectivity by exploiting the fact that

G is planar. Unfortunately, their implementation is not available. In Chapter 3, we show a simple solution to this problem that works for shifts and swaps and runs in $O(n/|N|)$ amortized time

Depending on the nature of ψ, it may be easy to identify moves which never lead to an improved solution, or which are improbable do to so. For example, when minimizing imbalance through local search Butsch et al. [28] only consider shifts $u \to i$ if $w(S(u)) > w(S_i)$, since the remaining moves cannot improve it. To optimize compactness, the same authors consider only shifts $u \to i$ where $S(u)$ is among the λ least compact districts, for some parameter λ, and in order to favor Eulerian tours, they only considers shifts that reduce the number of odd-degree units in the induced subgraphs of the districts. Similarly, Li et al. [132] limit their neighborhood to shifts $u \to i$ which $\psi(S[u \to i])$ is maximal for some fixed u, and Hanafi et al. [94] only allow shifts $u \to i$ and swaps $u \leftrightarrow v$ where $w(S(u))$ is maximal, i.e. units can only be removed from the heaviest district.

Some criteria, especially compactness, tend to be easier to optimize in partial rather than complete solutions, since connectivity constraints severely limit the set of available moves [190]. Therefore, when neighborhood search is preceded by a constructive algorithm, some authors define ψ differently from the constructive heuristic's fitness ϕ. The most common choice for ψ is a weighted sum of the compactness objective and constraint violations [174, 29, 181, 183, 191, 180, 28, 42, 43, 30, 22, 23, 184]:

$$\psi(S) = C(S) + \sum_{\text{constraint types } c} \lambda_c V_c(S), \tag{2.36}$$

for parameter weights $\lambda_c \in \mathbb{R}$.

When there are multiple constraint types (e.g. several balancing attributes, routing budgets, minimum similarity constraints), a common strategy is strategic oscillation [86, 22, 183, 185], which updates weights λ every μ iterations by setting

$$\lambda_c = \begin{cases} \alpha\lambda_c & \text{if } V_c > 0 \text{ in all previous } \bar{\mu} \text{ iterations,} \\ \lambda_c/\alpha & \text{if } V_c = 0 \text{ in all previous } \bar{\mu} \text{ iterations,} \\ \lambda_c & \text{otherwise,} \end{cases} \tag{2.37}$$

for given parameters $\alpha, \mu, \bar{\mu} > 1$.

2.4.2.4 *Evolutionary algorithms*

Given a solution set P obtained by repeated executions of some heuristic, evolutionary algorithms iteratively update P in order to improve its best element with respect to some fitness function ψ. Set P is called the *population* or *reference set*. Most evolutionary algorithms comprise the following components:

1. a recombination operator $recombine : [\mathcal{S}]^a \to [\mathcal{S}]^b$ which takes a solutions from P and combines them to produce a set of b solutions (in most cases $a = 2$ and $b = 1$),
2. a mutation operator $mutate : \mathcal{S} \to \mathcal{S}$ that modifies a solution, usually in a random way to obtain variability,
3. selection operators $selectRecombine : \mathcal{P}(\mathcal{S}) \to [\mathcal{S}]^a$ and $selectMutate : \mathcal{S} \to \mathcal{S}$ to choose solutions to undergo the above two operations, typically through k-tournaments [88],
4. an improvement operator $improve : \mathcal{S} \to \mathcal{S}$, typically a neighborhood search,
5. an update operator $update : \mathcal{P}(\mathcal{S}) \times \mathcal{P}(\mathcal{S}) \to \mathcal{P}(\mathcal{S})$ that, given sets P_1 and P_2, selects the population of the next iteration from $P = P_1 \cup P_2$, typically with $|P| = |P_1|$.

Algorithm 1 General evolutionary algorithm structure

Input: a solution set P
Output: a solution S_{best} s.t. $\psi(S_{best}) \leqslant \operatorname{argmin}_{S \in P} \psi(S)$

repeat
  $S_{best} \leftarrow \operatorname{argmin}\{\psi(S), S \in P \cup \{S_{best}\}\}$
  $P_{new} \leftarrow \emptyset$
  $P_r \leftarrow selectRecombine(P)$ ▷ recombination
  for $R \in P_r$ **do**
    $R \leftarrow recombine(R)$
    $P_{new} \leftarrow P_{new} \cup \{improve(S) \mid S \in R3\}$
  $P_m \leftarrow selectMutate(P \cup P_{new})$ ▷ mutation
  for $S \in P_m$ **do**
    $S \leftarrow mutate(S)$
    $P_{new} \leftarrow P_{new} \cup \{improve(S)\}$
  $P \leftarrow update(P, P_{new})$
until termination criterion
return S_{best}

The different evolutionary algorithms vary in the definition of the 5 operators above. For example, in genetic algorithms [105] $update$ typically clamps the pop-

ulation down to the best $|P|$ solutions, and do not use $improve$; memetic algorithms [153] extend genetic algorithms by using $improve$ as a neighborhood search. In scatter search [84], often $mutate$ is not used and $update$ maintains a set of $|P|$ maximally diverse solutions with respect to some distance metric $d(S, S')$ between solutions. Evolutionary GRASP [172] follows the same idea as scatter search, but $selectRecombine$ randomly picks a single solution S^1 and $recombine$ generates a semi-greedy solution S^2 to be combined with S^1, typically through path relinking. Algorithm 1 shows the general structure for these four algorithms.

In districting, evolutionary approaches are limited to the four algorithms described above: genetic algorithms [33, 201, 12, 67, 80, 46, 129, 40, 41], memetic algorithms [17, 207], scatter search [190] and evolutionary GRASP [180]. In the following we briefly describe some of the more interesting component choices that have been proposed, which we believe could be useful across multiple districting domains.

Most authors mutate solutions by applying a fixed number of randomly chosen moves from N_{shift} [33, 17, 201, 40, 41]. This is often computationally costly, however, since connectivity constraints must be updated after every move. Another approach, due to Gliesch et al. [80], is to set all units u such that $\min_{v \in \partial(S(u))} d_G(u, v) \leqslant k$ as unassigned, where $d_G(u, v)$ is the graph distance (in hops) from u to v. Then, the resulting partial solution is reconstructed using a semi-greedy approach. Similarly, Tavares-Pereira et al. [207] select a random subset of districts to simultaneously reconstruct using the merge-based constructive heuristic of Caballero-Hernández et al. [29], which we saw in Section 2.4.2.1.

Regarding recombination, given two solutions S^1 and S^2 a common approach is to "cut" a subset of districts from S^1 and "paste" them onto S^2, therefore substituting the overlapping assignments and relabeling disconnected parts. Since it is unlikely that this covers perfectly the intersected districts in S^2, typically this leads to a solution with more than p districts. Tavares-Pereira et al. [207] and Steiner et al. [201] repair this by iteratively merging districts according to some heuristic, until there are p left. Lei et al. [129], on the other hand, set all non-overlapped units in the intersected districts of S^2 as unassigned, and reconstruct the solution using a semi-greedy algorithm. If the problem allows a variable p, however, no repair heuristic is needed, e.g. in Datta et al. [40, 41].

Another recombination method was developed independently by Gliesch et al. [80] for a genetic algorithm and Salazar-Aguilar et al. [190] for scatter search. Given solutions S^1 and S^2, first a correspondence between districts is obtained through a minimum cost matching m on the weighted bipartite graph $K_{p,p}$ where the cost of

edge $\{i,j\}$ is $|S_i^1 \cap S_j^2|$. Next, a partial solution S is constructed such that $S_i = S_i^1 \cap S_{m(i)}^2$ and, if $S_i = \emptyset$ for any i, a new seed is introduced among the unassigned units. Then, S is completed by a semi-greedy algorithm and returned.

We also mention the evolutionary GRASP method of Ríos-Mercado and Escalante [180], which recombines solutions through path relinking. We described this procedure in detail in Section 2.4.2.2.

Most evolutionary approaches for districting use the same solution encoding we have thus far, i.e. a mapping $V \to P$. However, other encodings sometimes provide more flexibility when choosing recombination and mutation operators, and open the possibility of using well-known operators from the literature. We note two examples from the literature.

First, Bação et al. [12] consider a variant of the p-Median Districting Problem and encode solutions as a list $S = (s_1, \ldots, s_p)$ of median units, such that a districting plan can be constructed by greedily assigning each unit from V to its nearest median. Mutation is done by moving one median to a neighbor unit, and recombination by Uniform Crossover [204]: from two parent solutions, each median in the recombined solution has equal chance of being selected from either parent. Demetriou et al. [46] use a similar approach for a continuous version of the land allocation problem. In their case, medians are the (x,y) coordinates of centroids which are expanded into p regions through weighted Voronoi diagrams. They mutate by displacing a median by a random vector in the plane and recombine through Blend Crossover [205]: given two parent solutions $(s_1^1, \ldots, s_p^1)$ and $(s_1^2, \ldots, s_p^2)$ and parameter $\lambda \in \mathbb{R}$, a recombined solution $(s_1, \ldots, s_p)$ is constructed such that $s_i = \lambda s_i^1 + (1-\lambda)s_i^2$.

Second, Forman and Yue [67] encode solutions as permutations of V. To build a districting plan, their method iterates linearly over a permutation adding units to the current district, until an attribute threshold is reached. Then, a new district is started. Since this is unlikely to yield connected solutions, the authors propose a heuristic to repair disconnectivity. Mutation is done by a random move from the 2-opt neighborhood [38], and crossover by Maximal Preservative Crossover [155], which, given S_1 and S_2, selects a randomly-chosen contiguous subsequence r from S_1, deletes every element of r from S_2, then inserts r in a random location in S_2.

3 A GENERIC HEURISTIC FOR DISTRICTING

In this chapter we present our generic heuristic for districting problems. The chapter combines the contributions of the following four conference papers:

- A. Gliesch, M. Ritt, and M. C. Moreira. A multistart alternating tabu search for commercial districting. In A. Liefooghe and M. López-Ibáñez, editors, *European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 10782 of *Lecture Notes in Computer Science*, pages 158–173, Cham, Switzerland, 2018. Springer.
- A. Gliesch and M. Ritt. A generic approach to districting with diameter or center-based objectives. In M. López-Ibáñez, editor, *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 249–257, Prague, Czech Republic, July 2019. ACM.
- A. Gliesch, M. Ritt, A. H. S. Cruz, and M. C. O. Moreira. A hybrid heuristic for districting problems with routing criteria. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–9, Glasgow, United Kingdom, July 2020. IEEE.
- A. Gliesch, M. Ritt, A. H. S. Cruz, and M. C. O. Moreira. A heuristic algorithm for districting problems with similarity constraints. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, Glasgow, United Kingdom, July 2020. IEEE.

Abstract. Districting is the general problem of partitioning a set of geographic units into clusters, called districts. Districts must be geometrically compact, contiguous, and mutually balanced with respect to attributes of the units. In this chapter we propose a single-objective heuristic that can solve the three most common problems in districting: the p-Median, p-Center and Diameter Districting Problems with multiple balancing attributes, and can be extended to consider additional criteria. In a multi-start way our heuristic iteratively constructs greedy solutions and improves them by a hybrid approach that alternates between optimizing compactness and satisfying bal-

ancing constraints. We further propose dynamic data structures to efficiently evaluate candidate solutions under small modifications, which we show to be independently effective and applicable to other methods. Experimentally our heuristic is competitive when compared to approaches from the literature which are specific to each objective, and improves best known solutions in many cases. Finally, in separate analyses we extend our heuristic to include two common domain-specific districting criteria, similarity to existing solutions and routing costs, and in experiments show that it can effectively handle these variants with few modifications.

3.1 Introduction

Because districting has been used in such distinct applications ranging from land allocation to electoral districting, as we saw in Chapter 2, problem formulations vary depending on the specific requirements of each domain. Ultimately, the combination of a particular compactness measure among a wide range of choices, the way balancing is handled, and the set of additional domain-specific criteria make each problem in the literature unique from a computational standpoint. This leads to solution methods being tailored to each problem, often to exploit familiar structures suitable to specific algorithms.

As a consequence, one observes a certain fragmentation in the districting literature: problems are studied independently and solution methods are developed anew without reuse of algorithmic components from other methods. This is despite many formulations sharing key elements which often make underlying problems computationally similar. Examples are common compactness functions (e.g. p-median or p-center) or difficult domain-specific requirements (e.g. routing), where it is likely that a method that is effective for one problem will also be for similar ones. Comparisons between methods from different research groups are also rare, and to our knowledge, as of writing the only two cases are from Validi and Buchanan [211] and Ríos-Mercado et al. [184].

Until now, few authors have studied districting as a standalone problem, independent of domain [113, 114, 160]. To quote Kalcsics et al. [113] in their review from 2005 about the districting literature, “the tendency to separate the model from the application and establish the model itself as a self-contained topic of research cannot be observed”. As a result, we see a comparative lack of baseline algorithms and standard instance sets with best known values which could be drawn upon to determine the effectiveness of a newly developed method, or to assess the difficulty of a new

problem variant. Contributing to this are that many implementations belong to private companies who are unwilling to open their source code, and that many authors consider only one or two instances as case studies, which leads to low statistical confidence when drawing conclusions. This creates a barrier of entry for new researchers, which are left with few options for testing a new method short of self-comparisons through component and ablation analyses, and artificial instance generation.

In this chapter we look at districting problems from an algorithmic, application-independent point of view. We propose a generic single-objective heuristic solver for problems following the general formulation given in Model (1.1), that can serve as a baseline solution for multiple problem variants. We have implemented it with the p-Median, p-Center and Diameter Districting Problems (PMDP, PCDP and DDP) in mind, since these are the most common variants and have been studied over several domains. Despite this, our method can be modified to consider all the compactness functions described in Section 2.3.2, and extended towards additional criteria.

Algorithmically, our method uses a multistart method which builds solutions by a greedy algorithm and improves them by an alternating search heuristic. This heuristic is novel in the literature, and draws upon approaches that have been effective in the past. It alternates between a tabu search to improve the objective, and a series of constrained tabu searches to improve constraint violations, which are embedded within a binary search. Our implementation uses data structures to efficiently recompute criteria such as compactness and connectivity under neighboring operations, which provide a significant speedup. These data structures are independent of our heuristic, and could be included in other neighborhood search methods.

We have tuned the parameters of our method experimentally, and compared it to a number of existing approaches from the literature for the PMDP, PCDP and DDP. Results show that, despite being a generic solution, it produces competitive results for all problem variants, and in most cases improves best-known values for instances from the literature.

Finally, to base our claim on the extendability of our method, as proofs-of-concept we have further extended it to include routing and similarity criteria, two common requirements, and provide thorough experimental analyses on the resulting methods.

This chapter is organized as follows. In Section 3.2 we review the districting formulation we consider in this chapter, and give some mathematical notation for the next sections. In Section 3.3 and subsections therein we give a detailed description of our heuristic, including our specific approaches for the p-median, p-center and diameter objectives, for balancing solutions, and the dynamic data structures for recomputing

neighbor moves. Next, in Section 3.4 we introduce the domain and problem instances we used in our experiments. In Section 3.5 we report on computational experiments where we calibrate the parameters of our method, and compare it to other approaches in the literature for the PMDP, PCDP and DDP. Then, in Sections 3.6 and 3.7 we show how we extended our heuristic to include routing and similarity requirements, respectively. We conclude in Section 3.8.

3.2 Problem definition and notation

We consider the following domain-independent districting problem, which we presented originally in Chapter 1:

$$\begin{aligned}
&\textbf{minimize}_{S\in\mathcal{S}} && C(S) && && (3.1a)\\
&\textbf{subject to} && L_a \leqslant w_a(S_i) \leqslant U_a, && \forall i \in P, a \in A, && (3.1b)\\
& && G[S_i] \text{ is connected}, && \forall i \in P. && (3.1c)
\end{aligned}$$

From the set $\mathcal{S}$ of possible solutions, each defined as a p-partition $S = (S_1, \ldots, S_p)$ of input graph G, the problem seeks a solution S minimizing some compactness function $C(S)$, such that S is i) *balanced*, i.e. for each attribute identifier $a \in A$ and district S_i the total attribute value $w_a(S_i) = \sum_{u\in S_i} w_a^u$ of S_i with respect to a is between lower and upper bounds L_a and U_a, and ii) *connected*, i.e. every district S_i is a connected subset of G. Any solution satisfying these two constraints is considered *feasible*.

In this chapter we focus particularly on the three most common districting variants, which we introduced in Chapter 2: the p-Median, p-Center and Diameter Districting Problems (PMDP, PCDP and DDP, respectively), with multiple balancing attributes. These problems differ in their definition of the compactness function $C(S)$, which are:

$$C_{pm}(S) = \sum_{i\in P} \min_{c\in S_i} \sum_{j\in S_i} d_{jc} \tag{3.2}$$

for the p-median, i.e. the sum, over all districts, of the minimum sum of distances from the district units to a center c^{pm};

$$C_{pc}(S) = \max_{i\in P} \min_{c\in S_i} \max_{j\in S_i} d_{jc} \tag{3.3}$$

for the p-center, i.e. the maximum, over all districts, of the min-max distance from

Algorithm 2 Main loop of the proposed hybrid heuristic.

```
1:  S_best ← ∅
2:  repeat
3:      S ← selectInitialSolution()
4:      T = {S}
5:      for i ∈ [A_max] do
6:          S′ ← optimizeCompactness(S)
7:          S′ ← optimizeBalance(S′, S)
8:          if B(S′) > 0 or S′ ∈ T then
9:              break
10:         S ← S′
11:         T ← T ∪ {S}
12:     S_best ← argmin{BC(S_best), BC(S)}
13: until time limit reached
14: return S_best
```

any unit in the district to a center c^{pc}; and

$$C_{diam}(S) = \max_{i \in P} \max_{j,k \in S_i} d_{jk} \tag{3.4}$$

for the diameter, i.e. the maximum, over all districts, of the maximum pair distance between units.

Our method can handle either choice of compactness as an input parameter. We assume $d \in \mathbb{R}^{V \times V}$ is a pairwise distance matrix between the basic units, also given by the problem input.

Finally, we review the notion of *imbalance*, which we have introduced in Section 2.3.1. The imbalance $b_a(S_i) = \max\{0, w_a(S_i) - U_a, L_a - w_a(S_i)\}$ of district S_i with respect to attribute $a \in A$ is given by the magnitude of the violation of constraints (1.1b), and the imbalance $B(S) = \sum_{a \in A} \sum_{i \in P} b_a(S_i)$ of a solution S is given by the sum of individual imbalances for all districts and attributes. By this definition, both b and B return values greater than zero for infeasible solutions, and zero for feasible solutions.

3.3 Proposed algorithm

Algorithm 2 outlines the main loop of our proposed algorithm. Repeatedly, in a multistart way it constructs a solution using a randomized greedy algorithm (line 3), and improves it by a hybrid alternating search heuristic (lines 6–7), returning at the end the best solution S_{best} found over all repetitions. We use by default a time limit

as a termination criterion, but this could be substituted by e.g. a maximum number of multistart iterations. The best solution is selected with respect to the lexicographic fitness function $BC(S) = (B(S), C(S))$[1], as it gives preference to solutions which are closer to being feasible, when no feasible solution exists.

Our method is similar to GRASP [172], with two key differences. First, during construction we do not select candidates randomly from a restricted candidate list, as is standard, but select the best overall candidate in two stages by different criteria, and use filter lists to select only high-quality solutions. Second, instead of improving solutions by local search, we use a heuristic which alternates between two different algorithms to improve compactness and balance, in order. When improving balance, we restrict the search to solutions of compactness no larger than the incumbent's.

In each iteration of the alternating part (lines 5–11), to prevent cycling we store the current solution in a hash table $\mathcal{T}$, and move on to the next multistart iteration if cycling is detected (lines 8-9). Since the total number of intermediate solutions is expected to be small (less than 1000, on average), and cycling is rare, this is not performance- nor memory-critical. We also stop when the solution is still infeasible after an attempt to balance it (line 8), since it is unlikely that it will become balanced in a later iteration, or after a maximum number A_{max} of alternations. We use parameter A_{max} to avoid alternating indefinitely between low and high compactness values without a global improvement. In practice this does not happen often, however, and thus in our implementation we have fixed $A_{max} = 100$ as a fallback.

In the following subsections we present the components of our heuristic in detail. Section 3.3.1 explains the construction and filtering of new solutions through procedure selectInitialSolution. Then, in Section 3.3.2 we outline the general idea behind the alternating heuristic, and in Sections 3.3.3 and 3.3.4 we explain procedures optimizeBalance and optimizeCompactness. Last, in Section 3.3.5 we describe our data structures to efficiently update balance, connectivity and compactness dynamically under neighborhood search.

3.3.1 Initial solutions

Initial solutions are constructed in two steps. First, we use a dispersion heuristic to select p initial units $(s_1, \ldots, s_p)$, which will serve as seeds for each district. Then, we grow districts from these seeds by iteratively assigning to them a greedily-selected unassigned unit on their boundary, until the solution is complete. Algorithm 3 out-

[1] Given tuples $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ of $n > 1$, we say that $x < y$ *lexicographically* iff $x_1 < y_1 \vee (x_1 = y_1 \wedge (x_2, \ldots, x_n) < (y_2, \ldots, y_n))$. If $n = 1$, then $x < y$ iff $x_1 < y_1$.

Algorithm 3 Constructive heuristic for generating new solutions.

procedure GREEDYCONSTRUCTIVE
  $s_1 \leftarrow$ a randomly selected unit from V
  for $i = 2, \ldots, p$ **do**
    pick random subset $V' \subseteq V \setminus \{s_1, \ldots, s_{i-1}\}$ such that $|V'| = \sqrt{n}$
    $s_i \leftarrow \operatorname{argmax}_{u \in V'} \min_{j \in [i-1]} d_{us_j}$
  $S_i \leftarrow \{s_i\}\ \forall i \in P$
  for $i \in P$ **do**
    $c_i \leftarrow \operatorname{argmin}\{C(S_i \cup \{j\}) \mid j \in \partial(S_i)\}$
  while S is not complete **do**
    $k \leftarrow \operatorname{argmin}\{\sum_{a \in A} w_a(S_k) \mid k \in P\}$
    $S_k \leftarrow S_k \cup c_k$
    for $i \in P$ s.t. $c_i = c_k$ **do**
      $c_i \leftarrow \operatorname{argmin}\{C(S_i \cup \{j\}) \mid j \in \partial(S_i)\}$
  return S

lines the method.

The first step is a modification of the greedy constructive heuristic proposed by Erkut et al. [56]. It was originally developed to solve the dispersion problem of finding p elements from a set of points on the plane such that the minimum distance between any two elements is maximized. The first seed s_1 is selected uniformly at random from V. Then, for each $i = 2, \ldots, p$ in increasing order, we pick a subset $V' \subseteq V \setminus \{s_1, \ldots, s_{i-1}\}$ of size $\sqrt{n}$ uniformly at random, and define the i-th seed as $s_i = \operatorname{argmax}_{u \in V'} \min_{j \in [i-1]} d_{us_j}$. In other words, we select repeatedly the unit from V' that maximizes the minimum distance to the previously chosen units. Ríos-Mercado and Escalante [180] used a similar strategy, with the difference that seeds s_i are chosen from $V' = V \setminus \{s_1, \ldots, s_{i-1}\}$. We decided to use a smaller, randomly selected V' because, with a time complexity of $\Theta(p^2|V'|)$, the method was too slow for larger instances when $|V'| = O(n)$. Our approach also has the advantage of increased variability and, in practice, we have found no loss in average solution quality.

The second step starts from partial districts $S_i = \{s_i\} \mid \forall i \in P$. We maintain in memory, for each district S_i, the candidate unit $c_i \in \partial(S_i)$ that minimizes $C(S_i \cup \{c_i\})$. (Recall that $\partial(S_i)$ represents the boundary of district S_i, as defined in Section 2.4.1.4.) If multiple such units exist, one is chosen randomly. At first, we compute c_i for every district by iterating over each boundary $\partial(S_i)$. Then, we repeatedly select the district S_j with smallest total attribute $\sum_{a \in A} w_a(S_j)$ and assign $S := S[c_j \to j]$, until S is complete. We recompute c_i for all districts S_i with $c_i = c_j$ after every assignment, because the boundaries of these districts change, and so the set of possible entry candidates for them also does. There are on average only $O(1)$ such districts, however, since G

is planar and thus has an average degree no larger than 6. To maintain dynamically the candidate costs c is significantly more efficient than the alternative of recomputing them at each iteration. Note that because only boundary units are considered, connectivity is preserved.

This strategy is greedy in the way that it selects a candidate with smallest increase in compactness C for each district. This steers the construction towards more compact initial solutions, as opposed to a different strategy that might, for example, try to find solutions that are as balanced as possible by minimizing B. The rationale is that, in practice, highly imbalanced solutions can be made feasible quickly by our balancing heuristic, while it is substantially harder to improve compactness. Moreover, early experiments showed that the quality of solutions obtained after neighborhood search highly depends on the compactness of the initial solutions. Still, by selecting the district with minimum total attribute at each step, we ensure a certain balancing and prevent districts in denser regions from growing too large.

3.3.1.1 Filtering

Since the alternating improvement phase is much more time-consuming than the construction phase, we filter out low-quality initial solutions to focus our computational effort in more promising ones. At each multistart iteration we construct p randomized greedy solutions and add them to a pool $\mathcal{P}$ of possible initial solutions. We then select the solution $S \in \mathcal{P}$ with minimum $CB(S) = (B(S), C(S))$, compared lexicographically, to be improved next, and remove S from $\mathcal{P}$. Set $\mathcal{P}$ is carried over to the next iteration. To limit memory usage, we limit $|\mathcal{P}|$ to $2p$ and discard the $\max\{0, |\mathcal{P}| - 2p\}$ worst solutions with respect to CB at each iteration. Here we are greedy with respect to CB, as opposed to BC, since as we mentioned previously, empirically the initial imbalance of a solution was not a good predictor of how difficult it is to balance it in the alternating phase.

3.3.2 Optimizing solutions by alternating search

To optimize different criteria alternatingly is not a completely new approach in districting [28, 128]. The main idea is to focus the effort on optimizing one objective at a time through a custom neighborhood that only considers moves with respect to that objective, while (to some extent) ignoring others. A common alternative would be to perform a single search using a composite fitness function that assigns weights to each objective, as is done e.g. by Bozkaya et al. [22] and Ríos-Mercado and Escalante

[180]. The main drawback of this alternative is that it is difficult to find weights that work well for all instances.

Both alternating procedures optimizeBalance and optimizeCompactness are based on tabu search. To improve compactness, we use a tabu search over a specific neighborhood depending on the compactness function being considered (e.g. p-median, p-center or diameter). In Sections 3.3.4.1 to 3.3.4.3 we give full descriptions of these neighborhoods. To improve balance, we execute a series of tabu searches on neighborhoods bounded by a maximum increase to compactness. Let S be the incumbent before optimizing for compactness, and S' after it (see line 7 in Algorithm 2). Specifically, using upper and lower bounds $C(S)$ and $C(S')$ we apply a binary search to find the smallest $c^* \in [C(S'), C(S)]$ for which a tabu search aimed at reducing imbalance and bounded by $C(S') \leqslant c^*$ obtains a feasible solution. We explain the full procedure in detail in Section 3.3.3.

Note that, although we use tabu search, this alternating strategy is agnostic of the choice of optimization algorithm for balancing or improving compactness, and other heuristic could be used instead, or even an exact method.

The neighborhoods used by our tabu searches are based on shift and swap moves. We have introduced tabu search as well as shifts and swaps in Section 2.4.2.3, and in the following we review them in the context of our heuristic.

A *shift* $u \to i$ is an operator over solution $S = (S_1, \ldots, S_p)$ which moves unit $u \in S(u)$ to district S_i. We denote by $S[u \to i]$ solution S after applying shift $u \to i$. (Recall that $S(u)$ is the district that u is currently assigned to, i.e. $u \in S(u)$). Similarly, a swap $u \leftrightarrow v$ exchanges the assigned districts of units u and v in S, and we denote by $S[u \leftrightarrow v]$ the resulting solution after swapping u and v.

Tabu search is a non-monotone local search proposed by Glover [85]. It iteratively executes the best neighboring operator on the current solution, with respect to some fitness function. In our case, the fitness function depends on the criterion being optimized – either compactness or balance. If multiple best candidates exist for a neighboring move, one is selected randomly. To avoid cycling, some operators are declared "tabu" after a neighboring move, and cannot be selected again for t iterations, where t is a parameter called tabu *tenure*. Specifically, after executing a shift $u \to i$ we mark all operators in $\{u \to j \mid j \in P\}$ and $\{u \leftrightarrow v \mid v \in V\}$ as tabu. Similarly, after swap $u \leftrightarrow v$ we mark as tabu all operators in $\{w \to j \mid j \in P, w \in \{u, v\}\}$ and $\{w^1 \leftrightarrow w^2 \mid w^1 \in V, w^2 \in \{u, v\}\}$. Moreover, we discard all moves which violate district connectivity, since they are infeasible. In Section 3.3.5.1 we show how these moves can be detected efficiently. Each tabu search terminates after I_{max} iterations without

Algorithm 4 Improving the balance of solutions through a series of tabu searches.

```
1:  procedure IMPROVEBALANCE(S, S_u)
2:      U ← C(S_u)
3:      if B(S_u) > 0 then
4:          U ← 2U
5:      L ← C(S)
6:      R ← S
7:      while L < U ∧ B(S) ≠ 0 do
8:          S' ← TS_b(S, (U + L)/2)
9:          if B(S') = 0 then
10:             R ← S'
11:             U ← C(R) − d_min
12:         else
13:             S ← S'
14:             L ← (L + U)/2 + d_min
15:     return R
```

improvement, where I_{max} is a parameter, and returns the intermediate solution of best fitness.

3.3.3 Optimizing balance

To balance solutions we use tabu searches over restricted neighborhoods, which are defined by setting an upper limit $\bar{c}$ such that no move can lead to a solution whose compactness exceeds $\bar{c}$. We use binary search to find the minimum value of $\bar{c}$ with which a feasible solution can be found. We outline this in Algorithm 4.

We start with initial solution S, which is likely imbalanced, and the current best solution S^u. We assume that $C(S) \leqslant C(S^u)$. Let the initial lower and upper limits of the binary search be $\mathcal{L} = C(S)$ and $\mathcal{U} = C(S^u)$. If S^u is not feasible, however, which can happen in the first alternation, we set $\mathcal{U} = d_{max}$ as a fallback, where $d_{max} = \max_{i,j \in V} d_{ij}$. Let also R be the solution to be returned, with $R = S$ initially. We then binary search for the smallest $\bar{c}^* \in [\mathcal{L}, \mathcal{U})$ such that the solution resulting from a tabu search $TS_b(S, \bar{c}^*)$ is balanced. We explain method TS_b in the next section. At each step of the binary search, we execute a constrained tabu search $TS_b(S, \bar{c})$ with maximum allowed compactness $\bar{c} = (\mathcal{U} + \mathcal{L})/2$ (line 8). If the resulting solution S' is balanced, independently of its compactness value, we update return solution $R \leftarrow S'$ and upper limit $\mathcal{U} = C(R) - d_{min}$ (line 11), where $d_{min} = \min_{i,j \in V} d_{uv}$ is the smallest distance between any two units. Otherwise, if S' is not balanced, we update the current solution S and the lower limit $\mathcal{L} = (\mathcal{U} + \mathcal{L})/2 + d_{min}$ (line 14). We use step size d_{min} since we found it empirically to be a good compromise between precision and

the total number of calls to TS_b.

3.3.3.1 Balancing tabu search

Given a solution S to be balanced and an upper limit $\bar{c} \geqslant C(S)$, each tabu search $TS_b : \mathcal{S} \times \mathbb{R} \to \mathcal{S}$ returns a solution S' such that $C(S') \leqslant \bar{c}$ and $B(S') \leqslant B(S)$. Tabu search TS_b uses shifts and swap moves. Specifically, it considers neighborhoods

$$N^b_{shift}(S, \bar{c}) = \left\{ m \in N_{shift}(S) \mid C(S[m]) \leqslant \bar{c} \right\} \tag{3.5}$$

and

$$N^b_{swap}(S, \bar{c}) = \left\{ m \in N_{swap}(S) \mid C(S[m]) \leqslant \bar{c} \right\}, \tag{3.6}$$

in a VNS-like fashion. Here, we use the definition of the standard shift and swap neighborhoods N_{shift} and N_{swap} from Section 2.4.2.3. At each iteration, we first look for non-tabu shifts $m \in N^b_{shift}$ which minimize $BC(S[m])$ and that are improving, i.e. $B(S[m]) < B(S)$. If such a move is found, we apply it immediately and move on to the next iteration; otherwise, we search for swaps in N^b_{swap} under the same criteria. We then apply the best move found with respect to BC. By using BC here we aim at moves with minimum imbalance, and use minimum compactness as a tie-breaker.

This VNS-like strategy which considers first shifts, then swaps in order has been generally effective for other grouping problems [25, 107] and, as we have argued in Section 2.4.2.3, is effective since empirically there are many more possible swaps than there are shifts, since $|N^b_{shift}| \approx p\sqrt{n/p}$ and $|N^b_{swap}| \approx n$. In fact, preliminary experiments showed that in over 80% of cases where improving moves are found, they are shifts. Besides the usual stopping criterion of I_{max} consecutive iterations, search TS_b also stops as soon as the incumbent is balanced, i.e. when $B(S) = 0$.

The effectiveness of binary search here is motivated by the fact that $B(TS_b(S, c))$ is expected to be monotonically decreasing as $\bar{c}$ grows: given some $\bar{c}_1$ and $\bar{c}_2$ such that $\bar{c}_1 < \bar{c}_2$, the neighborhood explored by search $TS_b(S, \bar{c}_2)$ is a superset of the one explored by $TS_b(S, \bar{c}_1)$. Thus, $TS_b(S, \bar{c}_2)$ typically has more available improving neighboring moves and should be more likely to reach better solutions. Of course, since TS_b is a heuristic algorithm this may not always be the case, but empirically we find that it is nearly always so.

3.3.4 Optimizing compactness

We optimize compactness with a single tabu search. It uses a custom neighborhood for each choice of compactness function. This is because it is common for compactness functions to have optimization landscapes where the majority of moves do not yield a change in the objective, thus making it wasteful to explore them fully. This is especially the case for functions like the p-center and diameter, which aggregate local compactness values by the maximum and thus "hide" intermediate values.

In this section we detail the neighborhoods used for the p-median, p-center and diameter compactness functions. Our algorithm is agnostic of the choice of compactness measure, however, and can be extended to other measures by simply defining a search neighborhood.

3.3.4.1 Optimizing diameter

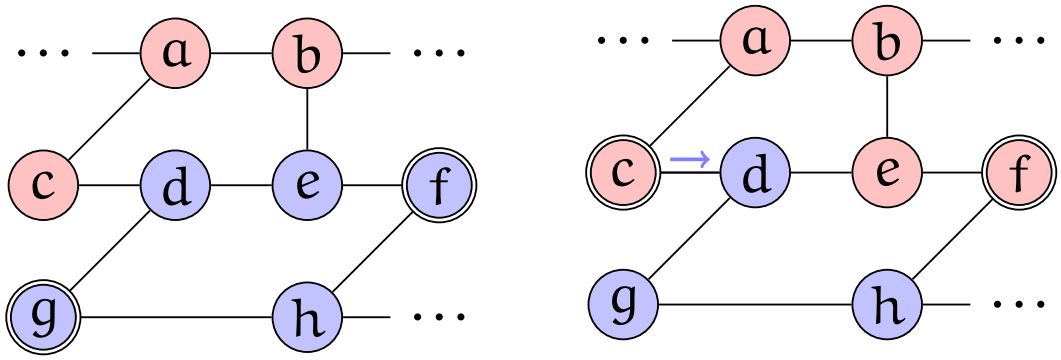
When optimizing the diameter, we consider only moves in the neighborhood

$$N_{dm}(S) = \{u \to i \in N_{shift}(S) \mid u \in K^{dm}(S)\},$$

where the set $K^{dm}(S) = \{u \in V \mid \exists v \in S(u),\, d_{uv} = C(S)\}$ contains *diameter-critical* units which are incident to some maximum diameter of S. These are the only moves which have the potential to improve $C(S)$

Note that if $|K^{dm}(S)| > 2$ even shifting a unit from $K^{dm}(S)$ might not change C, since there will still exist another pair of units with equal distance (except if a unit is incident to multiple diameters). This situation happens frequently when d is Euclidean and unit locations are regularly distributed on the plane, such as in a grid, for example. It is clear, however, that reducing the cardinality of K^{dm} is still an improvement. Thus, we rank moves accordingly with a lexicographic objective function $C'(S) = (C(S), |K^{dm}(S)|)$.

A special situation occurs when no units in $K^{dm}(S)$ are on the boundary of another district, meaning that $N_{dm}(S) = \emptyset$ and thus no moves can be made. This represents a plateau that is particularly difficult to escape, since it requires at least as many moves as the length of the shortest path from a unit in $K^{dm}(S)$ to the boundary of some district. In practice, it is very unlikely that this specific set of moves will be performed during the balancing tabu search TS_b. Therefore, when this kind of plateau is reached, we attempt to escape it as follows. We search G for a path π of smallest Euclidean distance starting from any unit $u \in K^{dm}(S)$ and ending at a unit $v \in \partial S(u)$. We then



(a) Before "shortest path escape".

(b) After "shortest path escape".

,

Figure 3.1: The "shortest path escape" process. We use "$\cdots$" to mean that the solution continues with other districts at that point. In Figure 3.1a, the two farthest units g and f are not on the boundary of any other district (i.e. $N_{dm} = \emptyset$). We therefore search for the shortest distance path from either g or f to the boundary of ■, and find feb. We then assign all units in feb to ■, obtaining the solution in Figure 3.1b. Notice that, after this process, the search is able to continue, since $N_{dm} = \{c \rightarrow ■\}$.

assign all units in π, including u but not v, to district $S(v)$. If this leads the original district containing u to become disconnected, however, we stop and give up on this attempt. Otherwise, reassigning unit u forces a change in K^{dm}, and consequently in C'. An exception is when u remains in K^{dm} paired to the same number of units and distances as before, but this is rare. We repeat this procedure at most sp_{max} times, or until $|N_{dm}| \neq \emptyset$. In our implementation we have fixed $sp_{max} = 25$, since this value was more effective in a calibration performed in an earlier version of our algorithm [81]. Using a standard shortest paths algorithm that orders active units with a priority queue, finding path π takes, on average, $O(n/p \log n/p)$ steps. Figure 3.1 illustrates the process, which call "shortest path escape".

3.3.4.2 Optimizing p-center

To optimize the p-center objective we use two different neighborhoods. To explain them, let us first define the set of p-*center-critical* units $K^{pc}_u(S) = \{u \in V \mid d_{uk} = C(S), k = c^{pc}(S(u))\}$ of solution S as the set of units whose distance to their district center is equal to $C(S)$, and the set of p-center-critical districts $K^{pc}_d(S) = \{S(u) \mid u \in K_u(S)\}$ of S as the set of districts that contain a critical unit. Here, as we have defined in Section 2.3.2, $c^{pc}(S_i) = \text{argmin}_{c \in S_i} \max_{j \in S_i} d_{jc}$ defines the p-center of a district S_i.

The first neighborhood we consider is

$$N^1_{pc} = \{u \to i \in N_{shift}(S) \mid u \in K^{pc}_u(S)\}.$$

It aims to shift critical units out of critical districts, in an attempt to reduce the local compactness of critical districts. This consequently reduces global compactness of the solution, since it depends on the critical districts.

The second neighborhood,

$$N^2_{pc} = \{u \to i \in N_{shift}(S) \mid S(u) \in K^{pc}_d(S), u \notin K^{pc}_u(S)\}\},$$

shifts non-critical units out of critical districts. This is to induce a change to the optimal district center c^{pc}, in such a way that it is moved closer to critical units and thus decreases the district's local compactness.

We explore N^1_{pc} and N^1_{pc} in order, similarly to the balancing tabu search TS_b. We first consider non-tabu moves $m \in N^1_{pc}$ which minimize $C(S[m])$. If no improving move exists for it, we then consider N^2_{pc} in the same manner. Neighborhood N^1_{pc} takes precedence here since, in practice, it is about a factor $\sqrt{n/p}$ smaller than N^2_{pc} and yields the best move in the majority of cases. Still, searching N^2_{pc} is nonetheless effective.

We have also experimented with a third neighborhood $N^3_{pc} = \{u \to i \in N_{shift}(S) \mid i \in K^{pc}_d(S)\}$ in which a critical district receives a unit which may become the new optimal district center, thus removing the critical status of the district's critical units. This neighborhood was seldom better than the two previous ones, yielding the best move in less than 1% of cases, on average, and is time-consuming to explore fully. Therefore, we have excluded N^3_{pc} from the final algorithm.

Like in K^{dm}, notice that if $|K^{pc}_u(S)| > 2$ shifting a critical unit will not reduce compactness C, since other critical pairs exist which induce it. Therefore, as we have done for the diameter we use a lexicographic objective $C'(S) = (C(S), |K^{pc}_u(S)|)$, such that moves which lead to smaller p-center-critical sets are preferred.

3.3.4.3 Optimizing p*-median*

Because the p-median function accounts for the distances to centers of not only some but all units, as it aggregates by the sum, to use a restricted neighborhood like N_{dm} or N_{pc} would disallow some improving moves. Therefore, we optimize the

p-median using the full shift neighborhood N_{shift}, as defined in Section 2.4.2.3:

$$N^{pm}(S) = N_{shift}(S) = \{u \to i \mid i \in P, u \in \partial(S_i), S(u) \setminus \{u\} \text{ is connected}\}.$$

We have also experimented with searching N_{swap}, but empirically it slowed down the search without any significant improvement in the quality of resulting solutions.

Finally, we note that for both p-median and p-center we compute the value of each candidate move optimally. This contrasts with the common strategy to approximate optimal values by assuming the centers do not change, i.e. to compute the compactness of candidates with the current centers fixed, and only recompute optimal centers when actually applying the neighboring move. Using optimal rather than approximate values for candidates leads to better informed neighboring decisions, since we can account for moves which improve compactness through changing optimal district centers. In the next section, we show how we dynamically and efficiently compute these optimal values.

3.3.5 Dynamic updates for local search candidates

During neighborhood search or a constructive algorithm which examines many candidate moves, several values must be recomputed in order to ascertain the potential of each move. These include, for example, (dis)connectivity tests, or the expected imbalance and compactness of the resulting solution. If implemented in a naïve way, i.e. without taking into account pre-computed information from the current solution, these computations become a bottleneck of high computational complexity that dominates the running time of the algorithm. This is particularly evident in larger instance sizes, or large neighborhoods such as N_{swap}.

In this section we discuss relevant data structures and speed-up techniques we have used in our implementation of these components to avoid this effect. We analyze their theoretical runtime complexity in contrast to a naïve approach, and discuss how they might be improved.

3.3.5.1 Dynamic connectivity

A shift or a swap operation may break the connectivity of a solution if a moved unit was an articulation vertex of the subgraph induced by its previous district. The straightforward way to test connectivity loss is to perform a depth- or breadth-first

search on the district from which a unit was removed. This takes $O(n/p)$ time, assuming an average district size of n/p. The majority of authors do not discuss how disconnectivity tests are implemented.

We propose a better approach that maintains a flag array indicating whether each unit is an articulation vertex in its district. For each district, this can be updated in $O(n/p)$ average time with the algorithm of Tarjan [206]. Since updating is only required after a neighboring move has been made, and only for the 2 districts involved in the move (note that both shift and swap moves only change 2 districts), each move incurs an update cost of $O(2n/p) = O(n/p)$. Therefore, given a neighborhood of size k, the amortized update cost per neighbor is $O(n/(kp))$. For N_{shift}, for example, since $|N_{shift}| \approx p\sqrt{n/p}$ this cost becomes $O(n/(p^2\sqrt{n/p}))$, while for N_{swap}, since $|N_{swap}| \approx n$ the amortized update cost is constant as $O(n/(np)) = O(1/p) = O(1)$.

When considering a candidate shift $u \rightarrow i$ we simply test whether i is an articulation node: if yes, then shift $u \rightarrow i$ leads to a disconnected solution and should be discarded. Since this only requires checking whether a flag is set, it is done in $O(1)$. Similarly, for a swap $u \leftrightarrow v$ we discard the move if either u or v is an articulation node. Note that this may inadvertently discard valid moves in the special case where $G \cap (S(u) \setminus \{u\})$ is not connected, but $G \cap (S(u) \setminus \{u\} \cup \{v\})$ is. In practice, however, this situation rarely happens, and since we have not identified a method to test this condition in constant time, we have found that the performance gained by ignoring such cases compensates for the possible lost moves.

The approach above is simple to implement and effective in practice, since it has low amortized computational cost and ultimately becomes shadowed by other components of our method. Still, we note two approaches in the literature which have better time complexity. The first is the algorithm of Łącki and Sankowski [122] for decremental connectivity in planar graphs, as it allows both tests and updates in constant time. However, the algorithm has only been discussed in theoretical terms and, to our knowledge, no implementation of it exists. The second is the geo-graph algorithm of King et al. [118] developed specifically for districting, which has a time complexity of $O(\Delta(G))$, which amounts to $O(1)$ if G is planar. Unfortunately, the authors did not respond to our requests to share their implementation and, since the algorithm is not simple, it was outside our scope to reimplement it.

3.3.5.2 Dynamic imbalance

To compute the expected imbalance $B(S)$ of candidate solutions, for each district S_i we maintain in memory its total value $w_a(S_i)$ with respect to each attribute $a \in A$. These values are updated in constant time when a unit u is inserted or removed from district S_i by setting, respectively,

$$w_a(S_i) \leftarrow w_a(S_i \cup \{u\}), \qquad \forall a \in A, \tag{3.7}$$

$$w_a(S_i) \leftarrow w_a(S_i \setminus \{u\}), \qquad \forall a \in A, \tag{3.8}$$

where

$$w_a(S_i \cup \{u\}) = w_a(S_i) + w_u^a, \qquad \forall a \in A, \tag{3.9}$$

$$w_a(S_i \setminus \{u\}) = w_a(S_i) - w_u^a, \qquad \forall a \in A, \tag{3.10}$$

Given that individual imbalances $b_a(S_i)$ can be computed in constant time using $w_a(S_i)$ for $a \in A, i \in P$ (recall the definition of $b_a(S_i)$ from Section 3.2), the expected imbalance of a solution resulting from candidate shift $u \rightarrow i$ can be obtained also in constant time through

$$B(S[u \rightarrow i]) = B(S) + \sum_{a \in A} \Big(- b_a(S_i) + b_a(S_i \cup \{u\}) - b_a(S(u)) + b_a(S(u) \setminus \{u\}) \Big). \tag{3.11}$$

Similarly, the expected imbalance of a swap $u \leftrightarrow v$ is computed through

$$B(S[u \leftrightarrow v]) = B(S) + \sum_{a \in A} \Big(- b_a(S(u)) + b_a((S(u) \cup \{v\}) \setminus \{u\}) - b_a(S(v)) + b_a((S(v) \cup \{u\}) \setminus \{v\}) \Big). \tag{3.12}$$

3.3.5.3 Dynamic diameter compactness

When district S_i receives a unit, a trivial way to update its local diameter is to check the distance from the new unit to all units currently in S_i, in $O(|S_i|)$ time. However, if a unit is removed from S_i, without any cached information a brute-force method might check all pairwise distances in $O(|S_i|^2)$ time, which becomes a bottleneck. This approach is used by some authors [180].

One solution is to maintain a max-heap of all pairwise distances for each district, such that the critical pair is always on top. In this way, the optimal diameter for district S_i can be maintained in $O(|S_i| \log |S_i|)$ time. Using a lazy removal strategy for pairs which are no longer in S_i, at most $|S_i|$ insertions or removals will be made from the heap after each move. Empirically this complexity tends to be much lower, however, since the worst case only happens when the removed unit is the closest to every single other unit in the district, which is extremely rare. The downside to this approach is that it requires $O(n^2/p)$ memory over all districts, on average.

When distances are Euclidean and plane coordinates associated with each unit are available, which is the case of all instance sets we use, we propose a different approach. It uses the fact that the most distant pair of points of a point set on the plane must lie on that point set's convex hull [198]. We therefore maintain the convex hull of each district dynamically. Over each convex hull, we compute diameters using the rotating calipers algorithm of Shamos [198], which runs in $O(k)$ where k is the size of the convex hull. Har-Peled [96] showed that the convex hull of a set of n uniformly-distributed points (a premise which can be reasonably assumed in the instances we considered) has expected size $O(\log n)$, and so, assuming an average of n/p units per district, each diameter is computed in $O(\log n/p)$. Finally, the maximum diameter of a solution can be obtained by inspecting all p districts in $O(p)$, whose diameter values we also cache.

To compute convex hulls we use the monotone chain algorithm of Andrew [4], which runs in $O(k \log k)$ time for any set of k points or in $O(k)$ time if the points are already sorted lexicographically by their coordinates. We maintain the latter condition. Before neighborhood search we sort the list of units of every district by their coordinates, and keep them sorted after each neighboring move using a simple linear update of average cost $O(n/p)$, for both insertions and deletions. This lets us update diameters in $O(n/p)$ after a neighboring move (recall that shift and swap moves change at most 2 districts), which incurs an $O(n/(pk))$ amortized cost per candidate on a neighborhood of size k. This could be further improved using the algorithm of Overmars and van Leeuwen [164] which maintains convex hulls of k points in $O(\log^2 k)$ time per update, but we did not find any available implementations of this algorithm.

Using the strategies above, we compute the expected diameter of a candidate shift

$u \to i$ as

$$C_{dm}(S[u \to i]) = \max\left\{ \max_{\substack{j \in P \\ S_j \notin \{S(u), S_i\}}} C^{dm}_{S_j},\ \Delta^{dm}_{S_i},\ \Delta^{dm}_{S(u)} \right\}, \tag{3.13}$$

where

$$\Delta^{dm}_{S_i} = \max\{C^{dm}_{S_i}, \min_{v \in CH(S_i)} d_{uv}\}, \tag{3.14}$$

$$\Delta^{dm}_{S(u)} = \left[C^{dm}_{S(u)} = C_{dm}(S)\right]\ rc\big(CH(S(u) \setminus \{u\})\big), \tag{3.15}$$

are the expected local compactness values for S_i and $S(u)$, respectively. The first term of (3.13) accounts for the diameter of districts that are not included in the shift, and is obtained in $O(p)$ by taking the maximum of the local district diameters $C^{dm}_{S_j}$, $j \in P$, which are cached in memory. The second term, $\Delta^{dm}_{S_i}$, defines the new diameter of district S_i after the move and is computed by considering the distance between u and every point on S_i's convex hull $CH(S_i)$ in expected time $O(\log |S_i|)$. Finally, $\Delta^{dm}_{S(u)}$ defines the new diameter of district $S(u)$ after losing unit u. It is computed in $O(|S(u)|)$ time by re-computing $S(u)$'s convex hull without u and executing the rotating calipers algorithm rc on the result. Note that computing $\Delta^{dm}_{S(u)}$ is only needed if $S(u)$ is diameter-critical, since removing a unit from a district cannot increase its diameter. Hence, to encode this we multiply the third term by $[C^{dm}_{S(u)} = C_{dm}(S)]$[2], which short-circuits the rest of the expression if it is 0. Because there are p districts, this is expected to occur with probability $1/p$, on average, and so expected amortized cost of computing $\Delta^{dm}_{S(u)}$ is $O(|S(u)|/p)$. Assuming an average of n/p units per district, computing expressions (3.13) therefore has an expected amortized cost of $O(p + \log n/p + n/p^2)$ per shift.

This extends to swaps in the same manner. Given candidate swap $u \leftrightarrow v$, the expected diameter of the resulting solution is computed as

$$C_{dm}(S[u \to i]) = \max\left\{ \max_{\substack{j \in P \\ S_j \notin \{S(u), S(v)\}}} C^{dm}_{S_j},\ \Delta^{dm\,\prime}_{S(v)},\ \Delta^{dm\,\prime}_{S(u)} \right\}, \tag{3.16}$$

[2]For a given boolean expression b, Iverson's bracket operator $[b]$ returns 1 if b is true, and 0 otherwise.

where

$$\Delta^{dm}_{S(u)}{}' = \max\Big\{ \min_{w \in CH(S(u)\setminus\{u\})} d_{vw}, \quad \big[C_{dm}(S) \in \{C^{dm}_{S(u)}, C^{dm}_{S(v)}\}\big]\ rc\big(CH(S(u)\setminus\{u\})\big) \Big\}, \tag{3.17}$$

and vice-versa for $\Delta^{dm}_{S(v)}{}'$. The asymptotic costs calculation is similar as for shifts, and we omit it here. Note that we test whether either $S(u)$ or $S(v)$ was critical – in other words, whether the global diameter may have decreased – for both $\Delta^{dm}_{S(u)}{}'$ and $\Delta^{dm}_{S(v)}{}'$, since it is possible e.g. that $S(v)\setminus\{v\}$ will be diameter-critical in $S[u\leftrightarrow v]$ even though $S(u)$ (and not $S(v)$) was critical in S.

3.3.5.4 Dynamic p-median compactness

To compute the p-median compactness dynamically we maintain two sets of values: the optimal p-median value $C^{pm}_{S_i}$ for each district $i \in P$, and the local compactness δ^{pm}_{ui} of district S_i if u were made district center, for every $u \in V$ and $i \in P$. Upon neighboring moves we update these values in $O(n)$ as follows. When unit u is to be inserted into district S_i, we set

$$\delta^{pm}_{vi} \leftarrow \delta^{pm}_{vi} + d_{uv} \qquad \forall v \in V, \tag{3.18}$$

$$C^{pm}_{S_i} \leftarrow \min\Big\{ \delta^{pm}_{vi} \mid v \in S_i \cup \{u\} \Big\}, \tag{3.19}$$

and if u is to be removed from S_i, we set

$$\delta^{pm}_{vi} \leftarrow \delta^{pm}_{vi} - d_{uv} \qquad \forall v \in V, \tag{3.20}$$

$$C^{pm}_{S_i} \leftarrow \min\Big\{ \delta^{pm}_{vi} \mid v \in S_i \setminus \{u\} \Big\}. \tag{3.21}$$

We note that this could be improved by updating δ^{pm}_{vi} only for $v \in S_i \cup \partial(S_i)$ (i.e. only if v is liable to be part of a feasible neighboring move involving S_i, either by being in S_i or on its boundary). Since, as we have argued in Section 2.4.2.3, the boundary of a set of n uniformly distributed points has a size of roughly $\sqrt{n}$, this would lead to an average update cost of $O(\sqrt{n} + n/p)$. However, this would require a data structure which dynamically maintains ∂, which we did not use in our implementation.

When considering candidate moves, using C^{pm} and δ^{pm} we can then compute the

p-median value of a candidate shift $u \to i$ in $O(n/p)$ as

$$C_{pm}(S[u \to i]) = \left(\sum_{\substack{j \in P \\ S_j \notin \{S(u), S_i\}}} C^{pm}_{S_j} \right) + \Delta^{pm}_{S_i} + \Delta^{pm}_{S(u)}, \tag{3.22}$$

where

$$\Delta^{pm}_{S_i} = \min \left\{ \delta^{pm}_{vi} + d_{uv} \mid v \in S_i \cup \{u\} \right\}, \tag{3.23}$$

$$\Delta_{S(u)} = \min \left\{ \delta^{pm}_{vj} - d_{uv} \mid j : S_j = S(u),\ v \in S(u) \setminus \{u\} \right\} \tag{3.24}$$

are the expected changes to the compactness values of districts i and $S(u)$, respectively. Similarly, we compute the expected value of a swap $u \leftrightarrow v$ also in $O(n/p)$ as

$$C_{pm}(S[u \leftrightarrow v]) = \left(\sum_{\substack{j \in P \\ S_j \notin \{S(u), S(v)\}}} C^{pm}_{S_i} \right) + \Delta^{pm\ \prime}_{S(u)} + \Delta^{pm\ \prime}_{S(v)}, \tag{3.25}$$

where

$$\Delta^{pm\ \prime}_{S(u)} = \min \left\{ \delta^{pm}_{wj} + d_{vj} - d_{uj} \mid j : S_j = S(u),\ w \in \big(S(u) \cup \{v\}\big) \setminus \{u\} \right\} \tag{3.26}$$

and vice-versa for $\Delta^{pm\ \prime}_{S(v)}$.

Given a neighborhood of size k, updates to C^{pm} and δ^{pm} take $O(n/k)$ amortized time per candidate, while computing candidate costs values takes $O(n/p)$ per candidate. For both N_{shift} and N_{swap} this ultimately incurs an amortized cost per candidate of $O(n/p)$, as the latter costs are the bottleneck. This is significantly more efficient than a brute-force approach which might recompute the local compactness of both districts in $O((n/p)^2)$ time.

3.3.5.5 Dynamic p-center compactness

The idea behind dynamic p-centers is similar to that of dynamic p-median. We maintain three sets of values: the optimal p-center value $C^{pc}_{S_i}$ for each district $i \in P$ and, for each $u \in V, i \in P$, the largest and second-largest distances δ^{pc}_{ui} and γ^{pc}_{ui} from u to any unit in i. Values δ^{pc}_{ui} can be interpreted as the compactness of district S_i if u were made its center, and γ^{pc}_{ui} as the compactness of S_i if u were made center while the unit incident to δ^{pc}_{ui} is removed.

Upon neighboring moves, we update C^{pc}, δ^{pc} and γ^{pc} in $O(n^2/p)$ as follows. When unit u is to be inserted into district S_i we set, in order,

$$\gamma^{pc}_{vi} \leftarrow \text{sec}\left(\delta^{pc}_{vi}, \gamma^{pc}_{vi}, d_{vu}\right) \qquad \forall v \in V, \tag{3.27}$$

$$\delta^{pc}_{vi} \leftarrow \max\left\{\delta^{pc}_{vi}, d_{vu}\right\} \qquad \forall v \in V, \tag{3.28}$$

$$C^{pc}_{S_i} \leftarrow \min\left\{\delta^{pc}_{vi} \mid v \in S_i \cup \{u\}\right\}, \tag{3.29}$$

where $\text{sec}(S)$ returns the second-largest value of list S. When unit u is to be removed from district S_i, we set

$$\delta^{pc}_{vi} \leftarrow \left\{ \begin{array}{ll} \gamma^{pc}_{vi}, & \text{if } \delta^{pc}_{vi} = d_{uv} \\ \delta^{pc}_{vi}, & \text{otherwise} \end{array} \right\} \qquad \forall v \in V, \tag{3.30}$$

$$\gamma^{pc}_{vi} \leftarrow \left\{ \begin{array}{ll} \gamma^{pc}_{vi}, & \text{if } \gamma^{pc}_{vi} > d_{uv} \\ \text{sec}\left(d_{vw} \mid w \in S_i \setminus \{u\}\right), & \text{otherwise} \end{array} \right\} \qquad \forall v \in V, \tag{3.31}$$

$$C^{pc}_{S_i} \leftarrow \min\left\{\delta^{pc}_{vi} \mid v \in S_i \setminus \{u\}\right\}. \tag{3.32}$$

When considering candidate moves, using the three cached values we can compute the p-center value of candidate shift $u \rightarrow i$ in $O(n/p)$ as

$$C_{pc}(S[u \rightarrow i]) = \max\left\{ \max_{\substack{j \in P \\ S_j \notin \{S(u), S_i\}}} C^{pc}_{S_j}, \Delta^{pc}_{S_i}, \Delta^{pc}_{S(u)} \right\}, \tag{3.33}$$

where

$$\Delta^{pc}_{S_i} = \min\left\{\max\left\{ d_{uv}, \delta^{pc}_{vi} \right\} \mid v \in S_i \cup \{u\}\right\}, \tag{3.34}$$

$$\Delta^{pc}_{S(u)} = \min\left\{ \left[d_{uv} = \delta^{pc}_{vj}\right]\gamma^{pc}_{vj} + \left[d_{uv} \neq \delta^{pc}_{vj}\right]\delta^{pc}_{vj} \mid j : S_j = S(u), \right. \tag{3.35}$$

$$\left. v \in S(u) \setminus \{u\}\right\}. \tag{3.36}$$

This extends to swaps in a similar way with

$$C_{pc}(S[u \leftrightarrow v]) = \max\left\{ \max_{\substack{j \in P \\ S_j \notin \{S(u), S(v)\}}} C^{pc}_{S_j}, \Delta^{pc}_{S(u)}{}', \Delta^{pc}_{S(v)}{}' \right\}, \tag{3.37}$$

where

$$\Delta^{pc}_{S(u)}{}' = \min\Big\{ \big[d_{wu} = \delta^{pc}_{wj}\big] \max\{d_{wv}, \gamma^{pc}_{wj}\} + \big[d_{wu} \neq \delta^{pc}_{wj}\big] \max\{d_{wv}, \delta^{pc}_{wj}\} \tag{3.38}$$

$$\mid j : S_j = S(u),\ w \in \big(S(u) \cup \{v\}\big) \setminus \{u\} \Big\} \tag{3.39}$$

and vice-versa for $\Delta^{pc}_{S(v)}{}'$.

Given a neighborhood of size k and assuming an average of n/p units per district, updates to C^{pc}, δ^{pc} and γ^{pc} take $O(n^2/(pk))$ amortized time per candidate in the worst case. However, note that the second case of (3.31), which leads to the added $O(n/p)$ factor, need only be considered if $d_{uv} > \gamma^{pc}_{vi}$, i.e. only if d_{uv} is a critical distance in $S_i \setminus \{u\}$. In a scenario where the set of unique distances is roughly equal to the set of unit pairs, which is generally expected in Euclidean domains, a critical unit will be shifted on average once every n/p moves, leading to an average amortized update cost of $O(n)$ per candidate. When combined with the $O(n/p)$ costs to compute candidates' potential p-center values, this ultimately leads to an amortized cost of $O(n/p)$ per candidate in both N_{shift} and N_{swap}.

We note two ways by which this cost could be improved. First, similar to the strategy mentioned in Section 3.3.5.3 one could use pn max-heaps to maintain δ^{pc}_{vi} and γ^{pc}_{vi} in average time $O((n/p)\log(n/p))$, thereby mitigating the cost of the second case (3.31). This would have an added memory cost of $O(n^2)$, however. Second, like in Section 3.3.5.4, in (3.27), (3.28), (3.30) and (3.31) one could update δ^{pc}_{vi} and γ^{pc}_{vi} only for $v \in S_i \cup \partial(S_i)$, which would lead to a worst-case update cost of $O(\sqrt{n}(n/p))$.

3.3.5.6 Caching movements on TS_b

To avoid recomputing movements on the balancing tabu search TS_b, we cache the outcome of operators. Given a solution S and a maximum compactness value $\bar{c}$ for TS_b, for each district $i \in P$ we maintain a list Λ_i of potential shift and swap moves incident to district i, sorted increasing by BC of the resulting solution. Lists Λ_i are always updated with respect to shift moves, but include swap moves lazily as needed, since the swap neighborhood is larger. Therefore, we also maintain binary flags $\lambda_i \in \{0, 1\}$ that indicate whether list Λ_i contains swap moves.

Algorithm 5 outlines the strategy. Initially, for all $i \in P$ we set $\lambda_i = 0$ and Λ_i to be a list of feasible shifts incident to district S_i, sorted increasingly by expected BC (lines 1–4). When computing the best neighboring move, we then proceed as follows. For each $i \in P$ we iterate over Λ_i until we find a feasible move, i.e. a move

Algorithm 5 Caching strategy for TS_b.

```
 1: for i ∈ P do                                   ▷ initialization, run before the search starts
 2:     λ_i ← 0
 3:     Λ_i = {u→i ∈ N^b_shift(S, c̄) | u ∈ V, S[u→i] is feasible}
 4:     sort Λ_i increasingly w.r.t. BC.
 5:
 6: procedure CACHEITERATE(S)
 7:     m_best ← null
 8:     for i ∈ P do
 9:         for m ∈ Λ_i, in order of increasing BC(S[m]) do
10:             if S[m] is infeasible then
11:                 remove m from Λ_i
12:             else
13:                 m_best ← argmin{BC(S[m_best]), BC(S[m])}
14:     return m_best
15:
16: procedure CHOOSENEXTMOVEFROMCACHE(S)
17:     m_best ← cacheIterate(S)
18:     if m_best is null then
19:         for i ∈ P such that λ_i = 0 do
20:             Λ_i = Λ_i ∪ {u↔v ∈ N^b_swap(S, c̄) | u, v ∈ V, S(u) = i, S[u↔v] is feasible}
21:             sort Λ_i increasingly w.r.t. BC.
22:         m_best ← cacheIterate(S)
23:     if m_best is not null then
24:         for each district i incident to m_best, as well as i's neighbors do
25:             λ_i ← 0
26:             Λ_i = {u→i ∈ N^b_shift(S, c̄) | u ∈ V, S[u→i] is feasible}
27:             sort Λ_i increasingly w.r.t. BC.
28:     return m_best
```

m where $S[m]$ is connected and $C(S[m]) \leqslant \bar{c}$ (line 17, procedure cacheIterate). In the process we discard all moves from Λ_i that come before m, since they are infeasible. Note that feasibility must be tested here and cannot be cached, since cached moves in Λ_i may become infeasible as the search progresses. (However, we can still do this test in $O(1)$ by keeping, for each candidate move cached, the change in compactness caused by it.) If any feasible move is encountered, we select the best such move among all $i \in P$. Otherwise, we iterate over the districts once again to add swap moves to caches that do not include swaps (lines 18–22). Finally, we iterate once again over each λ_i until we find a feasible move, discard infeasible moves seen in the process, and select the best move encountered, if any (line 22).

After applying move m_i, we reset λ_i and Λ_i for each district i incident to the selected move, as well as each of i's neighbors, since they could include moves that af-

fect i (lines 23–27). Because neighboring relations between districts induce a planar graph (since G is planar) the average number of districts reset is at most $12 = O(1)$, in contrast to $O(p)$ if we did no caching. This saves us a factor p in time complexity compared to computing the entire N_{shift} and N_{swap} neighborhoods, but could require us to store all neighbors in memory, in the worst case. Note, however, that for the p-center and diameter objectives we only need to store the first element of each Λ_k, thus using constant space, since due to the maximum-based objective a feasible move with respect to $C(S) < \bar{c}$ will continue to be feasible regardless of later moves.

3.4 Problem instances

For our experiments we consider a problem in the sales districting domain. The domain was originally introduced by Segura-Ramiro et al. [197], and later studied by several authors [181, 187, 183, 189, 191, 182, 54, 177, 180, 184]. In it, basic units consist of city blocks and districts of regions which will be serviced independently by salespersons of a company. As is common in sales districting, these instances include multiple balancing attributes, namely three: the workload, the number of customers and the product demand of each city block. These attributes must be evenly distributed among the districts. We used all three attributes in our implementation.

We chose this domain because it is likely the most widely studied in the districting literature, and open source implementations as well as instance sets derived from real-world data are available. Moreover, this domain has been studied in the light of several classical optimization models which include Model (1.1) as a subproblem, such as the PMDP, PCDP, DDP and variants thereof, which facilitates the assessment of our heuristic.

We use three instance sets from the literature:

- RF, originally introduced by Ríos-Mercado and Fernández [181] for the p-Center Districting Problem. It has two classes of instances, "DS" and "DT", that differ in the distribution of attribute values, each having 20 instances of size $n = 500$ with $p = 10$ districts each, for a total of 40 instances. Class DS selects attribute values uniformly from fixed intervals, while class DT selects them from a non-uniform symmetric distribution, which according to the authors models real-world applications more closely.
- SRC, originally proposed by Salazar-Aguilar et al. [187] to test exact formulations for the p-Center and p-Median Districting Problems. It contains 20 instances of each size $(n,p) \in \{(60,4),(80,5),(100,6),(120,7)\}$ and 10 of $(n,p) \in$

$\{(150,8),(200,11)\}$, for a total of 80 instances. These comparatively smaller instances reflect the instance sizes treatable by the exact method of Salazar-Aguilar et al. [187] at the time. As in instance class RF/DS, all instances use uniform attribute generation.

- RS, proposed by Ríos-Mercado and Salazar-Acosta [183] for a variant of the Diameter Districting Problem with routing budget constraints. It has two subclasses: "DU", with 30 instances of $n = 1000$ and $p = 10$, and "DT", with 15 instances of $n = 1000$ and $p = 10$. Ríos-Mercado and Salazar-Acosta [183] state that these instances were obtained using the generator from Ríos-Mercado and Fernández [181], but it is not clear which attribute generation scheme was used.

All three sets RF, SRC and RS define planar coordinates to each unit drawn randomly from a uniform distribution. Using these coordinates, distances d_{uv} are defined as Euclidean distances. The original papers do not specify how planar graph topologies for these instances were generated.

Because our methods are also able to handle larger instances, for a broader evaluation we have generated an additional instance set, which we named GRM after the authors' initials from Gliesch et al. [81]. It has 4 instances for each combination of $n \in \{1000, 2500, 5000, 10000\}$ and $p \in \{n/200, n/100, n/62.5\}$, for a total of 48 instances. The three levels of p roughly represent difficulties "easy", "medium" and "hard" with respect to achieving feasibility, as per our preliminary experiments. We chose these size ranges since our method did not consistently find feasible solutions for instances of $n > 10000$, whereas smaller instances than $n < 1000$ were well-covered by other instance sets. We use the same uniform attribute generation as in instance class DS from set RF, since we have found it to be more challenging than the one used for class DT. As in sets RF, SRC and RS we define planar coordinates to each unit, and use Euclidean distances. These coordinates were drawn uniformly at random from $[0, 1000]^2$. The graph topology was obtained by computing a Delaunay triangulation on the unit coordinates, which ensures connectivity and planarity.

Table 3.1 summarizes information about the instance sets used. To be consistent with other works in this application domain, in all experiments we use balancing tolerances $\tau_1 = \tau_2 = \tau_3 = 0.05$.

3.4.1 A note on the experimental configurations of the following experiments

Because this chapter combines computational analyses from four different publications, which in the interest of time we did not replicate, the experiments we report

Table 3.1: Test instance data.

Inst.	#	n	p	Attr.	Dist.	Source
SRC	100	60–200	4–11	Uniform	Euclidean	[187]
RF/DS	40	500	10	Uniform	Euclidean	[181]
RF/DT	40	500	10	Symmetric	Euclidean	[181]
RS/DU	30	1,000	10	—	Euclidean	[183]
RS/DT	15	1,000	40	—	Euclidean	[183]
GRM	48	1,000–10,000	5–160	Uniform	Euclidean	This work

in the following sections for the main heuristic (Section 3.5), for our extension towards routing criteria (Section 3.6.3), and for our extension towards similarity criteria (Section 3.7.3) used different experimental configurations and choices of instance set. Namely, Section 3.5 uses instance sets SRC, RF and GRM, 10 replications, and a time limit of 60 minutes; Section 3.6.3 uses sets SRC, RF, RS and only a subset of GRM; and Section 3.7.3 uses sets SRC, RF and RS. Both Sections 3.6.3 and 3.7.3 run a single replicate with a fixed seed, limited to 10 minutes and 1000 multistart iterations. We are confident, however, that the different experimental configurations do not invalidate our analyses, since we only compare variants run under the same configuration.

3.5 Computational experiments

In this section we report on experiments that assess the different components of our heuristic, and compare it to other approaches in the literature for the PMDP, PCDP and DDP. First, in Section 3.5.1 we calibrate the parameters of our heuristic using an automatic calibration method. Then, in Sections 3.5.2 to 3.5.4 we compare our method to existing approaches for the p-median, p-center and diameter objectives, respectively. For both the p-median and p-center objectives we consider the exact algorithm of Salazar-Aguilar et al. [187], which solves the Hess model with lazy connectivity constraints of type (2.24), as we saw in Section 2.4.1.4. Further, for the p-center objective we also consider the GRASP heuristic of Ríos-Mercado and Fernández [181]. Finally, to assess the diameter objective we consider the GRASP heuristic of Ríos-Mercado and Escalante [180], as well as an earlier method developed in our research group [81].

We have executed all experiments on a PC with an 8-core AMD FX-8150 processor and 32 GB of main memory, running Ubuntu Linux 18.04. For each experiment, only one core was used. Our algorithms were implemented in C++ and compiled with

Table 3.2: Parameter calibration: optimization ranges and best setting found by irace.

Parameter	Optimization range	p-med.	p-cen.	diam.
t	$\{0.1, 0.25, 0.5, 1, 1.5, 2, 2.5, 5\}p$	1.5p	0.25p	5p
I_{max}	$\{50, 100, 250, 500, 1000, 2500, 5000\}$	100	1000	5000

GCC 7.2 with maximum optimization. The source code, instance sets and detailed results are available upon request.

Each test was run with a time limit of 60 minutes. Since the method is stochastic, we report averages over 10 replications. In these experiments we used instance sets SRC, RF and GRM, and report results by instance set and number of units n. For the heuristic methods we report the average deviation of the compactness value relative to the best known value, in percent (C (r.d.)), the number of multistart iterations (Iter.), the percentage of multistart iterations which resulted in a feasible solution (Fea. (%)), and the time to find the best solution, in seconds (t.t.b.). For the exact methods, we report the percentage of instances for which an optimal solution was found (Opt. (%)), the running time in seconds (t), and the number of lazy connectivity cut calls (Iter.).

3.5.1 Calibrating the parameters of our heuristic

We used the irace package version 3.1 in GNU R [138] to calibrate parameters t (the tabu tenure) and I_{max} (the maximum number of tabu search iterations without improvement) of our heuristic. We calibrated these parameters independently for the p-median, p-center and diameter objectives. For each objective we ran irace with a budget of 1000 executions and a time limit of 10 minutes per run. To avoid overfitting, for the calibration we used a separate instance set consisting of 48 random instances, which we have generated with the same parameters and algorithm as instance set GRM. As recommended by Lei et al. [128], we use tabu tenure values t relative to the number of districts p.

Table 3.2 shows the parameter ranges and best values found by irace, for each of the three objective functions.

3.5.2 Experiment 1: p-median objective

In this experiment we compare our approach to the exact method of Salazar-Aguilar et al. [187] for the p-median objective, in the implementation provided by the au-

Table 3.3: Comparison of our proposed method to the exact approach of Salazar-Aguilar et al. [187] for the p-median objective.

Inst.	n	Our algorithm				Salazar-Aguilar et al. [187]		
		C (r.d.)	t.t.b.	Iter.	Fea. (%)	Opt. (%)	t	Iter.
SRC	60	**0.00**	0	193,733	100	100	25	1.2
SRC	80	1.34	130	74,056	100	100	35	1.4
SRC	100	0.02	52	54,927	100	100	40	1.3
SRC	150	0.02	147	20,195	100	100	147	2.0
SRC	200	0.02	334	12,453	99	100	765	1.3
RF	500	0.04	701	1,999	89	—	—	—
GRM	1,000	0.03	661	255	100	—	—	—
GRM	2,500	0.13	853	101	94	—	—	—
GRM	5,000	0.28	882	50	98	—	—	—
GRM	10,000	0.35	955	21	91	—	—	—
Avg.		0.22	471	35,779	97	100	202	1.4

thors. It was written in C++, and we have compiled it in the same environment as our own, using CPLEX 12.7.1. Their method iteratively solves Model (2.10) to completion without connectivity constraints, then identifies violated constraints of type (2.24) using graph search, and adds them to the model. This is repeated until the exact solution provided by the solver is connected. Table 3.3 shows the comparison results.

We see that our heuristic found feasible solutions in 97% of multistart iterations, on average, considering all instance sizes. Since the method is repeated over several iterations (see column Iter.), this means it found at least one feasible solution in each instance. For instances of size $n \in \{60, 80, 150, 1000\}$ all multistart iterations were feasible. The heuristic reached optimality in all instances of size $n = 60$, and overall found optimal solutions in 84.6% of cases with $n \leqslant 200$. For the remaining cases it found solutions very close to optimality, on average within a factor of 0.02% of it. For $n = 80$ it obtained poor results in two particularly difficult instances, which resulted in a higher deviation. We have not identified what makes these instances so difficult, but we suspect it is due to their particular graph topologies. As expected, the number of multistart iterations decreases as n increases, since operator costs and neighborhood sizes increase, which slows down the method. This same trend is not observed however in feasibility rates, as our method consistently finds feasible solutions even for $n = 10000$.

The exact method of Salazar-Aguilar et al. [187] found optimal solutions in all instances of size $n \leqslant 200$, but timed out in all other cases for $n \geqslant 500$ without finding a single feasible upper bound. A future investigation might shed some light on what the tractable limits of this method are for $200 < n < 500$. In all instances it solved, it con-

verged to a connected solution in less than 2 iterations, in average. This corroborates our observation from Section 2.3.3 that optimal p-median solutions are correlated with connectivity when the planar topology is uniform.

Although our proposed heuristic performed slightly worse than the method of Salazar-Aguilar et al. [187] in smaller instances, on average, it still found the optimal solution in the vast majority of cases, or solutions very close to it in objective value. Being a heuristic, it also finds feasible solutions in a fraction of the time of the exact method, and can handle much larger instances.

3.5.3 Experiment 2: p-center objective

In this experiment we compare our heuristic to two approaches for the p-center objective. First, we consider the exact method of Salazar-Aguilar et al. [187], which is near identical to the exact method we saw in the last section, only it uses a p-center version of the Hess model. Second, we consider the GRASP heuristic of Ríos-Mercado and Fernández [181]. It uses a greedy constructive heuristic followed by a local search, with an additional filtering technique to discard unpromising multistart solutions and a dynamic calibration of GRASPS's RCL parameter α. Because no source code for this algorithm was available, we reimplemented it in C++ and compiled it under the same environment as our own. For a fair comparison, in the reimplementation we used the dynamic update techniques discussed in Section 3.3.5. We have fixed the parameters ρ, β and A of the method to the values recommended by Ríos-Mercado and Fernández [181]: $\rho = 1.0, \beta = 0.5$ and $A = \{0.1, 0.2, 0.3, 0.4, 0.5\}$, and parameter λ to 0.95 as given by Ríos-Mercado [177].

Table 3.4 shows the results. We see that the exact method of Salazar-Aguilar et al. [187] performs much worse for the p-center than for the p-median. For $n = 150$ the exact method solved optimally only half the instances, while no instances of $n \geqslant 200$ were solved nor any upper bound was reached within the time limit. This is most likely because the p-center objective is less correlated with connectivity, as we saw in Section 2.3.3, and so the method requires more connectivity cuts and consequently more solves, as shown by column Iter.

Considering instances solved optimally by the method of Salazar-Aguilar et al. [187], our heuristic found optimality in 95.4% of cases, and obtained an average relative deviation of 0.27% from the optimal solutions otherwise. For the remaining instances, our heuristic found the best average p-center values in all cases, when compared to the heuristic of Ríos-Mercado and Fernández [181]. On data set SRC

Table 3.4: Comparison of our proposed method to the heuristic of Ríos-Mercado and Fernández [181] and the exact approach of Salazar-Aguilar et al. [187], for the p-center objective.

Inst.	n	Our algorithm				Ríos-Mercado and Fernández [181]				Salazar-Aguilar et al. [187]		
		C (r.d.)	t.t.b.	Iter.	Fea. (%)	C (r.d.)	t.t.b.	Iter.	Fea. (%)	Opt. (%)	t	Iter.
SRC	60	0.13	50	16,518	96	0.67	36	2,627,936	45	100	52	4.3
SRC	80	0.72	19	11,305	97	1.63	153	1,707,644	30	100	255	8.1
SRC	100	**0.00**	4	13,126	100	1.16	230	1,277,550	42	90	1,018	11.6
SRC	150	**0.06**	107	5,909	100	2.72	515	767,707	30	50	3,436	10.4
SRC	200	**0.00**	44	3,660	100	3.97	812	493,211	15	—	—	—
RF	500	**0.31**	499	1,086	97	13.11	870	111,890	15	—	—	—
GRM	1,000	**0.79**	688	254	100	27.29	934	25,531	7	—	—	—
GRM	2,500	**1.68**	949	57	99	65.82	1,149	6,468	6	—	—	—
GRM	5,000	**1.62**	946	23	99	101.89	1,827	2,243	5	—	—	—
GRM	10,000	**1.76**	980	9	94	161.57	2,347	725	2	—	—	—
Avg.		**0.71**	429	5,195	98	37.98	887	702,091	20	85	1,190	8.6

the heuristic of Ríos-Mercado and Fernández [181] performs a very large number of iterations and thus obtains relatively good solutions, but as n grows it has difficulty finding feasible solutions. This likely explains its poor solution quality for instance sets RF and GRM, since the effectiveness of GRASP relies on having a large sample of feasible solutions.

Compared to the p-median objective, the p-center version of our algorithm performs 22.16 times fewer multistart iterations in the same amount of time. This is due to the differences in the calibrated parameter I_{max} (see Table 3.2), which determines the amount of time spent on each tabu search. Therefore, although the p-center version performs fewer multistart iterations, it spends more time in each one. In practice, however, given equal values for parameters t and I_{max} we found that the p-center is usually faster, since balancing requires fewer binary search steps, on average. This happens because the number of possible p-center objective values is limited to $\binom{n}{2}$.

3.5.4 Experiment 3: diameter objective

In this experiment we compare our heuristic to two approaches for the diameter objective. First, we consider an earlier version of our method targeted only at the diameter, which was published in Gliesch et al. [81]. It differs from the current method in that it does not use a filtering strategy on multistart solutions, and uses a simple tabu search over N_{shift} to optimize balance, instead of the algorithm described in Section 3.3.3. This method was implemented in C++ and compiled under the

Table 3.5: Comparison of our proposed method to the heuristic approaches of Gliesch et al. [81] and Ríos-Mercado and Escalante [180] for the diameter objective.

Inst.	n	Our algorithm				Gliesch et al. [81]				Ríos-Mercado and Escalante [180]		
		C (r.d.)	t.t.b.	Iter.	Fea. (%)	C (r.d.)	t.t.b.	Iter.	Fea. (%)	C (r.d.)	Iter.	Fea. (%)
SRC	60	**0.00**	0	888,007	100	0.61	52	285,632	100	—	—	—
SRC	80	**0.00**	0	461,409	100	0.14	28	108,174	96	—	—	—
SRC	100	**0.00**	0	280,698	100	0.24	62	114,601	97	—	—	—
SRC	150	**0.00**	18	130,446	100	0.21	78	144,994	100	—	—	—
SRC	200	**0.03**	64	24,169	100	0.04	68	69,743	98	—	—	—
RF	500	**0.07**	329	3,634	99	1.05	378	61,316	87	5.94	404	99
GRM	1,000	**0.33**	532	1,070	100	1.06	564	18,664	84	11.52	94	100
GRM	2,500	**1.82**	883	264	99	5.01	803	9,843	75	30.47	23	95
GRM	5,000	**2.91**	886	147	99	7.32	915	5,555	64	65.54	4	71
GRM	10,000	**5.31**	958	65	92	30.82	874	2,477	53	64.52	1	75
Avg.		**1.05**	367	178,991	99	4.65	382	82,100	85	35.66	84	82

same environment as our own.

Second, we consider the heuristic of Ríos-Mercado and Escalante [180], whose original source code the authors have made available. The heuristic uses GRASP to generate a set of solutions, then performs path relinking between all solution pairs to potentially obtain a better intermediate solution. Because it was originally implemented in MATLAB, for a fair comparison we have translated it to C++ and compiled it under the same platform as our own. As we have done for the method of Ríos-Mercado and Fernández [181], we included the optimization techniques of Section 3.3.5 in the reimplementation. During this process, we discovered that the original implementation did not always maintain connectivity during the path relinking process. To address this, we have adapted their method to simply stop path relinking once no more moves which maintain connectivity are available. Finally, since our multistart approach is based on a time limit, in our experiments we repeated their full method with different random seeds until the time limit is reached, and report the best solution found.

Table 3.5 shows the results. For the method of Ríos-Mercado and Escalante [180] column "Iter." reports the number of repetitions of their algorithm, and column "Fea. (%)" reports the percentage of these repetitions which produced a feasible solution. Because in the original experiment, published in Gliesch et al. [81], we only used instance sets RF and GRM, here we report results of Ríos-Mercado and Escalante [180]'s method for these two sets only.

When compared to the version published in Gliesch et al. [81] we see a large im-

provement in our method with respect to average solution values, finding solutions with a relative deviation of 1.05% to the best known solution, compared to 4.65% of the previous variant. For instances of size $n \leqslant 500$ the current method produced the best known solution in nearly all executions, and for $n \leqslant 100$ it generates significantly more feasible solutions in the same amount of time, and finds the best known solutions almost immediately. As n grows, however, the total number of multistart iterations decreases rapidly, leading to comparatively higher diameters in some cases. This decrease in performance at higher values of n is likely due to the increased difficulty of finding balanced solutions.

In general, we see that the method of Gliesch et al. [81] has more difficulty in balancing instances with high p/n ratios, especially as the number of units n grows. On the other hand, the performance of our current method is less dependent on the number of districts p. One reason for this may be the caching strategy for TS_b (see Section 3.3.5.6), which reduces the cost of visiting the shift and swap neighborhoods by a factor of p, and which the original method did not use. Moreover, the current method found feasible solutions in 99% of multistart iterations, compared to a rate of 85% of Gliesch et al. [81]'s method, which also decreases rapidly for $n \geqslant 2500$. The cause of this difference is that the original method did not use a binary search strategy for optimizing balance, but a single tabu search instead.

Looking at the method of Ríos-Mercado and Escalante [180] we see that it has a high probability of finding a feasible solution in a single execution, 84% on average. Because path relinking is time-consuming, however, the number of re-executions was comparatively low, and so was the likelihood of finding feasible solutions. The method also found solutions of worse diameters, compared to the two other approaches. We believe the reason is its difficulty in balancing solutions, which reduces the pool of solutions path relinking can choose from. Yet, we note that in instance class DS from set RF the method of Ríos-Mercado and Escalante [180] was consistently better than that of Gliesch et al. [81]. We are not sure why this is the case.

Our method was significantly faster per iteration when optimizing for the diameter, when compared to p-center and p-median. This is despite the high value of parameter I_{max}, which sets the maximum number of non-improving tabu search iterations. This improved performance stems from the lower asymptotic costs of computing expected diameter values of neighborhood search candidates, as we have described in Section 3.3.5: for both p-center and p-median this takes $O(n/p)$ time, on average, while by using a specific algorithm that exploits the fact that instances are Euclidean, we do the same for the diameter in $O(p + \log n/p + n/p^2)$ time.

3.6 Extension to routing criteria

In this section we show how we have extended our heuristic to include routing costs, as either a constraint or as an objective. We have reviewed routing criteria in Section 2.3.5. Our goals are both to show how our heuristic can be extended to new optimization criteria, and to establish a baseline algorithm and best known values for the resulting problem. In Section 3.6.1 we present how we model routing costs. Then, in Section 3.6.2 we explain the changes we have made to the heuristic. These include a modified objective function, an updated constructive algorithm, and a heuristic approach to compute and dynamically update district routes, which is, in most cases, near optimal. Finally, in Section 3.6.3 we describe how we update existing instance sets to include routing criteria, and report on experiments for the different versions of our method.

The work presented in this section has been published in the following conference paper:

- A. Gliesch, M. Ritt, A. H. S. Cruz, and M. C. O. Moreira. A hybrid heuristic for districting problems with routing criteria. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–9, Glasgow, United Kingdom, July 2020. IEEE.

3.6.1 Modeling routing costs

We model routes as shortest Traveling Salesman Problem (TSP) tours that start and end at predefined depot units. Namely, given district S_i the route which attends it is modeled by a tour which starts and ends at a depot unit $h_i \in V$ associated with S_i, and visits all units of S_i in an order such that the total tour cost is minimal. We denote this optimal cost by $R(S_i)$. We assume input graph G is connected, and define the cost of each tour edge u–v as d_{uv} if u and v are neighbors, or as the cost of a shortest path between u and v in G using distances d as edge weights, otherwise. These shortest path distances are precomputed, meaning we allow routes to visit units outside their respective districts. An alternative would be to require routes to stay within their districts. We do not consider this variant, however, since as mentioned in Section 2.3.5 optimal distances must be recomputed after every neighboring move, making it too computationally costly.

Depot units $h_1, \dots, h_p$ are fixed to each district. For simplicity we consider depots to be locally unique, i.e., $h_i \neq h_j \mid \forall i, j \in P$, although some variants may also consider a single global depot for all districts, or ignore depots altogether. The algorithms

proposed here can also handle these variants, however, and in Section 3.6.3.6 we briefly examine them in an experiment. Note that not necessarily $h_i \in S_i$, since we allow cross-district routes. In Figure 2.5 we have presented an example of a districting and routing problem with a global depot.

The following formulation summarizes the resulting optimization problem, which follows the structure of Model (2.1):

$$\begin{aligned}
\underset{S \in \mathcal{S}}{\textbf{minimize}} \quad & \lambda R(S) + (1-\lambda) C(S) && && \text{(3.40a)}\\
\textbf{subject to} \quad & R(S_i) \leqslant \overline{R}, && \forall i \in P, && \text{(3.40b)}\\
& L_a \leqslant w_a(S_i) \leqslant U_a, && \forall i \in P, a \in A, && \text{(3.40c)}\\
& \text{connected}(S_i), && \forall i \in P. && \text{(3.40d)}
\end{aligned}$$

As objective, given solution S we optimize a convex combination of the compactness $C(S)$ and the total routing cost of all districts $R(S) = \sum_{i \in P} R(S_i)$, given weight $\lambda \in [0,1]$. Further, we impose a budget constraint $R(S_i) \leqslant \overline{R}$ on the cost of each tour, given $\overline{R} \in \mathbb{R}^+$. Both λ and $\overline{R}$ are part of the problem input. To reflect real-world scenarios, we assume routing is always either handled as a constraint by setting $\lambda = 0$, or as an objective, by setting $\overline{R} = \infty$. We have chosen this type of formulation rather than a multi-objective one to avoid modifications to our main heuristic. We note however that a Pareto front for the multiobjective problem optimizing both C and R could be obtained from systematically varying either (or both) λ and $\overline{R}$.

To maintain our focus on routing criteria, in this section we fix $C = C_{pm}$. We have chosen the p-median here because, as we saw in Section 2.3.2 it is correlated with connectivity, and accounts for the compactness of all districts locally, as a contrast to the p-median and diameter which only consider the least-compact district.

3.6.2 Extending the heuristic to include routing costs

In the following subsections we describe how we have extended our method of Section 3.3 to include routing costs. The core algorithm remains unchanged, and thus we only discuss our modifications to it. Two changes however apply to the algorithm as a whole.

First, we extend the definition of imbalance B to include violations to routing budget constraints, and thus define

$$E(S) = \sum_{a \in A, i \in P} b_a(S_i)/\mu_a + \sum_{i \in P} \max\{0, R(S_i) - \overline{R}\}/R(S_i) \qquad \text{(3.41)}$$

as the sum of violations to both constraints (3.40c) and constraints (3.40b). Constraint violations are normalized by $\mu_a = \mu_a = \sum_{v \in V} w_v^a / p$ and $R(S_i)$ respectively, to keep both in the same order of magnitude. We replace every call to $B(S)$ in the main algorithm with a call to $E(S)$.

Second, we replace every call to $C(S)$ with a call to the objective

$$Z(S) = \lambda R(S)/U_r + (1-\lambda)C(S)/U_c \tag{3.42}$$

of Model (1.1), with both terms normalized by constants U_r and U_c. This allows a more intuitive choice of λ. We explain how we define U_r and U_c in Section 3.6.2.6.

3.6.2.1 Computing routing costs

For each district $i \in P$, we keep a permutation T_i of $S_i \cup \{h_i\}$ representing district i's current TSP tour, and let $R(T_i)$ be its cost. Because the TSP is NP-hard [69], maintaining optimal tours during optimization is a computational bottleneck, particularly when compared to other aspects of the algorithm such as modifying solutions or recalculating p-median values. This remains true even when considering efficient exact solvers like Concorde [5], which was used in the local search of Ríos-Mercado and Salazar-Acosta [183], but dominated the running time of our algorithm by a large margin in early tests. Therefore, during optimization we compute tours heuristically, allowing us to complete GRASP iterations faster and thus execute more iterations in total. This, however, means we allow suboptimal tours during optimization, and so not necessarily $R(T_i) = R(S_i)$.

Routing costs are evaluated at several different stages of the algorithm, each with distinct requirements regarding performance and precision (i.e., gap from the optimal cost). For example, the routing costs of neighbors during the tabu search must be computed quickly lest they become a bottleneck. On the other hand, at the end of each multistart iteration we update the globally best solution, and so a high-precision heuristic is desired, even if it is slower. With this in mind, we have identified five different levels of increasing precision and decreasing performance requirements for the evaluation of routing costs:

1. When evaluating a candidate shift or swap move during tabu search. This action is performance-critical, since it happens at the inner-most loop of the algorithm. All other attributes of solutions, such as balancing constraints, p-median values, or connectivity can be kept dynamically in time $O(n/p)$, and so a TSP heuristic here must ideally also be computed in similar time.

2. When executing a tabu search move, i.e., applying a shift or swap to a solution, or when assigning fringe units during greedy construction. This is executed once every $O(n)$ candidate move evaluations, on average, and so the cost of a more expensive algorithm here is amortized over the n evaluations.
3. At the end of each tabu search. This happens after at least I_{max} moves have been made. A better TSP heuristic here helps mitigate possible deviations from optimality that may have been propagated by successively using a simpler heuristic earlier.
4. At the end of each multistart iteration. In practice this happens after around 25 tabu searches, on average. Besides reducing imprecisions from lower levels, a more precise algorithm here helps to select the best solution among all multistart iterations.
5. When the algorithm halts. Here, we always recompute tours optimally.

3.6.2.2 Solving the TSP

In the literature there are several TSP heuristics with varying trade-offs between effectiveness and running times [93, 6]. For our purposes we consider three: a constant-time greedy insertion heuristic GU, which we propose in Section 3.6.2.3; 2-opt local searches (2OPT), which are easy to implement and generally fast if the initial solution is close to a local minimum; and the Lin-Kernighan [137] heuristic (LKH), which is among the most effective and widely used heuristics for the TSP today. For the LKH we use the implementation in the Concorde solver [5] and consider two variants, LK_1 and LK_2, with different parameter configurations: LK_1 uses stallCount $= 1000$ and maxKicks $= n/5$, where stallCount is a stagnation parameter and maxKicks defines the number of perturbation steps, while LK_2 uses stallCount $= 5000$ and maxKicks $= n$. Variant LK_1 is aimed to be faster at the cost of slightly worse tours, while LK_2 invests more time in finding better solutions.

Let $A_1, \ldots, A_5$ be parameters that define the algorithms used in levels 1-5 described in Section 3.6.2.1. We have fixed A_1 to be GU, as it is imperative that evaluations at the lowest level be efficient. Because the best choice of A_2 and A_3 was not obvious, we consider $A_2 \in \{\text{GU}, \text{2OPT}\}$ and $A_3 \in \{\text{2OPT}, \text{LK}_1\}$, and calibrate this choice experimentally in Section 3.6.3.2. For A_4 we use LK_2, since an effective heuristic is needed to select the best multistart solution. For the exact algorithm A_5 we use Concorde's exact solver. Algorithms 2OPT, LK_1, LK_2 and the exact solver are warm-started by the existing, already optimized tours, which improves their performance considerably.

Note that, since heuristic solutions for the TSP are upper bounds on the optimal tours, if any heuristic tour satisfies budget constraints then the corresponding optimal tour also does. Similarly, any value of Z computed with heuristic tours is an upper bound on the optimal value of Z.

3.6.2.3 The greedy update heuristic

In order to update TSP tours efficiently upon unit insertions and removals, we propose the following fast greedy heuristic, which we call GU after "greedy update". It assumes the given tour has been previously optimized by a more precise TSP heuristic, and therefore tries to maintain the original permutation as much as possible.

When removing unit u from tour T_i, GU simply removes u and maintain the rest of the permutation intact, i.e., we link the units that come before and after u in T_i. Because T_i must always include depot h_i, however, if $u = h_i$ we simply do not remove it.

When adding unit u to tour T_i, we choose some position $j \in \{0, \ldots, |T_i|\}$ and insert u after the j-th unit in T_i, maintaining the rest of the tour intact. Of course, the quality of the resulting tour depends on the choice of j. One way to select it would be to test all possible locations and take the one leading to the shortest tour. Since there are $O(n/p)$ possible candidates on average, however, this is too slow for our purposes, as GU should ideally take constant time. We therefore only consider positions j directly before or after neighbor units of u that are also in S_i. Our rationale here is that, because instances generally have Euclidean topologies, in practice optimal tours tend to visit neighboring units in G in sequence. Both the constructive algorithm and the tabu searches only allow moves that keep connectivity, thus u is guaranteed to have at least one such neighbor. Because G is planar its average unit degree is bounded by 6, and so examining u's neighbors takes amortized constant time. Again, if $u = h_i$ we do nothing, since in this case u is already present in T_i.

3.6.2.4 Changes to solution construction

Procedure greedyConstructive remains the same as we have described in Section 3.3.1, but during it we compute objective Z differently from the rest of the algorithm. If routing budget constraints are enabled, i.e. $\overline{R} \neq \infty$ and $\lambda = 0$, we exceptionally use a value $\lambda = 0.5$ when computing Z here. This is because the original heuristic does not use B to select each next unit to be included, but rather chooses the unit with smallest total attribute value. In this way we still give a preference to

initial solutions with shorter TSP tours when budget constraints are imposed, and maintain the original greedy algorithm unchanged. Moreover, do not consider depots when computing R, for reasons explained in the next section. Therefore, tours are computed simply with the units of each district.

We note that the caching mechanism described in Section 3.3.1 which maintains the best possible assignment in each district's fringe is still valid under objective Z.

3.6.2.5 Matching depots to districts

Our greedy constructive heuristic provides no guarantee that districts will be nearby their predefined depot locations $h_1, \ldots, h_p$, which are part of the problem input. This, of course, can have a significant impact on final routing costs, since a far away depot will induce additional travel times, and the alternating strategy is unlikely to restructure the solution enough to mitigate this. One way to address this issue could be to set the initial seed of each district in greedyConstructive as its associated depot. This would lead to very poor solutions, however, if depots are not evenly distributed over the instance topology. Therefore, we have opted for another approach.

Because districts are anonymous, i.e., they share the same balancing constraints, routing budgets and do not have location requirements, any assignment of depots to districts is valid, so we choose one minimizing total routing costs. We achieve this by solving a minimum cost bipartite matching subproblem on a graph of parts $v_1, \ldots, v_p$ and $u_1, \ldots, u_p$, where the cost of an edge (v_i, u_j) is the cost of a TSP tour over units $S_i \cup \{h_j\}$, where S_i is the i-th district of the greedily constructed solution and h_j the depot of index j given by the instance. This optimal matching can be computed in $O(p^3)$ time by the algorithm of Munkres [157], and each match (v_i^*, u_j^*) corresponds to depot h_j servicing district S_i. We therefore reorder the list h of depots accordingly.

In practice, computing p^2 optimal TSP tours to build the above graph is usually too expensive computationally, and so we compute these tours heuristically. We first optimize each $T_i, i \in P$ by 2OPT starting from a list of units in S_i in the order which they were added by greedyConstructive. Next, for each pair $i, j \in P$ we compute T_i^j as tour T_i with h_j inserted to it by GU, or simply $T_i^j = T_i$ if $h_j \in S_i$. We then optimize T_i^j once again with 2OPT, and set the cost of edge (v_i, u_j) to $R(T_i^j)$. Since T_i^j is only a slight modification over T_i to include h_j, in practice it is already close to a local minimum and 2OPT takes only a few iterations.

Note that the above is applied only once per multistart solution, after greedy construction. We have also considered running this procedure before each call to A_4 or

A_5 to reorganize depots and potentially obtain better solutions, but in practice this was not effective.

3.6.2.6 Normalizing objective terms

In order to keep compactness C and routing costs R in the same order of magnitude, thus allowing a more intuitive choice of λ, we divide them by upper bounds U_c and U_r. This places both terms within $[0,1]$. These bounds are obtained by generating a dummy solution S_d through a simple algorithm, and setting $U_c = C(S_d)$ and $U_r = R(S_d)$. Solution S_d is obtained as follows. First, p initial seed units, one per district, are selected uniformly at random. Then, while there are unassigned units, we iterate cyclically over the districts, each time assigning to the current district the unit of least index on its boundary, if it exists. Routes are updated by GU at every assignment, and we run 2OPT on each district at the end. Since upper bounds are only used for normalization in the interest of choosing the λ weight, solution S_d does not need to be feasible. In practice, final solution values deviate on average 4.4% from U_c, and 21.3% from U_r.

3.6.3 Computational experiments

We have implemented and run the modified method under the same environment described in Section 3.5, over all four instance sets SRC, RF, RS and GRM. Since computing routing costs slows down the heuristic considerably, in the following experiments we excluded instances of size $n \geqslant 5000$ from GRM. To avoid recalibrations, we fixed parameters $t = 1.5p$, $A_{max} = 100$ and $I_{max} = 100$ to the best values found in Section 3.5.1 for the p-median. It is likely that better parameter settings could be obtained from re-executing irace with the modified method. However, this would require a separate calibration for each value of λ and $\overline{R}$, which was outside our time budget. To save time, we limited each run in the experiments below to both 1000 multistart iterations and 10 minutes of runtime, and use a fixed seed. The time used by algorithm A_5 to compute the exact tours at the end is not counted towards the time limit, and so we do not include it in the tables below.

3.6.3.1 Setting instance depots

Because the instances described in Section 3.4 were originally proposed for models without routing criteria, no district depots are given. We assume that, in real-world

scenarios, depots tend to be more or less dispersed over the input graph. Therefore, we define p depots for each instance by executing the randomized p-dispersion algorithm of Section 3.3.1 with a fixed random seed, prior to the main algorithm. Since this defines the instances, the time used to do this is not counted towards the time limit.

We define routing budgets $\overline{R}$ for each instance in a separate experiment, reported in Section 3.6.3.4.

3.6.3.2 Experiment 1: influence of different TSP algorithms

In this experiment we calibrate algorithms A_2 and A_3 used to recompute district routes after neighboring moves and after tabu searches, respectively. As mentioned in Section 3.6.2.2, we make this choice experimentally since it was not clear what the best algorithm choice in these two situations is. We considered the choices $A_2 \in \{\text{GU}, \text{2OPT}\}$ and $A_3 \in \{\text{2OPT}, \text{LK}_1\}$. In the case $A_2 = \text{2OPT}$, when applying a neighboring move we first execute GU to update the given tour, and then optimize it by 2OPT. For this calibration we have fixed $\lambda = 0.5$ and no routing budgets. Table 3.6 shows the results. For each instance set and parameter choice we report averages of the final objective value (Z (r.d.)), shown as the relative deviation (in percent) to the best known values, the number of multistart iterations (Iter.), and the total running time in seconds (t).

We see that for all data sets the configuration $(A_2, A_3) = (\text{2OPT}, \text{LK}_1)$ achieved the best average Z values, despite being slightly slower when considering running times and iteration counts. This configuration includes the two heuristics of highest time-over-quality ratios, which suggests that investing more effort in finding better TSP routes during optimization pays off. The differences in performance between the four settings were not significant, since all allowed enough multistart iterations for feasible solutions to be found. Among the other three settings there are no clear losers or winners. In the experiments that follow, we have therefore fixed $(A_2, A_3) = (\text{2OPT}, \text{LK}_1)$.

Table 3.7 shows, for configuration $(\text{2OPT}, \text{LK}_1)$ and each instance class, the average tour size n/p and the ratios between tour costs obtained at the five stages considered. Here, each column A_i/A_j displays the average difference, in %, between tour costs found by algorithm A_i when it was executed on a tour previously computed by algorithm A_j.

We see that the proposed greedy update heuristic $A_1 = \text{GU}$ generally obtains results

Table 3.6: Calibrating TSP algorithms A_2 and A_3.

Inst.	A_2	A_3	Z (r.d.)	Iter.	t
SRC	2OPT	2OPT	0.01694	1,000.0	70.2
SRC	2OPT	LK_1	0.01231	1,000.0	76.3
SRC	GU	2OPT	0.01682	1,000.0	65.0
SRC	GU	LK_1	0.01382	1,000.0	71.1
RF	2OPT	2OPT	0.24025	373.2	600.0
RF	2OPT	LK_1	0.20466	390.5	600.0
RF	GU	2OPT	0.29308	403.3	600.0
RF	GU	LK_1	0.25944	416.5	600.0
RS	2OPT	2OPT	0.34796	164.8	600.0
RS	2OPT	LK_1	0.20894	162.5	600.0
RS	GU	2OPT	0.44179	176.4	600.0
RS	GU	LK_1	0.40115	171.6	600.0
GRM	2OPT	2OPT	0.34927	19.2	600.0
GRM	2OPT	LK_1	0.25759	24.3	600.0
GRM	GU	2OPT	0.30405	20.1	600.0
GRM	GU	LK_1	0.44132	25.8	600.0

that are very close (under 0.08% difference, on average) to the local minima found by $A_2 = 2OPT$. This shows that, despite making some simplifications to be computable in constant time, GU can be an effective way to maintain tours dynamically. Looking at $n \geqslant 500$, we also see GU scales well for larger tours. The largest deviations are found between $A_3 = LK_1$ and $A_2 = 2OPT$, and are likely due to the large disparity in effectiveness between these two heuristics. This same effect has been observed before by Johnson and McGeoch [110]. Column A_4/A_3 shows only a small difference between LK_2 and LK_1, specially for smaller instances, which could indicate that the LKH stagnates quickly and additional restarts do not help. Looking at the last column, we see that LK_2 is always optimal for $n \leqslant 200$, and very close to optimal (less than 0.005%) for $n \geqslant 500$, and thus is usually successful in selecting the best multistart iteration.

3.6.3.3 Experiment 2: effect of weight λ

In this experiment we analyze the effects of using different values of λ when considering routing as objective. Recall that λ is the weight given to routing costs R in Z, leaving a weight of $1-\lambda$ for compactness C. We consider values $\lambda \in \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$, and no routing budgets. At $\lambda = 0.0$ (i.e., with routing costs disabled) for performance we do not update routes during neighborhood search.

Table 3.7: Relative deviations of tour lengths between the different TSP algorithms used.

Inst.	n	n/p	A_2/A_1	A_3/A_2	A_4/A_3	A_5/A_4
SRC	60	15.0	0.1098	1.6274	0.0035	0.0000
SRC	80	16.0	0.0999	1.7490	0.0026	0.0000
SRC	100	16.7	0.1085	1.7289	0.0032	0.0000
SRC	120	17.1	0.0998	1.7041	0.0028	0.0000
SRC	150	18.8	0.0983	1.9437	0.0040	0.0000
SRC	200	18.2	0.1066	1.9533	0.0044	0.0000
RF	500	50.0	0.0417	2.0347	0.0223	0.0004
RS	1,000	75.0	0.0615	1.6006	0.0241	0.0013
GRM	1,000	120.8	0.0224	1.1444	0.0361	0.0044
GRM	2,500	123.6	0.0344	1.1705	0.0976	0.0047
Avg.		47.1	0.0783	1.6657	0.0201	0.0011

The same is done for C when compactness is disabled at $\lambda = 1.0$. Table 3.8 shows the results. For each instance set and value of λ we report the average deviation C (r.d.) of the compactness and the total routing costs R (r.d.) relative to the best known values, as well as the total number of multistart iterations (Iter.).

We observe a clear progression in the quality of p-median values with decreasing λ: all best known values for p-median are found at $\lambda = 0$. For routing costs, on the other hand, we see an interesting effect: the best R was consistently found not at $\lambda = 1.0$, but rather between $\lambda = 0.6$ and $\lambda = 0.8$. This can be better seen in Figure 3.2, where we plot the progression of route cost deviations as a function of λ, averaged over all instances. We believe this effect is due to p-median values being related to short TSP tours: geometrically compact districts are intuitively faster to traverse, and it is likely that C_{pm} has a better neighborhood landscape than R for shifts and swaps.

Looking at the number of iterations we see that variants $\lambda = 0$ and $\lambda = 1.0$ are the fastest, since they need not compute R and C, respectively. There does not seem to be a clear trend in performance for the other values of λ.

3.6.3.4 Experiment 3: determining routing budgets for instances

Since instances do not specify routing budgets $\overline{R}$, we define them experimentally. Ideally $\overline{R}$ should not be so low as to make all instances infeasible, nor so high as to make them trivial by allowing even unoptimized tours to satisfy them.

We use the Beardwood-Halton-Hammersley formula [13] as a base for our choice of $\overline{R}$. It states that the cost of an optimal TSP tour on n uniformly distributed units on

Table 3.8: Effect of different values of λ in the objective function.

Inst.	λ	C (r.d.)	R (r.d.)	Iter.
SRC	0.0	0.00	5.26	1,000.0
SRC	0.2	0.33	5.32	1,000.0
SRC	0.4	1.28	3.00	1,000.0
SRC	0.6	2.68	1.43	1,000.0
SRC	0.8	4.56	0.49	1,000.0
SRC	1.0	7.07	0.23	1,000.0
RF	0.0	0.00	5.38	619.7
RF	0.2	0.60	3.51	384.8
RF	0.4	1.46	1.51	391.8
RF	0.6	2.68	0.92	383.1
RF	0.8	5.94	0.28	346.4
RF	1.0	11.88	0.57	679.8
RS	0.0	0.00	3.61	231.9
RS	0.2	0.93	3.56	157.1
RS	0.4	2.07	1.55	159.6
RS	0.6	4.14	0.58	163.6
RS	0.8	7.13	0.36	159.4
RS	1.0	11.14	1.01	262.3
GRM	0.0	0.00	3.45	25.7
GRM	0.2	0.80	2.28	24.7
GRM	0.4	2.22	0.79	25.1
GRM	0.6	4.45	0.41	22.3
GRM	0.8	8.23	0.66	17.4
GRM	1.0	13.38	1.15	63.8

surface of area A, with Euclidean distances, converges to $H = b_d/\sqrt{An}$ for large n and some constant b_d. Here we extrapolate the formula to p districts by setting $\overline{R} = H/p$, and since units are associated with plane coordinates, use area A as the axis-aligned bounding box of all the units. Parameter b_d has been empirically determined to be about 0.714 [6]; however, since input graphs here are usually not complete and thus routing distances d not always Euclidean, we have found $b_d < 0.8$ to be too tight a constraint in almost all cases. Further, since our instance sets use different generators for their topologies, and because instance difficulties also vary according to n and p, it is unlikely that a single b_d works well for all instances. Thus, we choose b_d in a per-instance basis.

For each instance we considered $b_d \in \{0.8, 0.85, 0.9, \ldots, 1.25, 1.3\}$ and executed 10 multistart iterations of our algorithm with no time limit. In order to choose budgets that are challenging, we then selected the smallest $b_d{}^*$ for which at least 2 such iterations were feasible. Table 3.9 reports, for each instance set and n, the average $b_d{}^*$

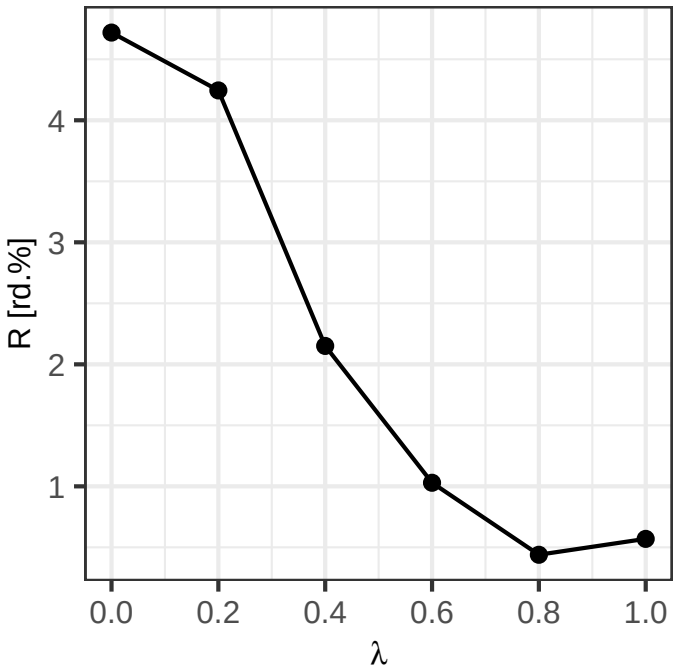


Figure 3.2: Effect of λ on routing costs, averaged over all instances.

found, as well as the actual number of feasible iterations (Fea.) obtained at $b_d = b_d{}^*$. We also report the average ratio between the mean tour cost R^*/p of the best known solutions and the calibrated budget $\overline{R} = b_d{}^*/\sqrt{An}$.

We see that $b_d{}^*$ has a mean of 1.02, suggesting that routing costs for the instances considered deviate about 30% from the empirical value 0.714 of the Beardwood-Halton-Hammersley formula. The selected values $b_d{}^*$ found feasible solutions in 3.65 out of 10 iterations, on average, and in at most 5.92 iterations for GRM instances of $n = 1000$, indicating that the generated budget constraints were neither trivial nor impossible to satisfy. The difference between the chosen budgets and the tour costs of the best known solutions was 20.8%, on average.

3.6.3.5 Experiment 4: routing budgets

In this experiment we solve our model with the routing budget constraints defined in Section 3.6.3.4. We use $\lambda = 0$ to disable routing costs in the objective function. Table 3.10 shows, for each instance set and n, the average deviations (in %) of the compactness (C (r.d.)) and routing costs (R (r.d.)) relative to the best known values, as well as the time (t), in seconds, the number of multistart iterations (Iter.), and the percentage of multistart iterations that were feasible (Fea. (%)).

Table 3.9: Calibrating b_d for determining routing budgets per instance.

Inst.	n	$b_d{}^*$	Fea.	$R^*/(p\overline{R})$
SRC	60	1.10	4.15	1.19
SRC	80	1.08	4.60	1.19
SRC	100	1.05	3.40	1.21
SRC	120	1.09	3.10	1.23
SRC	150	1.06	3.20	1.24
SRC	200	1.11	2.70	1.30
RF	500	1.01	2.68	1.26
RS	1,000	0.89	3.53	1.15
GRM	1,000	0.86	5.92	1.12
GRM	2,500	0.91	3.25	1.19
Avg.		1.02	3.65	1.21

Table 3.10: Results for the variant with routing budget constraints.

Inst.	n	C (r.d.)	R (r.d.)	t	Iter.	Fea. (%)
SRC	60	1.64	5.68	30.6	1,000.0	51.3
SRC	80	1.14	5.34	49.5	1,000.0	43.3
SRC	100	0.99	7.03	57.6	1,000.0	35.1
SRC	120	1.25	8.12	70.7	1,000.0	27.3
SRC	150	1.22	8.76	94.5	1,000.0	27.8
SRC	200	0.82	9.45	117.0	1,000.0	23.9
RF	500	0.14	6.87	593.2	730.4	25.0
RS	1,000	0.18	3.94	600.0	156.6	30.5
GRM	1,000	0.12	3.83	600.0	64.9	38.4
GRM	2,500	0.76	5.20	600.0	22.0	35.2
Avg.		0.83	6.42	281.3	697.4	33.8

We see that average compactness values C stay within 0.83% of the best known values, but nonetheless did not match them. This suggests budget constraints were binding, as procedure improveBalance (which, in this version, also minimizes routing budget constraint violations) likely was unable to make highly compact solutions feasible. Relative deviations of total routing costs R were high (6.42%, on average), since after satisfying budget constraints improveBalance stops considering routing costs at all. Feasibility rates per multistart iteration averaged only 33.8%; however, given the high number of total iterations, all runs found feasible solutions.

Table 3.11: Comparison of different variants regarding district depots.

Inst.	n	No depots		Global depot	
		C (r.d.)	R (r.d.)	C (r.d.)	R (r.d.)
SRC	60	-0.94	-1.72	-0.67	10.74
SRC	80	0.06	-3.65	0.01	13.49
SRC	100	-0.71	-1.30	-0.45	17.93
SRC	120	-0.37	-2.33	0.40	19.32
SRC	150	0.06	-2.17	-0.18	22.59
SRC	200	-0.62	-2.09	-0.53	29.64
RF	500	-0.83	-1.23	0.36	17.01
RS	1,000	-1.66	-1.70	0.12	24.46
GRM	1,000	-1.13	-0.07	0.43	12.00
GRM	2,500	-2.41	-0.86	1.39	22.90
Avg.		-0.85	-1.71	0.09	19.01

3.6.3.6 Experiment 5: different variants regarding routing depots

In this experiment we consider two additional variants regarding routing depots. The first uses a global depot h_G for all districts, which we define in the instances as the optimal 1-center unit $h_G = \text{argmin}_{i \in V} \max_{j \in V} d_{ij}$. The second does not use any depots. The adaptations to our algorithm required to treat these variants are minor, and omitted here. For each variant we ran our algorithm on all instances with $\lambda = 0.5$. Table 3.11 reports, instance set and n, the average compactness (C (r.d.)) and routing costs R (r.d.) of each variant as relative deviations (in percent) from the values of the default configuration, which uses a local depot to each district.

We observe that the variant with no depots finds 1.71% smaller routes, on average, compared to local depots. This was expected, as this variant incurs no additional costs when $h_i \notin D_i$. The single depot variant had significantly higher routing costs: 19% on average, which comes from all district tours having to visit h_G, regardless of the distance. Concerning C, we find that the variant without any depots was consistently better for most instance sizes. A possible explanation is that, because routes are shorter, R plays a lesser role in Z compared to C, and thus Z is more biased towards compactness.

3.7 Extension to similarity criteria

In this section we present how we have extended our heuristic to include similarity criteria, which are common in redistricting problems. We have reviewed this type of

criteria in detail in Section 2.3.4. As in Section 3.6, our goals are to show how our heuristic can be extended without changing its underlying algorithms, and to establish a baseline for future comparison for the resulting districting problem. This section is organized as follows. In Section 3.7.1 we present how we have modeled similarity criteria within Model (2.1). Next, in Section 3.7.2 we present the modifications made to our heuristic to handle these new criteria. They include a hard and a soft constraint strategy, changes to solution construction to ensure initial solutions are similarity-feasible, and efficient data structures to maintain similarity values. Last, in Section 3.7.3 we present how we modified existing instances to be redistricting problems using attribute diffusion models, and report on experiments for the different versions of our method.

The work presented in this section has been published in the following conference paper:

- A. Gliesch, M. Ritt, A. H. S. Cruz, and M. C. O. Moreira. A heuristic algorithm for districting problems with similarity constraints. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, Glasgow, United Kingdom, July 2020. IEEE.

3.7.1 Modeling redistricting problems with similarity criteria

The main reason for applying redistricting is when the existing attribute values have changed and thus lead to inefficient districts (e.g., when the total sum of the distances of voters to their voting center gets too large) or to infeasible solutions (e.g., when the current districts start to violate balancing constraints). In this sense, we assume that a reference solution $S^0 = (S^0_1, \dots, S^0_p)$ exists which corresponds to a previous districting plan, which has been optimized for an instance with different attribute vectors.

To avoid major changes to plan S^0, which can be costly, it is often desired that solutions produced by automated districting do not deviate much from S^0. This is commonly modeled by either a *global similarity* requirement, whereby any solution S should be similar to S^0 according to some global similarity metric $\sigma_G(S, S^0)$, or a *local similarity* requirement, such that districts $S_i, i \in P$ in a solution S should individually be similar to the corresponding district S^0_i in S^0, according to a local similarity metric $\sigma_L(S_i, S^0_i)$.

Although as we saw in Section 2.3.4 some authors model similarity as an objective, the most common approach is to model similarity as constraints that set lower limits to the global or local similarities of a solution. In this section, we take the latter

approach. This leads to the following optimization model, which follows the structure of Model (2.1):

$$\begin{aligned}
\underset{S \in \mathcal{S}}{\textbf{minimize}} \quad & C(S) && && \text{(3.43a)}\\
\textbf{subject to} \quad & \sigma_G(S, S^0) \geqslant \overline{\sigma^G} && && \text{(3.43b)}\\
& \sigma_L(S_i, S_i^0) \geqslant \overline{\sigma^L} && \forall i \in P, && \text{(3.43c)}\\
& L_a \leqslant w_a(S_i) \leqslant U_a && \forall i \in P, a \in A, && \text{(3.43d)}\\
& \text{connected}(S_i) && \forall i \in P. && \text{(3.43e)}
\end{aligned}$$

It minimizes compactness subject to balancing (3.43d), connectivity (3.43e), and similarity constraints. Constraint (3.43b) imposes a lower limit $\overline{\sigma^G}$ to the global similarity, while constraints (3.43c) impose a lower limit $\overline{\sigma^L}$ to the local similarity of each district. We assume that always at most one of (3.43b) and (3.43c) will be active, i.e., either $\overline{\sigma^G} = 0$ or $\overline{\sigma^L} = 0$.

For the same reasons given in Section 3.6.1, for the experiments in this section we fix $C = C_{pm}$. Still, the changes introduced here work identically irrespective of compactness measure.

3.7.1.1 Similarity measures

We recall from Section 2.3.4 the definition of overlap distance

$$o_a(S_i, S_j) = w_a(S_i \cap S_j) / w_a(S_i \cup S_j) \tag{3.44}$$

between two districts with respect to some attribute a. Using it, we define the local similarity between two districts as the total overlap distance between them, for all attributes:

$$\sigma_L(S_i, S_i^0) = \sum_{a \in A} o_a(S_i, S_i^0). \tag{3.45}$$

This definition is different from the typical approach which defines overlap distances over area, i.e., over the unitary attribute a with $w_a(S_i) = |S_i|$ [22, 23, 182], but is more generic since it can also be extended to include area.

With the above, we define the global similarity of a solution as the average local similarity

$$\sigma_G(S, S^0) = \sum_{i \in P} \sigma_L(S_i, S_i^0)/p. \tag{3.46}$$

In this way σ_G has range $[0,1]$, with $\sigma_G(S) = 0$ for totally dissimilar solutions and $\sigma_G(S) = 1$ for identical ones.

One may recall from Section 2.3.4 that many authors use a mapping $\mathfrak{m}$ between districts in S^0 and a newly constructed solution, in order to match district pairs for which similarity must be considered. We do not use this approach. As we shall see in Section 3.7.2.1, during solution construction we seed new districts from the district with same index in S^0, and thus ensure that $\mathfrak{m}(i) = i$ is always a feasible match.

3.7.2 Extending the heuristic to include similarity constraints

In the following we describe how we have adapted our heuristic to include similarity constraints. As in Section 3.6.2, since the core of the method remains unchanged, we discuss only our changes.

3.7.2.1 Changes to solution construction

We changed procedure greedyConstructive to generate initial solutions which always satisfy similarity constraints (3.43b) and (3.43c). To achieve this, instead of using a dispersion heuristic to generate initial seeds, we seed each district S_i with a unit s_i chosen uniformly at random from S_i^0. Then, when expanding the districts, while either $\sigma_L(S_i, S_i^0) \geqslant \overline{\sigma^L}$ or $\sigma_G(S, S^0) \geqslant \overline{\sigma^G}$ greedyConstructive restricts the set of candidate units to include in district S_i to $\partial(S_i) \cap S_i^0$. After feasibility is guaranteed, the construction proceeds normally.

3.7.2.2 Handling similarity during neighborhood search

We considered two different strategies to handle similarity: as hard constraints and as soft constraints. We compare them experimentally in Section 3.7.3.2.

When using hard constraints, only neighboring moves which result in a similarity above lower limits $\overline{\sigma^L}$ and $\overline{\sigma^G}$ are allowed during neighborhood search. This ensures intermediate solutions are always feasible with respect to similarity constraints, but may significantly reduce the number of moves available.

When using soft constraints, as we have done for routing budgets in Section 3.6.2 we extend the definition of imbalance to include violations to similarity constraints.

Therefore, we define

$$\begin{aligned} E(S) = & \sum_{a \in A, i \in P} b_a(S_i)/\mu_a \\ & + \sum_{i \in P} \max\{0, \sigma_L(S_i, S_i^0) - \overline{\sigma^L}\}/\sigma_L(S_i, S_i^0) \\ & + \max\{0, \sigma_G(S, S^0) - \overline{\sigma^G}\} \end{aligned} \tag{3.47}$$

as the sum of constraint violations to constraints (3.43b), (3.43c) and (3.43d), and replace every call to $B(S)$ in the main algorithm with a call to $E(S)$. We normalize the violations to local similarity constraints so they are in the same order of magnitude as the other two. Note that σ_G is already in $[0, 1]$.

3.7.2.3 Maintaining similarity dynamically

We can compute both local and global similarities dynamically under shifts and swaps in $O(p)$. We maintain, for each district $i \in P$ and attribute $a \in A$, the total attribute values $\iota_i^a = w_a(S_i \cap S_i^0)$ and $\omega_i^a = w_a(S_i \cup S_i^0)$ of S_i's intersection and union, respectively, with its corresponding district in the reference solution. Using ω and ι we can compute $\sigma_L(S_i, S_i^0) = \sum_{a \in A} \iota_i^a / \omega_i^a$, and $\sigma_G(S, S^0) = \sum_{i \in P} \sigma_L(S_i, S_i^0)/p$, both in $O(p)$.

Values ω and ι are maintained in constant time after neighboring operations. Initially, we set $\iota_i^a = 0$ and $\omega_i^a = w_a(S_i^0)$ for all $i \in P, a \in A$. Then, when a district S_i gains a unit u, we set

$$\iota_i^a \leftarrow \iota_i^a + w_u^a[u \in S_i^0], \tag{3.48}$$

$$\omega_i^a \leftarrow \omega_i^a + w_u^a[u \notin S_i^0], \tag{3.49}$$

and when a district S_i loses a unit u, we set

$$\iota_i^a \leftarrow \iota_i^a - w_u^a[u \in S_i^0], \tag{3.50}$$

$$\omega_i^a \leftarrow \omega_i^a - w_u^a[u \notin S_i^0]. \tag{3.51}$$

3.7.3 Computational experiments

In this section we report on computational experiments to assess our extension to similarity constraints. We implemented and ran our modifications under the same

environment described in Section 3.5. For these experiments we used instance sets SRC, RF and RS. Like for routing, to avoid a recalibration we use parameter values $t = 1.5p$, $A_{max} = 100$ and $I_{max} = 100$, use a fixed seed, and limit runs to both 1000 multistart iterations and 10 minutes of runtime. In Section 3.7.3.1 we extend the instance sets of Section 3.4 to redistricting through attribute modification models. Next, in Section 3.7.3.2 we evaluate the effectiveness of using soft versus hard constraints for maintaining similarity. Then, in Section 3.7.3.4 we assess the effect of different similarity metrics.

3.7.3.1 Modifying existing instances towards redistricting problems

Since the instance sets of Section 3.4 do not include data from previous districting plans, for each instance we have generated both a reference solution S^0 and modified attribute vectors to model this. Note that instance topologies remain the same. Solution S^0 is taken as the best solution found for the p-median objective in each instance over the experiments of Section 3.5. To model changes in attribute values over time we use a discrete diffusion model, detailed in the following.

Instead of the input graph $G = (V, E)$ we consider its directed version $G^d = (V, A)$, with $A = \{(u, v) \mid u, v \in V, \{u, v\} \in E\}$, where each edge $\{u, v\} \in E$ has been expanded into two arcs (u, v) and (v, u). A diffusion model is defined by a doubly stochastic matrix $P^a = (p^a_{uv})_{u,v \in V}$ for each attribute $a \in A$, i.e., $\sum_{u \in V} p^a_{uv} = 1$ for all $v \in V$ and $\sum_{v \in V} p^a_{uv} = 1$ for all $u \in V$. Here p^a_{uv} is the probability that a unit of attribute a at $u \in V$ diffuses to unit $v \in V$. In particular, p^a_{uu} is the probability of a unit of attribute a at unit u remaining at its current unit. Diffusion is limited to neighbors, meaning $p^a_{uv} = 0$ for all $(u, v) \notin A$. In one step of the diffusion, values of attribute $a \in A$ change to $P^a r^a$ where $r^a = (r^a_v)_{v \in V}$ is the vector of the a-th attribute's values.

We consider two diffusion models: a *uniform* model and a *migration* model. In both we define a basic probability p_r for an attribute unit to remain at the current basic unit, and set $p^a_{vv} = p_r$ for all $a \in A, v \in V$. In our experiments we used $p_r = 0.95$ which models a high probability of remaining at the current unit. We then distribute the remaining probability $1 - p_r$ over the neighbors of a unit. In the uniform model, we define $p^a_{uv} = (1 - p_r)/|N(u)|$ for all $v \in N^+(u)$, where $N^+(u)$ are the (outgoing) neighbors of unit u in graph G^d. This represents an unspecific diffusion to the neighbors.

In the migration model, we model the concentration of attributes in centers. This usually applies to the population. To this end we use the gravity model

Table 3.12: Results for the uniform model after 10 steps, using soft and hard similarity constraints.

Inst.	n	b_0 (%)	Hard constraint			Soft constraint		
			b (%)	C (r.d.)	Sim. (%)	b (%)	C (r.d.)	Sim. (%)
SRC	60	100.0	100.0	0.0	100.0	100.0	0.0	100.0
SRC	80	100.0	100.0	0.0	100.0	100.0	0.0	100.0
SRC	100	85.0	100.0	2.0	100.0	100.0	6.4	100.0
SRC	120	80.0	100.0	1.5	100.0	100.0	1.0	100.0
SRC	150	70.0	100.0	2.3	100.0	100.0	2.3	100.0
SRC	200	80.0	100.0	0.7	100.0	100.0	3.6	100.0
RF/DS	500	30.0	100.0	−17.2	100.0	100.0	−17.1	100.0
RF/DT	500	85.0	100.0	−1.7	100.0	100.0	−2.0	100.0
RS/DT	1,000	0.0	100.0	−21.4	100.0	100.0	−22.4	100.0
RS/DU	1,000	70.0	100.0	−1.2	100.0	100.0	−1.1	100.0
Avg./Tot.		70.0	100.0	−3.5	100.0	100.0	−2.9	100.0

of migration that assumes that interaction between places u, v is proportional to $I(u,v) = (r_u^a)^{\alpha_u}(r_v^a)^{\alpha_v}/d_{uv}^{\gamma}$ for some attribute $a \in A$ [203]. Following Poot et al. [168] we select values $\alpha_v \in U[0.8, 0.9]$ for all units $v \in V$, and $\gamma \in U[0.8, 0.9]$. As before, we set $p_{vv}^a = p_r$ for all $v \in V$, and distribute the remaining probability $1 - p_r$ over the neighboring units. For each neighbor $u \in N(v)$ we define a utility value of $U_v(u) = I(u,v)$. Then the probability of a attribute value r_v^a migrating to neighbor u is $p_{vu}^a = U_v(u)/\sum_{u \in N(v)} U_v(u)$.

3.7.3.2 Comparison of hard and soft constraints for similarity

In our first experiment we analyze whether it is better to handle similarities as a hard or a soft constraint. Recall that when treated as hard constraints, local and global similarities are always maintained above the minimum values $\overline{\sigma^L}$ and $\overline{\sigma^G}$. Consequently, initial solutions that do not satisfy minimum similarity constraints, and neighborhood search moves that violate them are discarded. On the other hand, when treated as a soft constraint we allow arbitrarily low similarities, in particular after improving the compactness, but insist in subsequent balancing steps on a feasible solution.

For this comparison we generated modified instances with a uniform diffusion model of 10 steps, and used $\sigma_L = 0.8$ and $\sigma_G = 0$, i.e., only local similarity enabled. Table 3.12 shows the results when using hard and soft constraints. For a more detailed comparison, here we report separately the results for instances of type DS and DT of

Table 3.13: Results for the uniform model after 5 and 10 steps when similarity is a hard constraint.

Inst.	n	5 steps				10 steps			
		b_0 (%)	b (%)	C (r.d.)	Sim. (%)	b_0 (%)	b (%)	C (r.d.)	Sim. (%)
SRC	60	100.0	100.0	0.0	100.0	100.0	100.0	0.0	100.0
SRC	80	100.0	100.0	0.0	100.0	100.0	100.0	0.0	100.0
SRC	100	90.0	100.0	1.5	100.0	85.0	100.0	2.0	100.0
SRC	120	100.0	100.0	−0.8	100.0	80.0	100.0	1.5	100.0
SRC	150	70.0	100.0	2.1	100.0	70.0	100.0	2.3	100.0
SRC	200	100.0	100.0	−1.1	100.0	80.0	100.0	0.7	100.0
RF/DS	500	40.0	100.0	−19.9	100.0	30.0	100.0	−17.2	100.0
RF/DT	500	90.0	100.0	−1.7	100.0	85.0	100.0	−1.7	100.0
RS/DT	1,000	33.3	100.0	−21.3	100.0	0.0	100.0	−21.4	100.0
RS/DU	1,000	76.7	100.0	−1.0	100.0	70.0	100.0	−1.2	100.0
Avg./Tot.		80.0	100.0	−4.2	100.0	70.0	100.0	−3.5	100.0

instance set RF, as well as types DT and DU of instance set RS. For each instance group we report the percentage b_0 of instances where reference solution S^0 is balanced in the diffused model, and for each variant the percentage b of balanced solutions obtained after optimization, the relative deviation $C\ (r.d) = (C(S^0) - C(S))/C(S^0)$, in percent, of each solution S's compactness from the compactness of the reference solution, and the percentage (Sim.) of solutions that satisfied similarity constraints.

We see that in all cases our method found a new feasible, balanced solution. The compactness of the instances after redistricting is sometimes smaller, and the improvement increases with the instance size. This can be explained by the additional computation time invested. Comparing compactness values, we see that the hard constraints overall work slightly better, and so we use this strategy in the next experiments.

3.7.3.3 Effect of different attribute modification models

In this experiment we look at both the uniform and gravity diffusion models. For each instance we run both models with 5 and 10 steps, and then solve it using similarity as a hard constraint. As before, we used $\sigma_L = 0.8$ and $\sigma_G = 0$.

Tables 3.13 and 3.14 show results for the uniform and gravity diffusion models, respectively, with the same columns as Table 3.12. In both models we observe that with an increasing number of diffusion steps, the percentage of initially balanced instances decreases. Similarly, relative p-median values generally increase. This is

Table 3.14: Results for the gravity model after 5 and 10 steps when similarity is a hard constraint.

Inst.	n	5 steps				10 steps			
		b_0 (%)	b (%)	C (r.d.)	Sim. (%)	b_0 (%)	b (%)	C (r.d.)	Sim. (%)
SRC	60	100.0	100.0	0.0	100.0	60.0	100.0	95.4	100.0
SRC	80	90.0	100.0	8.9	100.0	25.0	90.0	47.0	100.0
SRC	100	25.0	100.0	41.7	100.0	5.0	100.0	65.4	100.0
SRC	120	35.0	100.0	40.2	100.0	0.0	75.0	249.4	100.0
SRC	150	50.0	100.0	24.1	100.0	0.0	100.0	190.9	100.0
SRC	200	30.0	100.0	21.5	100.0	0.0	80.0	144.1	100.0
RF/DS	500	10.0	100.0	−4.1	100.0	0.0	100.0	37.2	100.0
RF/DT	500	10.0	100.0	3.3	100.0	0.0	100.0	11.7	100.0
RS/DT	1,000	0.0	100.0	−3.9	100.0	0.0	20.0	16.8	100.0
RS/DU	1,000	10.0	100.0	1.2	100.0	0.0	100.0	7.8	100.0
Avg./Tot.		36.0	100.0	13.3	100.0	9.0	86.5	86.6	100.0

expected, since the attribute distributions get increasingly different, which require a compromise in compactness in order to simultaneously satisfy balancing and similarity constraints. Instance sets RF/DS and RS/DT are exceptions to this, however, and in all cases the heuristic found better p-median values after attribute diffusion. This is explained by the fact that diffusion models lead to a concentration of attributes in regions of small Euclidean area, which allows for balanced districts to exist which are more compact.

Comparing both models we see that the percentage of initially infeasible instances is much lower in the gravity model, with 36% and 9% after 5 and 10 steps. The reason for this is that the gravity model is a global model where all basic units interact, and thus the change in attributes is comparatively higher. Furthermore, the gravity model leads to a concentration of attribute values. As a consequence these instances are more realistic but also harder to solve. This is corroborated by higher p-median values. After 5 steps the heuristic still is able to find a feasible solution for all instances, but after 10 steps the heuristic fails in 15% of instances when using the gravity model, on average. In all cases, solutions satisfied the minimum similarity constraint.

In Table 3.15 we show performance data for the gravity model. For 5 and 10 steps we present the average number of iterations (Iter.), the average time find the best solution (t_b, in seconds), and the total runtime (t, in seconds). We can see that for the smaller instances with up to 200 units, the limit of 1000 iterations is reached in less than two minutes, and the best solution is found quickly, within at

Table 3.15: Number of iterations, time to best and total runtime for the gravity model after 5 and 10 steps when similarity is a hard constraint.

Inst.	n	5 steps			10 steps		
		Iter.	t_b	t	Iter.	t_b	t
SRC	60	1,000.0	0.0	3.9	1,000.0	0.1	7.3
SRC	80	1,000.0	0.0	14.9	1,000.0	0.1	20.0
SRC	100	1,000.0	0.2	17.2	1,000.0	0.5	24.4
SRC	120	1,000.0	0.1	34.7	1,000.0	1.6	51.6
SRC	150	1,000.0	0.2	38.6	1,000.0	2.0	64.4
SRC	200	1,000.0	2.0	62.8	1,000.0	19.6	103.0
RF/DS	500	968.6	147.6	546.2	974.1	198.7	512.7
RF/DT	500	1,000.0	85.3	436.6	998.8	115.1	468.9
RS/DT	1,000	1,000.0	216.2	429.2	1,000.0	199.4	414.1
RS/DU	1,000	259.8	158.3	600.0	268.5	240.0	600.0
Avg./Tot.		922.8	61.0	218.4	924.1	77.7	226.6

most 20 seconds. The larger instances also achieve close to 1000 iterations within 10 minutes, except for RS/DU which always hits the time limit after less than 300 iterations. The performance of the heuristics is mainly driven by instances size, and less dependent on the number of steps. We do not show results for the uniform model, which are very similar.

3.7.3.4 Effect of different similarity metrics

In this section we look at the difference between local and global similarity measures. We ran our algorithm twice for each instance after 5 and 10 steps of the gravity model: with $\overline{\sigma^G} = 0$ and $\overline{\sigma^L} = 0.8$, then with $\overline{\sigma^G} = 0.8$ and $\overline{\sigma^L} = 0$.

Table 3.16 shows, for each instance set and number of diffusion steps, the smallest local similarity $s_l = \min_{i \in P} \sigma_L(S_i, S_i^0)$ over all p districts and the global similarity $s_g = \sigma_G(S, S^0)$ obtained from the first and second runs, respectively. We find that final similarity values are always well above the lower limit of 80%, hinting that the difficulties in finding feasible solutions lie in the balancing step. Consequently, as expected similarity values decrease with an increasing number of steps, as balancing becomes more difficult.

Setting $\overline{\sigma^G} = 0.8$ ensures 80% of overall attributes are fixed to the same districts as in the reference solution, but this percentage may be larger or smaller for individual districts. On the other hand, setting $\overline{\sigma^L} = 0.8$ forces each district to individually be at least 80% similar to the corresponding district. As a consequence, to satisfy local sim-

Table 3.16: Local and global similarity in the gravity model after 5 and 10 steps.

Inst.	n	5 steps		10 steps	
		s_l (%)	s_g (%)	s_l (%)	s_g (%)
SRC	60	100.0	100.0	96.3	97.3
SRC	80	98.4	99.0	91.5	96.0
SRC	100	94.0	97.7	90.9	96.1
SRC	120	95.1	98.6	89.2	94.2
SRC	150	95.8	98.2	89.1	94.9
SRC	200	94.1	98.3	86.1	93.5
RF/DS	500	90.0	95.1	88.3	94.0
RF/DT	500	97.2	99.0	95.2	98.4
RS/DT	1,000	87.2	95.5	92.8	98.2
RS/DU	1,000	97.0	98.9	95.1	98.3
Avg./Tot.		94.9	98.0	91.5	96.1

ilarity constraints with lower limit $\overline{\sigma^L}$ guarantees also $\sigma_G(S, S^0) \geqslant \overline{\sigma^L}$, i.e., the feasible space induced by constraints (3.43b) in Model (3.43) is always stricter than the one induced by constraints (3.43c). The reverse is not true, however, since global similarity alone may lead to a solution where a few districts suffer a large change. Hence, we believe that imposing a local, per-district similarity lower bound is preferable.

3.8 Conclusions and outlook

In this chapter we have proposed a heuristic that is generally effective across the multiple districting variants encompassed by Model (2.1). Despite the existence of several domain-specific requirements, the three pillars of districting: compactness, connectivity and balance are present in nearly all problems. We believe these three criteria remain major facet-defining forces even when problems are extended with more constraints, and thus a method effective the core subproblem will be effective, too, when it is extended. In a similar vein, although they differ in definition, as we have seen in Chapter 2 the many compactness measures are aimed at one goal: to achieve geometrically compact districts. Therefore, a method which is very effective for some compactness measure will remain effective if it is swapped.

We have implemented and tested our method on the three most common districting subproblems: the p-Median, p-Center and Diameter Districting Problems with multiple attributes. We showed experimentally that it is effective in solving large instances of all three, and yields state-of-the-art results when compared to methods which are targeted at a single variant. Algorithmically, our method uses a multistart approach

which builds solutions greedily, then improves them by a search strategy that alternates between the optimization of the objective function and the problem constraints. When optimizing constraint violations, to maintain past progress we limit solutions to a maximum worsening of the objective. A key advantage to this approach is that, since the optimization of both objective and constraints are decoupled, we can use different heuristics which may be more suitable to certain criteria, e.g. by taking advantage of underlying problem structures. In our case, we use tabu searches over specific neighborhoods to optimize compactness, and a series of tabu searches within a binary search strategy to optimize balance.

In separate studies we demonstrated the extendability of our heuristic by including two common criteria found in districting: similarity to a previous plan and routing costs. In both cases we showed that additional objective terms can be treated by redefining compactness function C as a weighted sum of criteria, and that local (per-district) and global (over all districts) constraints can be included by summing their relative violations to the imbalance B. Both modifications required few changes to the existing method, and did not significantly impact its performance. Additionally, we showed how existing instances might be adapted to include these requirements.

Finally, we proposed efficient algorithms to dynamically recompute compactness functions, balancing constraints, routing criteria and (dis)connectivity under changes incurred by neighborhood search. Experimentally this provided a significant speedup, since these computations are clear bottlenecks. These dynamic algorithms are not specifically tied to our method, and could be used in other optimization algorithms for districting, making them an independent contribution.

3.8.1 Outlook

We have identified four promising directions to improve our heuristic. First, in preliminary experiments we observed that the constructive heuristic and filtering mechanism of Section 3.3.1 play a large role in the final solution quality, which beckons a deeper analysis of filtering strategies [172]. Second, it would be worthwhile to investigate the addition of a shaking mechanism [141] to the alternating strategy. Currently, once the alternating search stagnates we discard the current solution and start a new multistart iteration, and so rely mainly on the constructive heuristic for variability. By applying a small perturbation to the current solution, i.e. "shaking" it, we may reach unvisited parts of the search space without foregoing the current improvements in balance and compactness. Third, in terms of search neighborhoods we

find that N_{shift} and N_{swap} often overlook longer improving paths that go over several worsening moves before finally improving the incumbent. To address, very large neighborhood search strategies [1] could be effective. We provide a deeper discussion of this issue in the next chapter, in Section 4.6, and show one such strategy that was effective for the Maximum Dispersion Problem. Fourth, we believe that due to its alternating nature that zigzags between solutions of e.g. high compactness and low imbalance, our solver could be effective in generating Pareto fronts for bi-objective formulations, which are common in districting. Therefore, it would be useful to develop a clear methodology to achieve this.

4 A HYBRID HEURISTIC FOR THE MAXIMUM DISPERSION PROBLEM

In this chapter we present our hybrid heuristic for the Maximum Dispersion Problem, published in the following article:

- A. Gliesch and M. Ritt. A hybrid heuristic for the maximum dispersion problem. *European Journal of Operational Research*, 288(3):721–735, 2021.

Abstract. In this chapter we propose a hybrid heuristic for the Maximum Dispersion Problem, which asks to find a balanced partition of a set of objects such that the shortest intra-part distance is maximized. In contrast to districting problems, dispersion problems aim for a large spread of objects in the same group. They arise in many practical applications such as waste collection and the formation of study groups. Our heuristic alternates between finding a balanced solution, and increasing the dispersion. Balancing is achieved by a combination of a minimum cost flow algorithm to find promising pairs of parts and a branch-and-bound algorithm that searches for an optimal balance, and the dispersion is increased by a local search followed by an ejection chain method for escaping local minima. We also propose new upper bounds for the problem. In computational experiments we show that the heuristic is able to find solutions significantly faster than previous approaches. Solutions are close to optimal and, in many cases, provably optimal.

4.1 Introduction

The Maximum Dispersion Problem (MaxDP) is to find a partition of a set of weighted objects such that the total weight of each part is close to a given target weight, and the *dispersion*, defined as the minimum distance between two objects in the same group, is maximal. Dispersion problems can be understood as the opposite of clustering problems, and arise in many practical situations. We give two example

applications from the literature.

The first is the design of waste collection territories in Germany in accordance with the European Waste Electrical and Electronic Equipment (WEEE) recycling directive [165]. This directive requires to assign waste collection stations to companies such that stations assigned to the same company are as dispersed as possible, in order to prevent regional monopolies. The problem extends the MaxDP by considering additional application-specific criteria, for example allowing the same station to be assigned to multiple companies. The main model for this application has been introduced by Fernández et al. [60], who also propose a GRASP algorithm to find heuristic solutions. Recently, Ríos-Mercado et al. [185] proposed a tabu search heuristic and Ríos-Mercado and Bard [179] an exact approach by mixed integer programming (MIP) to solve the same model.

The second application was considered first by Baker and Powell [10] and concerns the design of heterogeneous learning groups. Here, study groups comprised of students with backgrounds as different as possible are desired, to promote intercultural and interdisciplinary exchanges. The distances between students are given by the weighted differences between binary attribute vectors associated with each student. Thus, the problem can also be modeled as finding balanced groups of maximum dispersion.

The MaxDP was introduced by Fernández et al. [61] as a generalization of the two applications above. They propose an exact solution method based on mixed integer programming. It uses the fact that an instance of the MaxDP with n objects has at most $\binom{n}{2}$ solution values and iteratively solves a series of models with fixed upper and lower bounds on the solution value. The exact method is reported to solve instances of sizes up to 700 objects for instances of the WEEE application, and 300 objects for study group instances. Fernández et al. [61] also show that the MaxDP is NP-hard, by reduction from the Partition Problem. Moeini et al. [150] have also proposed a variable neighborhood search (VNS) to heuristically find solutions for the MaxDP. However, the authors consider only instances of the WEEE application of relatively small size (up to 500 objects), which can be solved exactly by the method of Fernández et al. [61].

Upper bounds on the MaxDP can be obtained by relaxing the constraints on the total group weight. The resulting *unrestricted MaxDP* (UMaxDP) problem can be solved by reduction to a series of graph coloring subproblems. Fernández et al. [61] propose an upper bounding scheme which considers the optimal solution of a set of smaller, heuristically generated subinstances. Ríos-Mercado and Bard [179] consider

the design of recycling districts based on MIP models and also use upper bounds to UMaxDP in order to improve the models.

There are a number of related grouping problems in the literature. In contrast to the MaxDP, which asks for dispersed groups, as we saw in Chapter 2 districting problems generally ask for units to be assigned to contiguous and compact groups. The Capacitated Clustering Problem [156] seeks groups of a given maximum overall weight and minimizing the sum of distances from the median object of each group. In terms of dispersion-based objectives, the Maximally Diverse Grouping Problem, first studied by Arani and Lotfi [7], asks that objects be partitioned into groups meeting balancing criteria and that maximize the sum of intra-group distances. Brimberg et al. [25] and Lai and Hao [124] have proposed heuristics based on incremental neighborhoods to solve it. In the context of location theory, the p-Dispersion Problem [55] asks for facilities to be located such that the minimum distance between two facilities is maximized. Some problem variants consider equity measures such as the minimum differential dispersion among selected facilities among a set of candidates [170, 51].

In this chapter we propose a heuristic to solve the MaxDP. Starting from a greedy initial solution, it iteratively alternates between improving the dispersion and balancing the solution. The heuristic stops when the dispersion cannot be improved, or optimality can be shown, by a matching upper bound.

The main contributions of this chapter are:

1. a hybrid method combining a local search to improve dispersion with an effective ejection chain algorithm to resolve conflicts, which operates in the space of imbalanced solutions,
2. an algorithm for balancing solutions, that repeatedly selects two promising groups based on information computed by solving a minimum cost flow problem and applies a truncated branch-and-bound that searches for the best strategy to balance them,
3. a proof that two existing upper bounds from the literature are the same, and a new family of better upper bounds, together with algorithms to compute them efficiently,
4. a new set of large, challenging instances,
5. a detailed computational analysis that shows that the overall heuristic can solve more instances than previous methods in one to two orders of magnitude less time, and expands the limit of solvable MaxDP instances from 800 to about 4000 objects.

Although the method is heuristic, combined with the new upper bound many solutions are provably optimal; in particular, we solve more instances to optimality than previous approaches.

The rest of this chapter is organized as follows. In the next section we give a formal definition of the problem. In Section 4.3 we review existing upper bounds and propose an improved upper bound. The proposed heuristic algorithm is outlined in Section 4.4, and Sections 4.5 and 4.6 explain in detail the algorithms used to improve the dispersion and to balance solutions. In Section 4.7 we report on experiments that assess the effectiveness of the proposed heuristic, as well as each of its components. We conclude in Section 4.8.

4.2 Problem definition

An instance $I = (V, m, d, a, \alpha)$ of the MaxDP is defined by a set of objects V of size $n = |V|$, the desired number of groups m, a matrix $d \in \mathbb{R}_+^{n \times n}$ of distances the objects, a vector $a \in \mathbb{R}_+^n$ of object weights, and a balancing tolerance parameter $\alpha \in [0, 1]$. Note that we use a different notation from the one introduced for districting in Chapter 1, in order to be consistent with the MaxDP literature [61]. Let $R = \{ d_{ij} \mid i, j \in V \}$ be the set of unique distance values, and $d^1 < \cdots < d^{|R|}$ denote these distances in increasing order. For convenience, we also introduce a value $d^0 < d^1$.

We define a solution as a function $S : V \to [m]$ mapping objects to groups. We allow S to be partial. In a partial solution the objects in $V \setminus \mathrm{dom}(S)$ are *unassigned* or *free*. A solution S is *complete* if all objects are assigned, i.e., $\mathrm{dom}(S) = V$. We write $S_k = S^{-1}(k)$ for the set of objects assigned to group k in solution S.

The dispersion of group k is defined as $D(S_k) = \min_{i,j \in S_k} d_{ij}$, and the dispersion of solution S as $D(S) = \min_{k \in [m]} D(S_k)$. Given some dispersion value d we call a pair of objects $\{i, j\}$ a d-*pair* in S if $d_{ij} = d$ and $S(i) = S(j)$, and a $D(S)$-pair a *critical* pair. Objects i and j are *conflicting* in S if $d_{ij} < D(S)$. Note that, by definition of D, conflicting objects cannot belong to the same group. Let $C(S)$ be the set of critical pairs in S. During optimization we often use a extended objective function $D'(S) = D(S) - \epsilon |C(S)|$, where $\epsilon \ll \min_{i,j \in V} d_{ij}$ is a small constant, to break ties and order solutions by the number of moves necessary to increase their dispersion. This is especially important for instances of type "study" (see Section 4.7.1) where object distances are integer and $|R| \ll \binom{n}{2}$, making $|C(S)|$ generally large.

Let $w(C) = \sum_{i \in C} a_i$ be the total weight of the set of objects C, and M_k be a given target weight for group k. We assume that $\sum_{k \in [m]} M_k = \sum_{i \in V} a_i$. The *imbalance* of

group k is the excess of the relative deviation of its weight from the target over α, namely $B(S_k) = \max\{0, -\alpha + |w(S_k) - M_k|/M_k\}$, and the imbalance of a solution S is the total imbalance $B(S) = \sum_{k\in[m]} B(S_k)$. The imbalance B represents the constraint violations of the classical balancing constraints given in the problem definition [61]. We say that group k is *balanced* if $B(S_k) = 0$, otherwise it is *imbalanced*. A solution S is *balanced* or *feasible* if all its groups are balanced. The tolerance parameter α allows a small deviation from the target weights, since the underlying packing problems may not always be feasible.

The goal of MaxDP is to find a balanced solution of maximum dispersion D. We also define the *unrestricted* MaxDP (*UMaxDP*, for short), which simply asks for a solution of maximum dispersion without balancing constraints. The UMaxDP is NP-hard, by reduction from the Vertex k-Coloring Problem [61]. Note that, since it is a relaxation of the original problem, the optimal solution value u^*_{ur} of the UMaxDP is an upper bound on the optimal value of the MaxDP.

For completeness we next give a mathematical model for the MaxDP. Let $x_{ik} \in \{0,1\}$ be a variable that indicates that object $i \in V$ is part of group $k \in [m]$. Then we can formulate

$$\textbf{maximize} \quad \min_{i,j\in V} \sum_{k\in[m]} d_{ij} x_{ik} x_{jk} \tag{4.1}$$

$$\textbf{subject to} \quad \sum_{k\in[m]} x_{ik} = 1, \qquad \forall i \in V, \tag{4.2}$$

$$M_k(1-\alpha) \leqslant \sum_{i\in V} a_i x_{ik} \leqslant M_k(1+\alpha) \qquad \forall k \in [m], \tag{4.3}$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in V, k \in [m]. \tag{4.4}$$

In this formulation, the objective function (4.1) maximizes the distance among any two object assigned to the same group. Note that the objective function is quadratic and contains a minimum; both elements can be linearized by standard techniques. Constraints (4.2) guarantee that each object is assigned to exactly one group, and constraints (4.3) make sure that solutions are balanced.

Finally, we define two operators over solutions, which are akin to the operators of the same names for districting problems we have defined in Section 2.4.2.3. A *shift* $i \to k$ reassigns object i to group $k \in [m] \setminus \{S(i)\}$. Similarly, a *swap* $i \leftrightarrow j$ swaps the assignments of objects i and j with $S(i) \neq S(j)$, i.e., i will be shifted to group $S(j)$ and j to $S(i)$. We use the notation $S[i \to k]$ and $S[i \leftrightarrow j]$ to refer to solution S after applying the respective operation, and denote the sets of all possible operations for solution S

as $N_{shift}(S)$ and $N_{swap}(S)$, for shifts and swaps, respectively.

In the rest of the chapter, when solution S is clear from the context we will omit S in unary functions such as B or D. Further, to make all procedures completely defined, unless otherwise stated we break ties by giving preference to a smaller object index i followed by a smaller group index k, when applicable.

4.3 Upper bounds

In this section we introduce new upper bounds for the UMaxDP. Since α is not relevant for the UMaxDP, we consider instances $I = (V, m, d)$ and write $D(I) = \max_{S \in S(I)} D(S)$ for the optimal solution value over all complete solutions $S(I)$ of I. We call $I' = (V', m, d)$ a k-*subinstance* if $V' \subseteq V$ has size $k = |V'|$. Let I^k be the set of all k-subinstances of instance I. Note that the optimal value of a k-subinstance I' of I is an upper bound on the optimal value of I, since any solution of I with given dispersion d when restricted to I' has dispersion at least d.

Fernández et al. [61] have proposed an upper bound $U^C = \min_{I' \in I^{m+1}} D(I')$, which is the least upper bound obtained by solving all $(m+1)$-subinstances of an instance I. An $(m+1)$-subinstance can be solved optimally in time $O(m^2)$ by computing the maximum distance between two objects and putting them in the same group, with all remaining $m-1$ items in a group by themselves. Since it is impractical to consider all $\binom{|V|}{m+1}$ subinstances, the authors propose a relaxed upper bound U_h^C which considers only a sample of all subinstances. Samples are generated heuristically by a greedy algorithm and improved by a local search.

Ríos-Mercado and Bard [179] show that $(m+2)$-subinstances can also be solved in time $O(m^2)$, and consequently extend the above idea to consider all $(m+2)$-subinstances. They define an upper bound $U^{RB} = \min_{I' \in I^{m+2}} D(I')$ as the least upper bound over all $(m+2)$-subinstances, and an efficiently computable heuristic version U_h^{RB}, which samples $(m+2)$-subinstances generated by a greedy algorithm. Empirically U_h^{RB} outperforms U_h^C on the tested instances. We refer the reader to the original papers for details on how U_h^C and U_h^{RB} are computed.

We generalize this idea to a family of decreasing upper bounds $U^1 \geqslant U^2 \geqslant U^3 \geqslant \cdots \geqslant U^{|V|-m}$, where U^σ is the least upper bound over all $(m+\sigma)$-subinstances. Observe that $U^1 = U^C$ and $U^2 = U^{RB}$, and $U^{|V|-m} = u^*_{ur}$ is the optimal solution of UMaxDP, since it contains all vertices.

In Section 4.3.1 we first explain the relation between UMaxDP and graph coloring and show that $U^1 = U^2$, and then propose in Section 4.3.2 a heuristic to generate

$(m+\sigma)$-subinstances for a given parameter σ, which can be solved by exact graph coloring algorithms in order to obtain better upper bounds.

4.3.1 The unrestricted MaxDP and graph coloring

The UMaxDP has a solution S with dispersion $D(S) > d$ iff the intersection graph $G_d = (V, E_d(V))$, where $E_d(V) = \{\{u,v\} \mid u,v \in V, d_{uv} \leqslant d\}$, is m-colorable. Such a solution can be obtained by computing a minimum vertex coloring for G_d. If there are at most m colors, each color can form a group, and by definition of the intersection graph the distance between vertices in the same group cannot be d or less, so $D(S) > d$ follows. Thus, the UMaxDP can be solved by finding the largest d^i, $i \in [|R|]$ for which $G_{d^{i-1}}$ is m-colorable.

Bounds can also be formulated by considering cliques. If the intersection graph G_d contains an $(m+1)$-clique it is not m-colorable, and thus d is an upper bound on the dispersion. Thus we obtain an upper bound by finding the smallest d^i such that contains an $(m+1)$-clique. Fernández et al. [61] show that upper bound U^C is equivalent to the smallest d^i such that $\omega(G_{d^i}) \geqslant m+1$, where $\omega(G)$ is the size of the largest clique in G.

Since the size of any clique of a graph G is a lower bound on the size of any coloring of G, the size of its largest clique, $\omega(G)$, is a lower bound on the size of the smallest coloring, its *chromatic number* $\chi(G)$. Note that if $G_{u^*_{ur}}$ has no chromatic gap (i.e., $\omega(G_{u^*_{ur}}) = \chi(G_{u^*_{ur}}) = m+1$) then $U^1 = U^C = u^*_{ur}$ is optimal. Otherwise, $G_{u^*_{ur}}$ has no $(m+1)$-clique and $u^*_{ur} < U^1$. Graphs with a chromatic gap tend to be harder to color by exact coloring algorithms, since the lower bound given by the clique number is weak [37]. In general, U^σ (and thus all U^k with $k \geqslant \sigma$) can still be optimal for larger chromatic gaps of $G_{u^*_{ur}}$. However, this does not hold for U^2, as the following theorem shows.

Theorem 4.3.1 *Upper bounds U^1 and U^2 are the same, i.e., $U^1 = U^2$.*

Thus, $U^2 = U^{RB}$ can never improve over $U^1 = U^C$. The sampling-based heuristic versions U^C_h and U^{RB}_h still can produce different results, if they fail to find the right subinstances that witness these upper bounds.

For the proof of Theorem 4.3.1 we use the following result of Dirac [48]:

Theorem 4.3.2 *If $0 \leqslant n \leqslant k-1$, a k-chromatic graph either contains a complete $(k-n)$-graph as a subgraph or has at least $k+n+2$ vertices.*

Proof. If $U^2 < U^1$, there must be an $(m+2)$-subinstance $I' = (V', m)$ that witnesses this, i.e., the intersection graph $W = G_{U^2}(V', E(V'))$ has $m+2$ vertices, and $\chi(W) = m+1$.

Now by Theorem 4.3.2 with $n = 0$ and $k = m+1$, W either contains a complete $(m+1)$-subgraph or has at least $m+3$ vertices. Since it has $m+2$ vertices it must contain a complete $(m+1)$-subgraph K. But then $U^1 = \min_{I' \in I^{m+1}} D(S) \leqslant D(K) \leqslant U^2$, since $K \in I^{m+1}$, contradicting our assumption. ■

4.3.2 An improved upper bound

Given a parameter $\sigma > 2$ and instance $I = (V, m, d)$, we search for an improved upper bound U_h^σ by searching for an $(m+\sigma)$-subinstance of I whose graph induced by the current upper bound is not $(m+1)$-colorable. If the search is successful, the current upper bound can be reduced, since the graph induced by u^*_{ur} must be $(m+1)$-colorable. The choice of σ must be small enough such that the coloring subproblems can be solved optimally within reasonable time, but at least as large as the smallest subgraph of $G_{u^*_{ur}}$ of chromatic number $m+1$. We calibrate parameter σ in Section 4.7.2.1.

Starting from an initial upper bound $d^u = U_h^C$, algorithm UB iteratively generates, for each $i \in V$, an $(m+\sigma)$-subinstance $I_i = (V_i, m, d) \subseteq I$ using a constructive heuristic seeded by object i, followed by a local search procedure, as explained below. For each I_i it then computes an exact coloring of the induced subgraph $H^i_u = (V_i, E_{d^u}(V_i))$. Here, we use the implementation of Lewis et al. [131] of the exact coloring algorithm of Korman [121]. If H^i_u is not m-colorable, then d^u is not optimal, and can be reduced. Algorithm UB then binary searches for the largest $1 \leqslant u' < u$ such that $\chi(H^i_{u'}) \leqslant m$, and updates u with this value. At most $O(n + \log_2|R|)$ exact colorings are computed. It processes the graphs H^i_u in order of decreasing degree of node i and stops if an improving subinstance is found or if a time limit is reached. In the end, UB returns the final upper bound $U^{GR}_\sigma = d^u$. The full method is given in Algorithm 6.

Each set V_i is constructed by starting with $V_i = \{i\}$ and iteratively adding the next object $j \in V \setminus V_i$ such that the number of edges $|E_{d^u}(V_i)|$ is maximized, with ties broken by maximum degree, until $|V_i| = m+\sigma$. Then, algorithm UB performs a first-improvement local search that swaps objects in V_i with objects in $V \setminus V_i$, again aiming to maximize $|E_{d^u}(V_i)|$. It iterates over the candidates in a round-robin fashion, starting from the first object index. Here, we search for subgraphs with as many edges as possible, as opposed to other criteria, since $|E_{d^u}(V_i)|$ is fast to compute and em-

Algorithm 6 Algorithm UB to compute an improved upper bound.

Input: an instance $I = (V, m, d)$ and a subinstance size $\sigma > 0$.
Output: an improved upper bound $U_h^\sigma \leqslant U_h^C$.

```
1:  procedure UB(I, σ)
2:      d^u ← U_h^C(I)
3:      let V = {v_1, ..., v_n} such that δ(v_1) ⩾ δ(v_2) ⩾ ··· ⩾ δ(v_n)
4:      for i ∈ [n] do
5:          V_i ← generateSubinstance(I, σ, d^u, v_i)
6:          H_u^i ← (V_i, E_{d^u}(V_i))
7:          if χ(H_u^i) > m then
8:              u ← binarySearch(V_i, 1, u − 1)
9:              break
10:     return d^u
```

pirically it has a large influence on the chromatic numbers. Smith-Miles et al. [200] have thoroughly studied the relevance of several other metrics to the difficulty of graph coloring instances, and found the density (and thus the number of edges, for a graph of fixed size) to be among the three most important. Algorithm 7 outlines this procedure.

Algorithm 7 Algorithm generateSubinstance to generate subinstances with high upper bounds.

Input: an instance $I = (V, m, d)$, a subinstance size $\sigma > 0$, the current upper bound index u, and a seed object $v_i \in V$.
Output: a subset $V_i \subseteq V$ of size σ such that $v_i \in V_i$.

```
1:  procedure GENERATESUBINSTANCE(I, σ, d^u, v_i)
2:      V_i ← {v_i}
3:      do
4:          j ← argmax{|E_{d^u}(V_i ∪ {j})| + δ(j)/n, j ∈ V \ V_i}
5:          V_i ← V_i ∪ {j}
6:      while |V_i| < σ
7:      do
8:          V_i' ← V_i
9:          for j ∈ V_i cyclically, in increasing order of index do
10:             for k ∈ V \ V_i cyclically, in increasing order of index do
11:                 if |E_{d^u}((V_i \ {j}) ∪ {k})| > |E_{d^u}(V_i)| then
12:                     V_i ← (V_i \ {j}) ∪ {k}
13:     while V_i ≠ V_i'
14:     return V_i
```

Note that since algorithm UB only considers a sample of n subgraphs, U_h^σ may not be as low as U^σ; we show in Section 4.7.2.2, however, that with an appropriate choice of σ it always improves existing upper bounds in practice. By starting with U_h^C, we ensure U^σ is at most U_h^C, but any feasible upper bound can be used as an initial value

for d^u. Here, we chose U_h^C rather than U_h^{RB} as it is faster to compute and empirically smaller.

4.4 A hybrid heuristic for the MaxDP

Algorithm 8 Hybrid alternating search heuristic "Hase".

```
1: ub ← UB()
2: S^0 ← infeasible solution
3: D(S^0) ← d^0
4: S^1 ← greedyConstructive()
5: repeat
6:     S^i ← optBal(S^i, D(S^i))
7:     if S^i is imbalanced then
8:         binary search for the largest d ∈ (D(S^{i−1}), D(S^i)) ∩ R s.t. B(optBal(S^i, d)) = 0
9:         if no such d exists then
10:            if i = 1 then
11:                return optimal solution of model (F), or S^0 if none found
12:            return S^{i−1}
13:        else
14:            return optBal(S^i, d)
15:    if D(S^i) = ub or time limit reached then
16:        return S^i
17:    i ← i + 1
18:    S^i ← optDisp(S^{i−1}, ub)
19: until D′(S^i) = D′(S^{i−1})
20: return S^i
```

We propose a hybrid heuristic which we call Hase (derived from "hybrid alternating search") to solve the MaxDP. Its overall structure follows our proposed alternating approach of Chapter 3 for solving districting problems. Algorithm 8 summarizes the method. Hase alternates between balancing solutions and increasing the dispersion. This is repeated until a time limit is reached or a feasible solution with a dispersion equal to the upper bound is found.

Algorithm optBal tries to balance a solution. Since balancing a solution may reduce its dispersion arbitrarily, optBal is restricted to produce only solutions with a minimal acceptable dispersion d. In the first trial d is set to the dispersion of the current solution, and thus the dispersion is not allowed to decrease. However, if this fails, Hase uses a binary search to find the largest dispersion that can be successfully balanced. The search interval includes all dispersion values between the last solution that could be balanced, and the first solution that could not. In this case a regress in disper-

sion is allowed in order to find a feasible solution. After finding the largest feasible dispersion, Hase assumes that no larger dispersion is feasible and stops.

Otherwise, if optBal is able to balance the current solution of highest dispersion, algorithm optDisp tries to increase the dispersion. To achieve this, it is allowed to produce imbalanced solutions. If the dispersion cannot be improved, the heuristic terminates. Otherwise, the next iterations starts to balance the improved solution, if required.

Algorithm optDisp starts from a balanced solution and stops whenever the dispersion improves, which typically entails only a small change. This is done to avoid a large regress in the solution's imbalance, such that from the second iteration of optBal starts from a nearly balanced solution, and thus tends to be much faster. Further, by doing small, incremental improvements the algorithm avoids terminating with an imbalanced solution when reaching the time limit. In the special case where optBal fails with $d = d^1$, in a final effort to find any feasible solution Hase tries to solve a feasibility MIP model (F), given by

$$\begin{aligned} \textbf{exists} \quad & x \in \{0,1\}^{V\times[m]} \\ \textbf{subject to} \quad & x \text{ satisfies (4.2), (4.3), and (4.4).} \end{aligned} \tag{F}$$

If model (F) is infeasible, then Hase returns S^0 with a proof of infeasibility; otherwise, it returns solution x. If (F) cannot be solved within the remaining time limit, Hase also returns S^0, but without an infeasibility guarantee. In the following subsection we explain the greedy construction of an initial, possibly imbalanced solution that will be balanced in the first iteration. Next, in Section 4.5.2 we explain algorithm optDisp, which tries to increase the dispersion by alternating between a local search and a tree-based ejection procedure, followed algorithm optBal in Section 4.6, which tries to balance a solution by a truncated branch-and-bound method.

4.4.1 Initial solutions

A greedy constructive algorithm generates initial solutions. It first seeds the m groups with m objects from an $(m+1)$-subinstance $I_{U_h^C}$ witnessing U_h^C (i.e., whose maximum pairwise distance is equal to $I_{U_h^C}$) generated using the method of Fernández et al. [61], excluding the object of lowest index that is part of the maximum pairwise distance. This is done to select initial seeds which are as close to each other as possible. No additional step is required to find these seeds, as $I_{U_h^C}$ is obtained when computing the upper bound in algorithm UB.

The algorithm then iteratively selects the group k with the smallest fraction of achieved target weight $w(S_k)/M_k$ and assigns to it a free object i which maximizes $D'(S[i \to k])$, with ties broken by minimum imbalance. This choice of the next assignment gives preference to initial solutions of high dispersion, as opposed to more balanced ones. Still, some initial balancing is ensured by preferring assignments with minimum imbalance. The algorithm stops when all objects have been assigned. Because both D' and B are recomputed in constant time upon object insertions, the solution is constructed from the seeds in time $O(n^2)$. Algorithm 9 outlines the method.

Algorithm 9 Algorithm greedyConstructive to generate initial solutions.

Output: an initial solution.

procedure GREEDYCONSTRUCTIVE
  let $(v_1, \ldots, v_{m+1}, k, d)$ be an $(m+1)$-subinstance witnessing U_h^C, s.t. $\exists j : v_j > v_{m+1} \wedge d_{v_j v_{m+1}} \geqslant d_{ik} \forall i, k \in [m]$
  $S_k \leftarrow \{v_k\} \forall k \in [m]$, inducing solution S
  do
    $k \leftarrow \operatorname{argmin}_{k \in [m]} w(S_k)/M_k$
    $i \leftarrow \operatorname{argmax}_{i \in [n] \setminus \bigcup_{l \in [m]} S_l} D'(S[i \to k]) - \epsilon B(S[i \to k])$
    $S_k \leftarrow S_k \cup \{i\}$
  while $\mathrm{dom}(S) \neq V$
  return S

4.5 Improving dispersion

Algorithm optDisp improves the dispersion of a solution by alternating between a local search (LS) and a recursive exchange procedure (EX) based on a tree search. LS repeatedly performs shift and swap operations, and stops if it cannot further improve the incumbent (see Section 4.5.1). Then, EX attempts to find a longer sequence of moves to escape the current local minimum (see Section 4.5.2). optDisp alternates between LS and EX until either the current extended dispersion D' cannot be improved, dispersion D has improved, the upper bound is achieved or a time limit is reached. At the end, if D has improved optDisp runs LS once more. Note that since 10 ignores balancing constraints, it may produce and work with infeasible solutions. Because it requires D' to improve at each iteration and stops once D is improved, optDisp iterates at most $|C(S)| = O(\binom{n}{2})$ times. Algorithm 10 outlines the method.

Algorithm 10 Algorithm optDisp to improve dispersion.

Input: a solution S^0 and an upper bound ub.
Output: a solution S such that $D'(S) \geqslant D'(S^0)$.

```
1: procedure OPTDISP(S^0, ub)
2:     S ← S^0
3:     do
4:         S' ← S
5:         S ← LS(S)
6:         S ← EX(S)
7:     while D'(S) ≠ D'(S') ∧ D(S^0) = D(S) < ub ∧ in time limit
8:     if D(S^0) < D(S) ∧ in time limit then
9:         S ← LS(S)
10:    return S
```

4.5.1 Local search

Algorithm LS searches neighborhoods N_{shift} and N_{swap} iteratively performing a shift or a swap which increases the dispersion D' until no more improving moves exist. Since $|N_{swap}(S)| \gg |N_{shift}(S)|$, LS first explores N_{shift} fully and considers swaps only if there is no improving shift. It uses a first-improvement strategy and explores each neighborhood in a round-robin fashion, i.e., each call to LS continues the search cyclically from the point where the previous call stopped. Note that only operations on objects in critical pairs can improve $D'(S)$, so LS considers the reduced neighborhoods $\{i \to k \mid i \in \bigcup C(S), k \in [m] \setminus \{S(i)\}\}$ for shifts and $\{i \leftrightarrow j \mid i \in \bigcup C(S), j \in [n] \setminus S(i)\}$ for swaps. These neighborhoods have sizes $O(|C|m)$ and $O(|C|n)$, respectively. Algorithm 11 outlines the method.

The number of critical pairs $|C|$ varies according to the instance and can potentially be as large as $\binom{n}{2}$, if all objects are in the same location. In practice, however, instances of the WEEE application typically have $|C| < 10$, while instances of study group design have more duplicate distances and average $|C| \approx n/m$. Using a dynamic data structure to update D' in time $O(n^2 \log n)$ and evaluate D' for candidate moves in time $O(m)$, each iteration takes $O(n^2(m + \log n))$ time, since the swap neighborhood dominates.

4.5.2 An ejection chain algorithm for improving dispersion

Algorithm EX searches for an improving ejection chain [85]. The idea is to eject some object $i \in \bigcup C(S)$ from its current group (i.e., free that object), obtaining a partial solution with better $D'(S)$, and then to insert i into some other group $k \neq S(i)$.

Algorithm 11 Local search LS to improve dispersion.

Input: a solution S^0.
Output: a solution S such that $D'(S) \geqslant D'(S^0)$.

```
 1: procedure LS(S^0)
 2:     S ← S^0
 3:     do
 4:         S' ← argmax{D'(S[i→k]) | i ∈ ⋃C(S), k ∈ [m] \ {S(i)}}
 5:         if D'(S') > D'(S) then
 6:             S ← S'
 7:             continue
 8:         S' ← argmax{D'(S[i↔j]) | i ∈ ⋃C(S), S(j) ≠ S(i)}
 9:         if D'(S') > D'(S) then
10:             S ← S'
11:     while D'(S) = D'(S') ∧ in time limit
12:     return S
```

This can lead to a sequence of accommodating moves. Insertions are recursively done as follows. Let d' be the dispersion $D(S)$ before the ejection of i and $\kappa(i,k,S) = \{j \mid j \in S_k, d_{ij} \leqslant d'\}$ be the set of conflicting objects induced by the placement of object i into some group k. When attempting to insert i into k, EX ejects every object $j \in \kappa(i,k,S)$ from k, and recursively tries to reinsert each object j into the solution using the same procedure. Note that $\kappa(i,k,S)$ is non-empty for all k at recursion depth 0, since we assume that the current solution is a local minimum of $N_{\mathtt{shift}}$. If an object is moved to another group, it is *fixed*, and cannot be moved again in that recursion branch. Reinserting an object i is considered *successful* if i is not fixed and inserting all $j \in \kappa(i,k,S)$ is successful for some $k \in [m]$ (the basic case is $\kappa(i,k,S) = \emptyset$, i.e., there are no conflicts).

This process can also be described by an AND/OR tree. Here, OR nodes branch on the possible groups an ejected object can be moved into, and only one branch must succeed for the OR node to be considered feasible, whereas AND nodes branch on the conflicts induced by an insertion, and all branches (each, in turn, an OR node representing a conflict) must be successfully placed for the AND node to be feasible.

Algorithm EX considers the possible receiving groups $k \in [m]$ in order of non-decreasing number of conflicts $|\kappa(i,k,S)|$, i.e., it prefers groups which will eject the least number of objects, and thus likely require fewer recursive calls. We have found this to be significantly better than an arbitrary group ordering. EX discards groups k where $\kappa(i,k,S)$ contains a fixed object, since they cannot be successful. When reinserting the ejected objects, it first tries objects which are more likely to fail: since all objects in κ must be inserted, a single failure allows to fathom unsuccessful nodes

early. To this end, EX keeps track of the number $\varphi(i,k)$ of unsuccessful placement attempts for each object i and group k and the total number of failed insertions $\varphi(i) = \sum_{k\in[m]} \varphi(i,k)$, and attempts to reinsert ejected objects in decreasing order of $\varphi(i)$, with ties broken by i. The value of φ is maintained across multiple calls to EX.

Since at least one object is fixed at each node in the recursion tree, the recursion depth is bounded by n. However, this still leads to a search space that is too large to be explored in practical time. Therefore, inspired by the well-known heuristic of Lin and Kernighan [137] for the TSP, we introduce parameters p_1, p_2 and p_3 such that:

- at depth $d \leqslant p_1$ EX expands all possible insertions of the current object into other groups,
- at depth $p_1 < d \leqslant p_2$ EX only recurses on groups k with a single conflict ($|\kappa(i,k,S)| \leqslant 1$),
- at depth $p_2 < d \leqslant p_3$ EX adopts an aggressive backtracking approach and recurses only on the group k with smallest lexicographic $(\varphi(i,k),k)$ among the groups with a single conflict,
- at depth $d > p_3$, if there are still conflicts the node is pruned, i.e., the current branch is declared unsuccessful.

Algorithm 12 shows the ejection chain approach. It processes the objects $i \in \bigcup C(S)$ in order of non-decreasing $\varphi(i)$, i.e., it prefers objects which have been easy to place, since only one successful insertion is needed. Each of these objects is ejected and then EX attempts to reinsert it, and stops as soon as a reinsertion is successful (since in this case $D'(S)$ must have improved; otherwise, there would still be unassigned conflicting nodes). Assuming groups have on average n/m objects, each recursive call takes time $O(n)$ at depth $d \leqslant p_2$, and $O(n/m)$ time at depth $p_2 < d \leqslant p_3$.

Figure 4.1 shows the EX procedure on an example with $m = 3$ and $(p_1,p_2,p_3) = (1,2,3)$. We represent OR nodes as squares and AND nodes as circles. Consider the critical pair set $C = \{\{u_1,u_2\}\}$ with $\{u_1,u_2\} \subseteq S_3$, and assume this is the first call to EX, i.e., $\varphi(u) = 0$ for all u. Since $\varphi(u_1) = \varphi(u_2)$, the tie is broken lexicographically and EX starts by trying to eject u_1. At depth $d = 1$ it can move u_1 to either S_1 or S_2. Since $|\kappa(u_1,S_2)| < |\kappa(u_1,S_1)|$, it explores the branch $u_1 \to S_2$ first. This branch induces the ejection of u_3, which can, in turn, be placed in S_1 or S_3. Since the search is now at depth $d = 2 > p_1$, subtree $u_3 \to S_3$ is pruned since $|\kappa(u_3,S_3)| > 1$. Next, branch $u_3 \to S_1$ induces the ejection of u_4, which can be placed in either S_2 or S_3. Because the search depth now is $3 > p_2$, however, EX prunes $u_4 \to S_3$ since it can only

Algorithm 12 The full ejection chain algorithm to improve dispersion.

Input: a complete solution S^0.
Output: a complete solution S with $D'(S) \geqslant D'(S^0)$ if successful, or S^0 on failure.

```
 1: procedure EX(S^0)
 2:     for i ∈ ⋃C(S^0) in order of non-decreasing φ(i) do
 3:         S ← ejectionChain(S^0, ∅, i, 0)
 4:         if S ≠ S^0 then
 5:             return S
 6:     return S^0
 7:
 8: procedure EJECTIONCHAIN(S^0, F, i, d)
 9:     S ← S^0; F ← F ∪ {i}
10:     first ← true
11:     for k ∈ [m] \ {S^0(i)} in order of increasing (|κ(i,k,S)|, φ(i,k), k) do
12:         if κ(i,k,S) ∩ F ≠ ∅ then
13:             continue
14:         if (d > p_3 and κ(i,k,S) ≠ ∅) or (d > p_2 and ¬first) or (d > p_1 and
    |κ(i,k,S)| > 1) then
15:             break
16:         for j ∈ κ(i,k,S) in order of decreasing φ(j) do
17:             S' ← ejectionChain(S, F, j, d+1)
18:             if S' = S then
19:                 continue next k-loop
20:             S ← S'
21:         return S
22:         first ← false
23:     return S^0
```

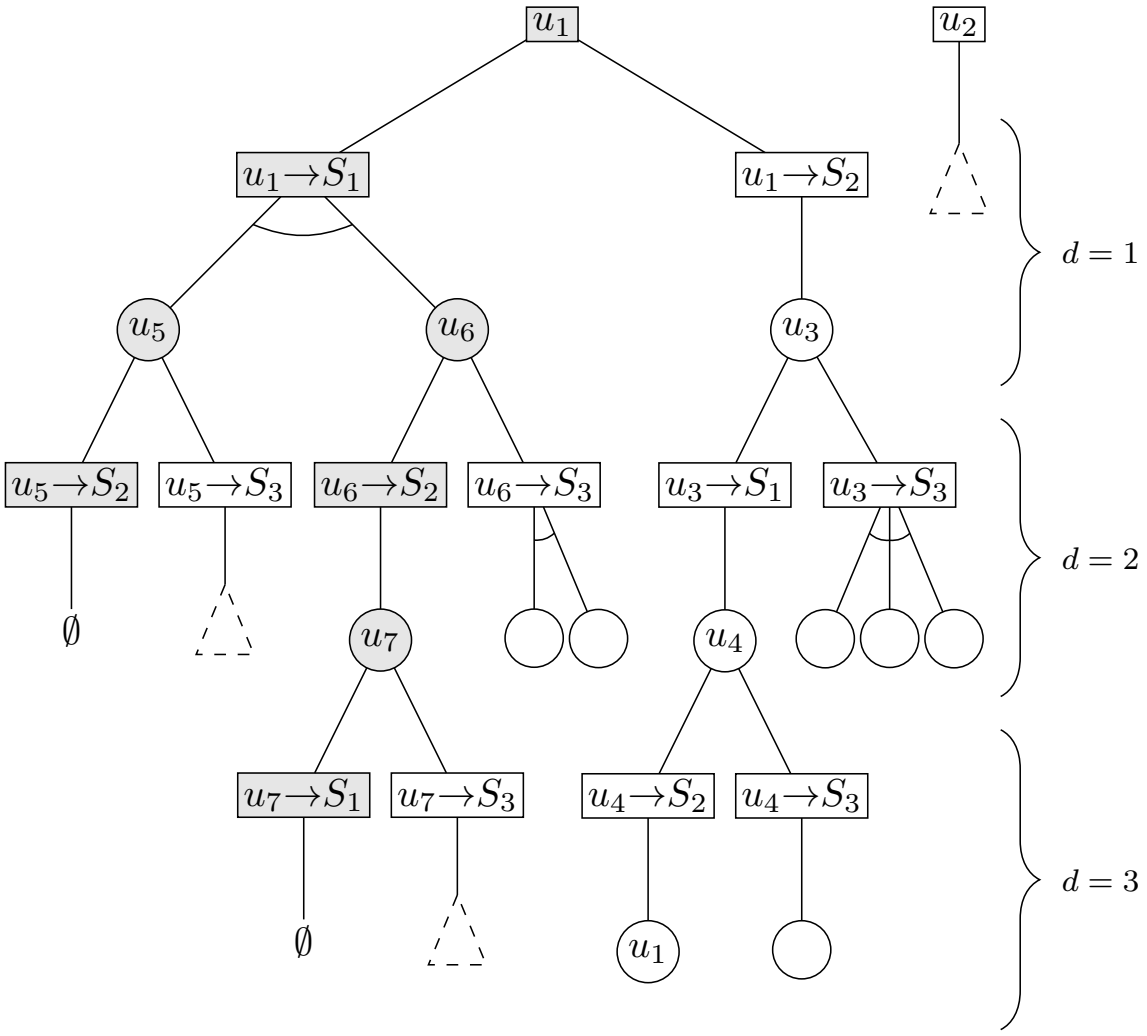


Figure 4.1: An example of the EX procedure.

explore the one branch (since $\kappa(u_4, S_2) = \kappa(u_4, S_3) = 1$ and $\varphi(u_4, S_2) = \varphi(u_4, S_3) = 0$ the tie was broken by group index). When visiting branch $u_4 \to S_2$ EX finds that u_1 generates a conflict. However, since u_1 is fixed and may not be ejected again, this branch is pruned and EX backtracks to $u_1 \to S_1$. Movement $u_1 \to S_1$ induces ejections u_5 and u_6. Because these are AND nodes, both u_5 and u_6 must be successfully placed in order to find a feasible sequence. Moving $u_5 \to S_2$ yields no conflicts and can be done immediately, regardless of subtree $u_5 \to S_3$. When reinserting u_6, EX prunes subtree $u_6 \to S_3$ since at depth $2 > p_1$ it has more than one child. Moving $u_6 \to S_2$ therefore causes u_7 to be ejected. Finally, u_7 is immediately placed in S_1 as $\kappa(u_7, 1, S) = \emptyset$. Since all AND nodes led to a leaf node with no conflicts, the set of movements $\{u_1 \to S_1, u_5 \to S_2, u_6 \to S_2, u_7 \to S_1\}$ is feasible and increases the dispersion.

4.6 Balancing solutions

A common approach to improve balance in similar problems is a variable neighborhood search (VNS) that first examines neighborhood N_{shift}, and if no improving shift can be found, neighborhood N_{swap} [25, 179]. If no improving move is found, the search continues from a perturbed solution. In our experience VNS-like methods for the MaxDP quickly reach local minima, and the perturbation step is typically not enough to escape them. This is mainly because these two operators can fail to explore large sections of the search space through improving paths. Although shifts are universal, in the sense that any solution can be transformed into any other using them, often finer adjustments to group weights can only be achieved by long, not strictly improving sequences of shifts. Different operators such as 3-chain [25] or double (triple) shifts [28], or other, more elaborate ones often help mitigate the issue, but they tend to have neighborhoods too large to be fully explored at each iteration.

We therefore propose a more flexible approach to generate longer move sequences which includes by design the shift and swap neighborhoods, outlined in Algorithm 13. It repeatedly selects two groups k and l and searches for a sequence of exchanges between them, which we denote as a *group pair exchange*, such that their relative imbalances are reduced, and the dispersion does not fall below some given limit d. If such a sequence can be found, it is applied, and the algorithm considers the next pair of groups. Group pair exchanges are obtained by a truncated branch-and-bound procedure bb (explained in Section 4.6.2) which aims at an optimal exchange of objects between two groups. Because the entire branch-and-bound tree must be exhausted if no improving exchange exists, which usually is too costly, this procedure expands at most BB_{max} nodes. Initially BB_{max} is set to BB^{lo}_{max}.

After each iteration, regardless of whether an improving exchange between groups k and l was found, the pair $\{k,l\}$ is marked as *tabu* (i.e., it can only be selected if no non-tabu moves exist) for τ iterations. Following the same rationale as in Section 3.5.1, we set $\tau = m$. Further, we denote a pair of groups $\{k,l\}$ as *hopeless* if an improving exchange between them is impossible. This happens if k and l are both balanced, or if the last exchange between k and l did not find an improving sequence and no other improving sequences incident to either k or l have been found since. Algorithm optBal never selects hopeless pairs. In optBal tabu and hopeless pairs are represented by the sets T and H, respectively. The algorithm stops if all pairs of groups are hopeless, meaning either the incumbent is balanced or exchange attempts for all imbalanced groups have failed. If the solution is still imbalanced, optBal dou-

bles BB_{max} and re-executes starting from the current solution, up to a maximum $BB_{max} = BB^{hi}_{max}$, at which it stops. In our implementation, we use $BB^{lo}_{max} = 2^{11}$ and $BB^{hi}_{max} = 2^{18}$.

In early tests we found that the main algorithm sometimes quickly converges to solutions where balancing fails even for low dispersion bounds, causing an early termination of the algorithm, but that small modifications to the input solution are often enough to allow balancing, if the instance is feasible. To exploit this, if balancing fails we perform a deterministic shuffling step to slightly modify the current solution, and re-execute the balancing algorithm. This is done at most ξ times, or until the resulting imbalance is larger than a threshold θ, above which another attempt at balancing is unlikely to succeed. In the end the solution with lowest imbalance B is returned.

To keep the all steps deterministic, shuffling is done as follows. The algorithm cyclically iterates over pairs of groups in lexicographical order, starting from $(1,1)$ in the first call and later from the point where the previous execution stopped. Given pair (k,l) it considers shifts $u \to l, u \in S_k$ in the order which the nodes in S_k are stored in memory. (This order is mostly arbitrary, and is determined by the sequence of operations done so far on S_k.) It applies the first shift s encountered for which $D(S[s]) \geqslant d$, and moves on to the next pair of groups. The algorithm stops after ρ shifts. We calibrate parameters ξ, θ and ρ in Section 4.7.4.1.

4.6.1 Selecting two groups

At each iteration, optBal selects the next pair of groups by solving a minimum cost flow problem on an auxiliary bipartite graph. The main idea is to select the two groups that would exchange the most weight if all possible shifts were allowed, regardless of decreases in the dispersion.

Let each group $k \in [m]$ be represented by two nodes v^s_k and v^t_k in different parts of the graph, where v^s_k is a source node with supply $w(S_k)$ and v^t_k is a sink node with demand M_k. A pair of nodes v^s_k and v^t_l which is not hopeless is connected by an arc (v^s_k, v^t_l) of different capacity and cost:

- If $k = l$ the capacity is $(1-\alpha)M_k$ and the cost 1. The low cost is an incentive for groups to maintain their lower target bound, and the capacity of $(1-\alpha)M_k$ forces them to send the excess over the lower bound to other groups.
- If $k \neq l$ and $\{k,l\}$ is not tabu, the capacity is ∞ and the cost 2. This allows groups to send any excess weight to other groups at a slightly higher cost.
- If $k \neq l$ and $\{k,l\}$ is tabu, the capacity is ∞ and the cost is an upper bound

Algorithm 13 Algorithm optBal to balance solutions.

Input: a solution S, and a minimum dispersion d.
Output: a solution S' such that $B(S') \leqslant B(S)$ and $D(S') \geqslant d$. $S' = S$

procedure OPTBAL(S, d)
  for $i \in [\xi]$ **do**
    $S' \leftarrow$ successiveGroupExchanges(S', d, BB_{max}^{lo})
    $S \leftarrow \text{argmin}\{B(S), B(S')\}$
    if $B(S) = 0$ **or** $B(S') > \theta$ **then**
      break
    $S' \leftarrow$ deterministicShuffle(S', ρ)
  return S

procedure SUCCESSIVEGROUPEXCHANGES(S^0, d, BB_{max})
  $S \leftarrow S^0$; $T \leftarrow \emptyset$; $H \leftarrow \{\{k, l\} \mid k$ and l are balanced in $S^0\}$
  repeat
    $k, l \leftarrow$ selectTwoGroups(S, T, H)
    $T \leftarrow (T \cup \{\{k, l\}\}) \setminus \{$pairs that have been tabu for more than τ iterations$\}$
    $S' \leftarrow bb(S, k, l, BB_{max})$
    if $B(S') < B(S)$ **then**
      $S \leftarrow S'$
      $H \leftarrow H \setminus \{\{i, j\} \mid \{i, j\} \cap \{k, l\} \neq \emptyset, i$ or j is imbalanced in $S\}$
    if $B(S') = B(S)$ or k and l are balanced in S **then**
      $H \leftarrow H \cup \{\{k, l\}\}$
  until $B(S) = 0$ **or** $|H| = \binom{m}{2}$
  if $B(S) > 0$ and $BB_{max} \leqslant BB_{max}^{hi}$ **then**
    return successiveGroupExchanges$(S, d, 2BB_{max})$
  return S

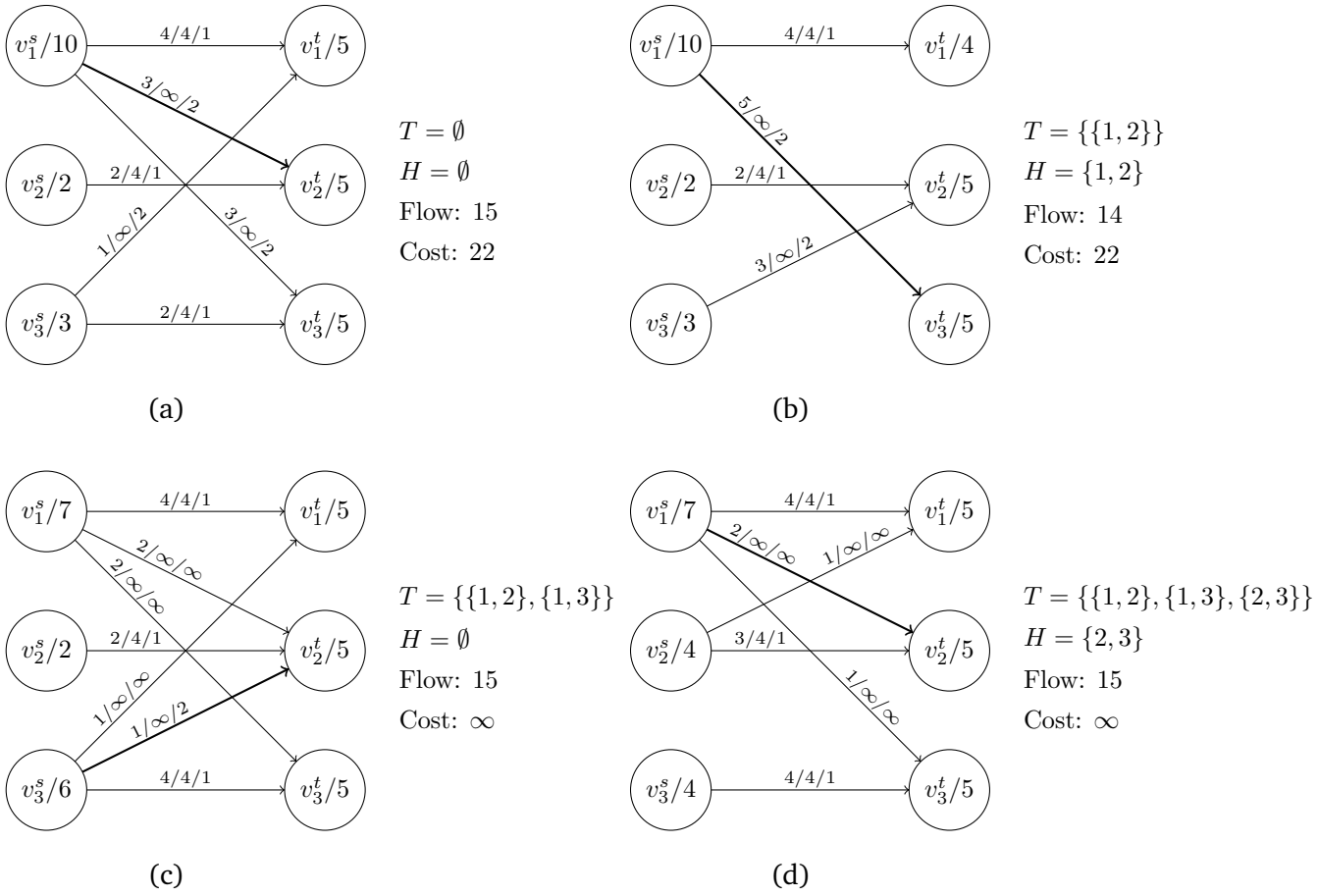


Figure 4.2: Four example iterations of the successiveGroupExchanges procedure of algorithm optBal.

$f_u = 2\sum_{i \in V} a_i$ on the maximum cost of all other edges. These arcs heavily penalize flow exchange on tabu pairs.

After finding the minimum cost flow in this graph, which can be done in time $O(f_u m^2 \log m)$ using the algorithm of Ford and Fulkerson [66], optBal selects the two non-tabu groups incident to the edge of highest flow, with ties broken lexicographically. If no flow passes through edges incident to non-tabu pairs, tabu pairs are also allowed to be selected.

Figure 4.2 illustrates four iterations of the main loop of successiveGroupExchanges on an example with $m = 3$. In this example we consider $M = (5,5,5)$ and $\alpha = 0.2$ (i.e., groups can assume weights between 4 and 6). For each iteration we show the flow graph with edge labels representing flow/capacity/cost, and node labels name/flow, with the name of the vertex and the flow passing through it. We omit edges which do not receive flow. Initially we have $w = (10,2,3)$, $T = H = \emptyset$, and the edge of highest

flow is $\{v_1^s, v_2^t\}$ with flow 3 (here, the tie with $\{v_1^s, v_3^t\}$ has been broken lexicographically), so optBal selects pair $\{1,2\}$ to exchange objects (Figure 4.2a). Suppose now that the branch-and-bound method fails to find an improving exchange between 1 and 2. In this case, $\{1,2\}$ is hopeless and optBal recomputes the minimum cost maximum flow without edge $\{v_1^s, v_2^t\}$, obtaining a highest flow 5 on edge $\{v_1^s, v_3^t\}$ (Figure 4.2b). Procedure bb now finds an improving exchange between groups 1 and 3, with new weights $w = (7,2,6)$. Pair $\{1,2\}$ is no longer hopeless, since an improving exchange incident to 1 was found, but $\{1,2\}$ and $\{1,3\}$ remain tabu. In the next iteration, edge $\{v_1^s, v_3^t\}$ has the highest flow of 2 but is tabu and therefore optBal selects the only non-tabu edge receiving flow, $\{v_3^s, v_2^t\}$ (Figure 4.2c). Executing procedure bb on groups 2 and 3 yields weights $w = (7,4,4)$. Since groups 2 and 3 are now balanced, no improving exchange between them is feasible and the pair $\{2,3\}$ is hopeless. In the fourth iteration all pairs are tabu, so optBal selects the edge with highest flow $\{v_1^s, v_2^t\}$ for the next exchange (Figure 4.2d).

4.6.2 Finding improving exchanges between two groups

The subproblem of optimally exchanging objects between two groups is: given a solution S and two groups k and l, find a sequence of shifts between them such that new the dispersion is not less than $D(S)$ and the new contribution $B_{kl}(S) = B(S_k) + B(S_l)$ of k and l to the imbalance of S is minimized. This problem is NP-complete, since the special case where $m = 2$, $D(S) = 0$, $M_1 = M_2$ and $\alpha = 0$ generalizes the Subset Sum Problem.

Given S, k and l we represent a solution by $x \in \{0,1\}^{|S_k \cup S_l|}$, where $x_i = 1$ if $i \in S_k \cup S_l$ is to be shifted to the other group, and $x_i = 0$ if i is to stay in its current group. We use a branch-and-bound algorithm to solve the problem. At each node in the search tree, the algorithm branches on the assignment of an unfixed variable x_i associated with the object of largest weight a_i, as empirically this leads to fewer expanded nodes. Since $D(S)$ is not allowed to decrease, fixing a single variable can potentially lead to a large sequence of fixations. When fixing $x_i = 1$, all conflicting x_j (i.e., $d_{ij} < D(S)$) may also be set to 1, since i and j cannot be placed in the same group. Similarly, fixing $x_i = 0$, all conflicting x_j can fixed to 0 as well. This is done recursively: when fixing a conflicting x_j, all of j's conflicts are also fixed, and so on. If we find that two conflicting objects should be fixed to the same group, the node is infeasible and can be pruned. The algorithm expands nodes depth first, giving preference to the child node of least lower bound, since this empirically leads to

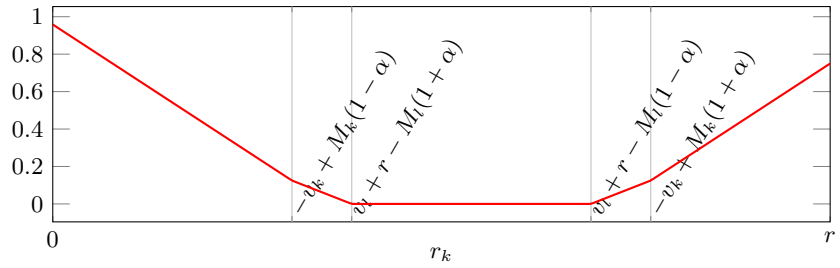


Figure 4.3: Example objective function when fractionally balancing two groups.

fewer expanded nodes. Because the decision variables represent the exchange of objects between the groups, we can obtain an upper bound at any node by setting all unfixed x_i to 0 (i.e., they remain in their current group).

For each new node a lower bound on B_{kl} is computed and the node is fathomed if it is not less than the current best value. Lower bounds are computed by ignoring conflict and integrality constraints of the unfixed objects, i.e., solving a relaxed version of the problem where any two objects can be placed together, and objects can be partially assigned. Let v_k and v_l be the total weights of the objects fixed to groups k and l, respectively, and let $r = (\sum_{i \in S_k \cup S_l} a_i) - v_k - v_l$ be the weight of the remaining objects that could still be assigned to the groups. The lower bound is then computed by finding a real-valued split $r = r_k + r_l$ of the remaining weight which minimizes B_{kl}, where r_k and r_l are the weights that will be (partially) assigned to groups k and l, respectively. This split is determined by minimizing the sum of expected imbalances

$$\max\{0, |v_k + r_k - M_k|/M_k - \alpha\} + \max\{0, |v_l + r - r_k - M_l|/M_l - \alpha\},$$

over all $r_k \in [0, r]$ (note that v_k, v_l, M_k, M_l and α are constants). This function is piecewise linear and therefore the optimum value always occurs either at the bounds of the domain or the extremes of some segment (see Figure 4.3 for an example). In other words, the optimal r_k can be found by examining, in $O(1)$, the six values 0, r, $-v_k + M_k(1 \pm \alpha)$, $v_l + r - M_l(1 \pm \alpha)$ for r_k.

4.7 Computational experiments

In this section, we report on computational experiments. We first describe in Section 4.7.1 the instances and the general experimental methodology. In Section 4.7.2 we calibrate upper bound U_h^σ and compare it to upper bounds from the literature. Next, in Sections 4.7.3 and 4.7.4 we calibrate and analyze the effectiveness of two

algorithmic components, namely solving the UMaxDP and balancing solutions, separately. We also compare our approach to solving the UMaxDP to the literature. Finally, in Sections 4.7.5 and 4.7.6 we present the main results. In Section 4.7.5 we compare our approach to the exact approach of Fernández et al. [61], and in Section 4.7.6 we present and analyze results on large test instances. We summarize our results in Section 4.7.7.

4.7.1 Test instances and methodology

We use two types of instances, "WEEE" and "study", originating from the applications discussed in Section 4.1. For each type our test set comprises instances of sizes $n = 400i$ and $m = 5 + 6i$, for $i \in [10]$ generated as proposed by Fernández et al. [61]. The number of objects n was chosen to match the range of instances solvable by our heuristic algorithm (instances smaller than $n = 400$ were almost always solved optimally, while instances larger than $n = 4000$ were intractable within an hour). For group size m we chose the largest (relative to n) of the three values used by Fernández et al. [61], since this leads to harder instances.

The WEEE instances consist of objects uniformly distributed in the plane, and distances d are Euclidean. The object weights are drawn uniformly from $[1000, 4000]$, and the target weights M_k from $[(1-\beta)\bar{A}, (1+\beta)\bar{A}]$, where $\bar{A} = \sum_{i \in V} a_i/m$ is the average group weight and $\beta \in [1,4]/4$ a slack parameter. For each combination (n, β) the test set contains 10 replicates, for a total of 400 WEEE instances.

The distances of the study instances are generated using Likert scales [134]. Each object is associated with a random vector in $[5]^{25}$, and the distance between two objects is the l_1 distance between these vectors. The target weights are chosen in the same way as for the WEEE instances, but here we use $\beta \in \{0, 0.1\}$. Again, for each combination (n, β) the test set contains 10 replicates, for a total of 200 study instances. As noted above, by definition of the distances we have $|R| \leqslant 125$ and therefore the set L of critical pairs of a solution is usually large.

Further, we use an additional set of instances, which is used to calibrate the different components of the algorithms. The calibration set consists of 10 instances for each of the 10 values of n and m used in the test set, each with the largest value of β (1 for WEEE instances, 0.1 for study instances), for a total of 200 instances. We used the largest β because these instances are more difficult to balance.

Finally, we use a baseline set of 860 instances consisting of the same number of replicates and parameters n, m and β as the ones used by Fernández et al. [61],

Table 4.1: Calibrating parameter σ of U_h^σ.

	σ/m	t	UB (r.d.)	i_{best}	t_{col}
	1.05	4.9	3.43	15.0	0.0
	1.10	4.9	3.39	13.7	0.0
	1.25	5.3	3.09	14.6	0.0
study	1.50	5.4	2.65	13.5	0.0
	2.00	388.7	**1.38**	8.4	28.7
	2.50	992.0	7.20	4.2	206.2
	3.00	1,281.7	10.69	2.8	291.9
	1.05	11.7	0.21	13.0	0.0
	1.10	11.7	0.20	34.5	0.0
	1.25	12.1	0.17	36.9	0.0
WEEE	1.50	114.4	**0.06**	29.4	5.8
	2.00	583.8	0.08	15.4	237.1
	2.50	731.4	0.13	5.9	438.0
	3.00	857.0	0.15	4.9	479.0

where n ranges from 100 to 300 for study instances and from 200 to 700 for WEEE instances. We use this data set to compare to the exact method of Fernández et al. [61], since it cannot handle larger instances in practical time.

We have executed all experiments on a PC with an 8-core AMD FX-8150 processor and 32 GB of main memory, running Ubuntu Linux 18.04. For each test, only one core was used. The heuristic algorithms were implemented in C++ and compiled with GCC 7.2 using maximum optimization. Model (F) is solved using CPLEX 12.7.1. Source code, data sets, instance generators, and detailed results are available at https://github.com/AlexGliesch/maxdp.

4.7.2 Experiment 1: upper bounds

4.7.2.1 Calibrating parameter σ for upper bound U_h^σ

In this experiment we calibrate the subinstance size σ of upper bound U_h^σ on the calibration data set. We have considered values $\sigma = \min\{3, i\,m\}$ for $i \in \{0.05, 0.1, 0.25, 0.5, 1.0, 1.5, 2.0\}$. Each run was limited to 30 minutes, and executes only the algorithm to compute U_h^σ. Table 4.1 shows the results. For each value of σ and instance type we report the running time t in seconds, the average relative deviation of the upper bound from the best known upper bound (UB (r.d.)), the average number of iterations i_{best} to find the upper bound, and the average running time t_{col} in seconds of the exact coloring algorithm.

We see that up to a certain point, larger subinstance sizes tend to yield better upper bounds. This is expected, since larger subinstances are more likely to contain intersection graphs which are not m-colorable. After $\sigma/m > 2$ for study instances and $\sigma/m > 1.5$ for WEEE instances, however, the intersection graphs become too large to be colored optimally in feasible time, and thus the quality of the upper bounds decreases, as the algorithm reaches the time limit before considering all n possible subinstances. This is supported by the coloring times t_{col}: the values of σ/m where the best upper bounds are found ($\sigma/m = 2$ for study and $\sigma/m = 1.5$ for WEEE instances) appear to also be a sweet spot for coloring, where smaller σ lead to trivial coloring subproblems and larger σ to subproblems which are too hard to solve in feasible time. We therefore set $\sigma = 1.5m$ in the following experiments, as this value performs well for both instance types.

4.7.2.2 Comparing U_h^σ, U_h^C and U_h^{RB}

In this experiment we compare the new upper bound U_h^σ to upper bounds U_h^C and U_h^{RB} from the literature on the test set. For a fair comparison, we start optimizing U_h^σ from $d^u = d^R$ instead of $d^u = U_h^C$. Running times were limited to 30 minutes.

Table 4.2 shows the results. For each instance type and size we report averages of the running time t in seconds, the relative deviation of the obtained bound from the best known upper bound (UB (r.d.)) and the optimality rate (Opt. (%)), i.e., the empirical probability of the upper bound to match the value found by our heuristic (using results of Section 4.7.3).

We find that the proposed bound U_h^σ was smallest for all instance sizes, on average, though often at the cost of being more computationally expensive. In particular, for WEEE instances of size $n \geqslant 2800$ the running times increase sharply as the sampled subinstances are larger and thus lead to harder coloring subproblems. Nonetheless, we will show in Section 4.7.3.3 that using U_h^σ even in these cases is still effective, as it leads to an overall faster execution. Regarding the other two bounds, U_h^C performed poorly for study instances, but was slightly better than U_h^{RB} for WEEE instances. As we saw in Theorem 4.3.1 upper bounds $U^C = U^1$ and $U^{RB} = U^2$ are the same, so the differences between U_h^{RB} and U_h^C are due to the sampling heuristics. We note that upper bound U_h^{RB} was originally proposed by Ríos-Mercado and Bard [179] for a variant of the MaxDP for a real-world problem in the context of WEEE waste collection. In this context, the authors report that U_h^{RB} was consistently better than U_h^C. The discrepancy between our results and those of Ríos-Mercado and Bard [179] can be explained

Table 4.2: Comparison of upper bounds U_h^C, U_h^{RB} and U_h^σ.

	n	U_h^C			U_h^{RB}			U_h^σ		
		t	UB (r.d.)	Opt.	t	UB (r.d.)	Opt.	t	UB (r.d.)	Opt.
study	400	0.0	11.74	0	0.1	2.14	0	0.1	**0.53**	0
	800	0.1	15.32	0	0.5	2.38	0	0.2	**0.00**	0
	1,200	0.2	16.97	0	1.9	0.87	0	0.5	**0.00**	0
	1,600	0.4	20.35	0	4.8	2.16	0	1.2	**0.00**	0
	2,000	0.8	20.19	0	10.1	2.05	0	2.3	**0.00**	0
	2,400	1.3	18.93	0	19.1	**0.00**	0	3.8	**0.00**	0
	2,800	2.1	19.03	0	32.7	1.17	0	6.0	**0.00**	0
	3,200	3.1	18.04	0	52.6	1.17	0	8.9	**0.00**	0
	3,600	4.5	20.72	0	80.0	1.59	0	12.7	**0.00**	0
	4,000	6.7	18.97	0	117.5	2.61	0	18.0	**0.00**	0
	Avg.	1.9	18.03	0	31.9	1.61	0	5.4	**0.05**	0
WEEE	400	0.0	**0.00**	100	0.1	2.11	60	0.1	**0.00**	100
	800	0.1	0.37	90	0.5	0.39	80	0.3	**0.00**	100
	1,200	0.4	0.03	90	1.9	3.60	60	0.8	**0.00**	100
	1,600	0.9	1.12	90	5.1	1.86	40	2.2	**0.00**	100
	2,000	1.8	0.18	80	11.6	0.25	70	4.4	**0.00**	90
	2,400	3.2	0.12	70	23.0	0.95	50	7.9	**0.00**	80
	2,800	5.3	0.41	40	41.5	2.93	0	266.0	**0.00**	68
	3,200	8.3	0.93	58	68.8	3.60	10	199.3	**0.00**	68
	3,600	12.4	0.22	30	108.4	3.35	8	55.2	**0.00**	50
	4,000	18.7	0.25	28	162.0	3.17	10	411.6	**0.00**	28
	Avg.	5.1	0.36	68	42.3	2.22	39	94.8	**0.00**	78

by the different nature of the problems investigated and of the instance generation methods.

Looking at optimality rates, we can see that for the WEEE instances the upper bounds frequently match the best heuristic value, and are useful for proving optimality. This is not the case for the study instances, where upper bounds never match the heuristic value. For this reason, in the experiments that follow we do not compute upper bounds for study instances.

4.7.3 Experiment 2: solving UMaxDP

4.7.3.1 Calibrating parameters p_1, p_2 and p_3

We used the irace R package [138] to calibrate parameters p_1, p_2 and p_3 of algorithm EX. To test a parameter setting we generate an initial solution with the greedy

constructive algorithm of Section 4.4.1 and execute algorithm optDisp' repeatedly until the current solution cannot be improved or a time limit is reached. We ran irace on the calibration data set with a budget of 4000 runs limited to 600 seconds each, over parameter ranges $p_1 \in [8], p_2 \in [16]$ and $p_3 \in [24]$. We did not use upper bounds in the calibration. The best values found were $p_1 = 4$, $p_2 = 8$ and $p_3 = 8$, and are used in further experiments.

4.7.3.2 Comparison to Fernández et al. [61]'s method for the UMaxDP

In this experiment we compare the proposed ejection chain approach to improving dispersion to the approach of Fernández et al. [61], here called FKNU, to obtain lower bounds to the unrestricted MaxDP. To obtain a lower bound with our method, we start with an initial solution obtained by the algorithm of Section 4.4.1, and repeatedly execute the EX procedure until it fails to improve D'. FKNU starts from a dispersion $d^u = U_h^C$ and iteratively decrements index u. At each iteration, it attempts to color the intersection graph G_{d^u} with the heuristic algorithm TabuCol [98]. If TabuCol reports $\chi(G_{d^u}) \leqslant m$ the algorithm returns the dispersion value d^{u+1}, otherwise it continues until $u = 1$, in which case coloring is trivial since G_{d^1} has no edges. We use the implementation of Lewis et al. [131] of TabuCol, with a fixed seed. Both algorithms were executed with a time limit of 30 minutes.

Table 4.3 shows the results. For both approaches we report the time t in seconds and the relative deviation (D (r.d.)) of the dispersion from the best known value. Since in the last iteration our algorithm will continue optimizing until the search tree is exhausted, which often consumes all available time, we also report the time to find the best value t_{best}, in seconds. This is not applicable to FKNU, since it optimizes top-down and stops as soon as a feasible solution is found.

We find that algorithm EX leads to better lower bounds, with two exceptions for study instances of size 1200 and 4000, where it is slightly worse. On the WEEE instances, in particular, lower bounds are significantly better. Finding better lower bounds takes longer, often considerably, and we also can see that EX usually consumes the available time, since it continues optimizing until the search tree is exhausted. Algorithm EX scales better on the WEEE instances, finding better upper bounds faster for $n \geqslant 2800$ objects. For such instances FKNU does not scale well with n since the number of different distances $|R|$, and thus the average number of iterations required, grows quadratically.

Table 4.3: Comparison of our ejection chain-based algorithm to the approach of Fernández et al. [61] for the UMaxDP.

	n	EX			FKNU	
		t	t_{best}	D (r.d.)	t	D (r.d.)
study	400	99.5	0.4	**0.00**	0.4	1.84
	800	1,727.9	2.6	**0.00**	0.7	**0.00**
	1,200	1,725.4	13.4	0.26	1.4	**0.00**
	1,600	1,800.0	31.0	**0.00**	2.8	**0.00**
	2,000	1,800.0	64.2	**0.00**	4.7	**0.00**
	2,400	1,800.0	32.0	**0.00**	7.5	**0.00**
	2,800	1,800.0	102.9	**0.00**	10.9	**0.00**
	3,200	1,800.0	48.6	**0.00**	15.4	**0.00**
	3,600	1,800.0	81.5	**0.00**	20.9	**0.00**
	4,000	1,800.0	92.5	0.98	27.2	**0.00**
	Avg.	1615.3	46.9	**0.12**	9.2	0.18
WEEE	400	17.0	0.1	**0.00**	0.0	**0.00**
	800	1,384.9	0.4	**0.00**	1.1	0.10
	1,200	1,676.2	1.6	**0.00**	1.2	**0.00**
	1,600	1,792.1	11.2	**0.00**	79.3	0.95
	2,000	1,800.0	68.8	**0.00**	150.7	1.08
	2,400	1,800.0	189.1	**0.00**	588.4	2.24
	2,800	1,800.0	340.1	**0.00**	1244.0	45.30
	3,200	1,800.0	347.1	**0.00**	1417.8	42.11
	3,600	1,800.0	768.1	**0.00**	1547.5	69.60
	4,000	1,800.0	1138.8	**0.00**	1800.0	86.89
	Avg.	1,567.0	286.5	**0.00**	683.0	24.83

4.7.3.3 Effectiveness of upper bounds for the WEEE instances

In this experiment we evaluate the effectiveness of upper bound U_h^σ in reducing total running time by allowing optimization to stop as soon as the upper bound is reached, and thus skip the last iteration of EX. As reported in Section 4.7.2.2, upper bounds were not effective for study instances, so here we consider WEEE instances only. We use the results of the experiments of Sections 4.7.2.2 and 4.7.3.2. Table 4.4 shows, for each value of n, averages of the optimality rate Opt. (%) of U_h^σ, the time t (EX) needed to compute EX, the time t (EX, last iter.) of the last iteration of EX, the time t (U_h^σ) to compute U_h^σ, and the time t (EX+U_h^σ) which is the average time needed to compute U_h^σ plus the time to best, for instances which were solved optimally, or the total time, for instances which were not.

We see that the last iteration is the most time-consuming part of the EX method.

Table 4.4: Effectiveness of upper bounds for algorithm optDisp, for WEEE instances

n	Opt. (%)	t (EX)	t (EX, last iter.)	t (U_h^σ)	t (EX+U_h^σ)
400	100	17.0	8.9	0.1	8.2
800	100	1,384.9	745.7	0.3	639.5
1,200	100	1,676.2	1,406.3	0.8	270.7
1,600	100	1,792.1	1,714.2	2.2	80.2
2,000	90	1,800.0	1,731.2	4.4	249.5
2,400	80	1,800.0	1,610.9	7.9	443.9
2,800	68	1,800.0	1,459.9	266.0	992.3
3,200	68	1,800.0	1,452.9	199.3	897.4
3,600	50	1,800.0	1,031.9	55.2	1,171.1
4,000	28	1,800.0	661.3	411.6	1,853.9
Avg.	78	1,567.0	1,182.3	94.8	660.7

The reason is that when no improvement is possible, EX must exhaust the search tree before halting. Consequently, the last iteration consumes in average more than 75% of the running time. We see in column t (U_h^σ) that the average time to compute the upper bound is usually smaller than the time for the last iteration of EX. Therefore, when the optimal solution is found and matches the upper bound, the overall running time decreases. This is confirmed by the results in column t (EX+U_h^σ), where running times are on average about 50% smaller than t (EX). As n grows, however, particularly after $n \geqslant 4000$, optimality rates decrease and upper bounds becomes less advantageous.

4.7.4 Experiment 3: balancing solutions

4.7.4.1 Calibrating parameters ξ, θ and ρ

Parameters ξ, θ and ρ of optBal have been calibrated by irace, using the calibration data set, with a budget of 4000 runs and a time limit of 30 minutes per run. Each run executes the full algorithm with $\alpha = 0.001$ and reports the dispersion value, or $-\infty$ if no feasible solution was found. We used parameter ranges $\xi \in 5 \times [10], \theta \in \{0.1, 0.25, 0.5, 1, 2, 5\}$ and $\rho \in [0.01, 0.2]$. The best values found were $\xi = 35$, $\theta = 0.5$ and $\rho = 0.01$, which are used in the remaining experiments.

4.7.4.2 Comparison to a variable neighborhood search-based approach

To evaluate the effectiveness of our balancing method, we implemented a VNS approach as a baseline. The VNS iteratively searches for an improving operation

in the neighborhoods $N^*_{shift} = N_{shift} \cap \{i \to k \mid w_{S(i)} > M_{S(i)}$ **and** $w(S_k) < M_k\}$ (i.e., we only allow shifts where a group exceeding its target weight sends an object to a group below its target weight), and N_{swap}. We use N^*_{shift} since we found it to be more effective than of N_{shift} in preliminary tests. The search explores the candidates of each neighborhood in a round-robin fashion and applies improving candidates as soon as they are encountered, proceeding then to the next iteration. If no improving candidates are found, the current best solution is "shaken" and the search continues from there. Shaking is done by performing up to $n\rho_{vns}$ random shifts which do not lower the current dispersion. The search stops after a maximum ξ_{vns} shakes. We used irace to calibrate parameters ρ_{vns} and ξ_{vns} as in Section 4.7.4.1, and found the best configuration $\rho_{vns} = 0.03$ and $\xi_{vns} = 50$.

For the comparison, we executed both algorithms on the test set with $\alpha \in \{5, 1, 0.1\} \times 10^{-2}$ for WEEE instances and $\alpha = 10^{-3}$ for study instances. We use these values of α to be consistent with Fernández et al. [61]. Table 4.5 shows the results. Since the VNS is stochastic, each test was replicated 7 times and we report averages. For both approaches we report the time t in seconds of the balancing procedure, the relative deviation (D (r.d.)) of the dispersion from the best known value, and the optimality rate (Opt. (%)), which accounts the cases where upper bounds were met and cases where the instance was proven infeasible by model (F). For each instance size n we report averages over all values of α and β.

The results show that Hase produces better results than VNS, in less time. On study instances both algorithms find solutions of almost the same quality in the same time, but on WEEE instances Hase finds significantly better solutions and is about 40 % faster, with a single exception at $n = 400$ where the relative deviation of the dispersion is slightly higher. In all study instances the execution stopped because EX timed out, and every lower bound on UMaxDP found by EX was made feasible by both approaches. This suggests that study instances are easy to balance, but that improving their dispersion is hard. The running times also corroborate this. Since $|R|$ is small there are typically several critical pairs, and thus an increasingly larger number of moves are necessary for each improvement to D. The better dispersion values of Hase on WEEE instances can be explained by its higher success rate in balancing solutions, so it can continue optimizing the dispersion. For both instance types, the number of feasible solutions was the same, i.e., either both approaches found a feasible solution, or none did. In 39% of executions our approach obtained the best solution, while the VNS approach was better in 10% of cases.

Table 4.5: Comparison of our algorithm for balancing solutions to a VNS approach.

	n	Hase			VNS		
		t	D (r.d.)	Opt. (%)	t	D (r.d.)	Opt. (%)
study	400	0.3	**0.28**	0.0	3.9	0.42	0.0
	800	0.5	**0.00**	0.0	0.5	**0.00**	0.0
	1,200	1.3	**0.00**	0.0	1.9	**0.00**	0.0
	1,600	3.0	**0.00**	0.0	3.7	0.06	0.0
	2,000	5.8	**0.00**	0.0	7.6	**0.00**	0.0
	2,400	9.8	**0.00**	0.0	12.2	0.03	0.0
	2,800	15.8	**0.00**	0.0	18.3	**0.00**	0.0
	3,200	24.7	**0.00**	0.0	25.4	**0.00**	0.0
	3,600	35.1	**0.00**	0.0	37.8	0.05	0.0
	4,000	47.7	**0.00**	0.0	45.4	**0.00**	0.0
	Avg.	14.4	**0.03**	0.0	15.7	0.06	0.0
WEEE	400	0.4	0.71	97.5	0.7	**0.65**	89.6
	800	36.4	**0.53**	90.8	24.7	1.50	77.4
	1,200	60.6	**0.27**	85.8	46.5	1.43	79.6
	1,600	144.4	**0.94**	88.9	301.1	2.59	60.0
	2,000	224.2	**1.10**	56.7	535.5	3.55	47.4
	2,400	271.3	**1.03**	44.2	687.9	5.75	38.9
	2,800	471.2	**0.41**	78.4	771.8	8.21	33.6
	3,200	618.7	**0.67**	56.2	920.7	9.69	27.1
	3,600	828.8	**0.56**	23.3	1,050.9	13.53	17.5
	4,000	779.8	**0.65**	56.3	839.6	10.75	15.6
	Avg.	343.6	**0.69**	67.8	517.9	5.77	48.7

4.7.5 Experiment 4: comparison to the exact approach of Fernández et al. [61]

In this experiment we compare our hybrid heuristic to the exact algorithm of Fernández et al. [61], called FKN here, on the baseline data set, with a time limit of one hour. We compare only to FKN because it dominates the heuristic of Moeini et al. [150]. We implemented FKN using Julia 0.6.1 and CPLEX 12.7.1. The upper and lower bounds of the algorithm are computed in C++ and given as parameters. Since solving the MIP models are the bottleneck of FKN, the performance loss of using Julia as opposed to C++ is negligible. Tables 4.6 and 4.7 show the results for WEEE and study instances, respectively. We report averages for each set of instances with the same number of objects n and slack β, since the number of groups m and slack α have little influence on difficulty. The number of instances of each configuration is given in column #. For each algorithm we report the running time t in seconds, and the number of optimal solutions found (Opt.). For the hybrid heuristic, column (Opt.)

Table 4.6: Comparison of our heuristic algorithm to the exact algorithm of Fernández et al. [61], for instances of type WEEE.

n	β	#	Hase				FKN		
			t	Opt.	Prv.	D (r.d.)	t	t (Opt.)	Opt.
200	0.25	90	0.1	88	88	0.04	11.3	11.3	**90**
	0.5	90	0.1	**90**	90	0.00	11.7	11.7	**90**
	0.75	90	0.5	88	88	0.01	17.4	17.4	**90**
	1	90	35.9	84	83	0.14	134.3	95.4	**89**
300	0.25	90	0.1	89	84	0.01	16.9	16.9	**90**
	0.5	90	0.1	88	83	0.03	15.8	15.8	**90**
	0.75	90	0.1	**90**	84	0.00	28.8	28.8	**90**
	1	90	1.1	84	76	0.51	141.3	62.7	**88**
400	0.25	90	0.2	88	88	0.29	43.5	43.5	**90**
	0.5	90	0.1	**90**	90	0.00	137.6	98.7	89
	0.75	90	0.2	**89**	89	0.04	205.1	167.0	**89**
	1	90	44.1	71	70	1.22	621.2	163.0	**78**
500	0.25	90	0.3	81	79	0.19	213.4	136.5	**88**
	0.5	90	0.3	87	84	0.00	265.7	228.2	**89**
	0.75	90	0.3	**86**	84	0.00	367.2	94.6	83
	1	90	183.7	59	59	0.82	1,234.1	220.2	**63**
600	0.25	90	0.3	82	82	0.31	421.4	194.4	**84**
	0.5	90	0.4	**88**	88	0.09	510.1	208.7	82
	0.75	90	0.5	**89**	89	0.03	533.2	274.6	83
	1	90	48.7	**76**	76	0.38	1,136.8	339.9	68
700	0.25	90	0.6	**85**	85	0.17	806.4	376.6	78
	0.5	90	0.6	**85**	85	0.05	881.1	422.0	77
	0.75	90	17.9	**85**	85	0.11	1,037.8	305.8	70
	1	90	56.6	**70**	70	0.91	1,805.6	553.0	53
	Total		16.4	**2,012**	1,979	0.22	441.6	170.3	1,981

considers both the case where optimality was proven by matching the upper bound, and the case where the optimal solution was found but not proven. We also report for Hase the number of solutions with optimality proofs (Prv.), and the average relative deviation (D (r.d.)) in percent from the optimal solution, considering only cases where FKN found optimality. For FKN we report the average running time (t (Opt.)), considering only instances which were solved optimally. In all cases, FKN either found the optimal solution, proved infeasibility or reached the time limit without obtaining any feasible solution.

For study instances, we see that FKN finds optimal solutions for all instances of $n = 100$ and around half the instances of $n = 200$ and $n = 300$. Using upper bounds

Table 4.7: Comparison of our heuristic algorithm to the exact algorithm of [61], for instances of type study.

n	β	#	Hase				FKN		
			t	Opt.	Prv.	D (r.d.)	t	t (Opt.)	Opt.
100	0	20	1.2	18	14	0.31	21.1	21.1	**20**
	0.1	20	0.4	17	13	0.46	135.3	135.3	**20**
200	0	30	5.5	**16**	8	0.00	1,826.3	274.3	**16**
	0.1	30	11.1	**15**	9	0.00	2,063.0	53.2	13
300	0	20	38.0	**9**	5	0.38	2,310.9	735.3	**9**
	0.1	20	24.1	**8**	5	0.43	2,470.5	776.3	**8**
Total			13.4	83	54	0.26	1,471.2	332.6	**86**

the hybrid heuristic was able to find provably optimal solutions on 54 tests, while in the remaining 29 it found the optimal solution but was not able to prove it. For $n = 200$ and $\beta = 0.1$ the hybrid heuristic found proven optimal solutions to 2 additional instances for which FKN did not. This suggests that, though results from Section 4.7.2.2 indicated upper bounds were not effective on study instances of the test set, they are sometimes useful on smaller instances. This is due to the baseline data set also having instances with fewer groups m, while the test set considers only the largest value of m used by Fernández et al. [61]. Overall, Hase found slightly fewer optimal solutions, 83 compared to 86 of FKN. On the instances which FKN solved optimally, the hybrid heuristic produces solutions which deviate, on average, 0.26% from the optimal values.

For WEEE instances the hybrid heuristic found a more optimal solutions than FKN (2012 compared to 1981), and most these, namely 1979, were proven optimal by the heuristic. Again, this confirms that upper bounds are much tighter on WEEE instances. After $n = 600$, Hase consistently finds more optimal solutions than FKN, as instances become too large for the exact algorithm to solve in one hour. Considering only instances where FKN found the optimal solution, our method produced solutions within 0.22% of optimality, on average. We can also notice that, for WEEE instances, there is a slight increase in difficulty for $\beta = 0.75$ and a sharp increase for $\beta = 1$, for both approaches. For study instances, as expected $\beta = 0.1$ was slightly harder than $\beta = 0$.

Concerning running times, the hybrid heuristic was significantly faster in all cases, as expected, with running times 24.8 times smaller for study instances and 10.4 times smaller for WEEE instances, on average.

Table 4.8: Results of “Hase” for large instances.

	n	D (r.d. ub)	Opt. (%)	Fea.	t	Iter.
study	400	12.85	0.0	100.0	687.5	15.2
	800	19.11	0.0	100.0	3,339.2	4.2
	1,200	21.78	0.0	100.0	3,600.0	3.3
	1,600	22.42	0.0	100.0	3,600.0	3.4
	2,000	22.22	0.0	100.0	3,600.0	3.6
	2,400	24.01	0.0	100.0	3,600.0	3.5
	2,800	24.41	0.0	100.0	3,600.0	3.4
	3,200	25.68	0.0	100.0	3,600.0	3.5
	3,600	24.83	0.0	100.0	3,600.0	3.6
	4,000	25.67	0.0	100.0	3,600.0	3.5
	Avg.	22.30	0.0	100.0	3,282.7	4.7
WEEE	400	13.52	90.8	100.0	0.5	2.7
	800	13.66	84.2	100.0	48.2	9.1
	1,200	6.41	85.8	100.0	65.5	11.8
	1,600	9.62	69.2	100.0	194.6	29.2
	2,000	9.42	56.7	100.0	470.9	46.0
	2,400	9.97	44.2	100.0	677.3	60.0
	2,800	8.57	37.5	90.0	1,422.5	76.4
	3,200	8.91	32.5	100.0	1,468.3	102.3
	3,600	9.50	28.3	100.0	1,970.1	114.4
	4,000	11.01	19.2	80.0	2,231.5	119.2
	Avg.	10.06	54.8	97.0	854.9	57.1

4.7.6 Experiment 5: final results on test instances

In this section we report full results for the hybrid heuristic on the test set. We have run Hase with a time limit of one hour, and report in Table 4.8 the relative deviation of the dispersion from the best upper bound (D (r.d. ub)), the optimality rate (Opt. (%)), the rate of feasibility (Fea. (%)) over instances which were not proven infeasible by model (F), the total running time t in seconds, and the number of iterations (Iter.). For each instance type and number of objects n we report averages over all values of α and β. As can been seen in Tables 4.6 and 4.7, the running times of the exact approach of Fernández et al. [61] increase sharply with the number of objects n, and the optimality rates decrease accordingly. We therefore do not report results for FKN on the test set.

We observe that dispersion values for study instances deviate about 25% from the upper bound, corroborating previous findings that the upper bounds are never effective for these instances. As a consequence no instance was solved to proven optimal-

ity. For WEEE instances, Hase usually either finds the optimal solution, or terminates with a relatively large relative deviation from the upper bound (about 10%). This is likely due to the heuristic reaching a difficulty barrier with respect to balancing, as upper bounds only consider the subproblem without balance constraints. As expected, optimality rates decrease with increasing n.

Looking at the running times, we see that the hybrid heuristic tends to terminate well before the time limit on WEEE instances, since upper bounds are tighter and often match the dispersion values found. On study instances, on the other hand, due to upper bounds not being as tight the hybrid heuristic times out for $n \geqslant 1200$, since without optimality detection algorithm EX must be executed until completion.

Concerning feasibility rates, the hybrid heuristic either found a feasible solution or proved infeasibility in 97% of the cases, except for a few difficult instances of size $n = 2800$ and $n = 4000$. Out of the 1400 configurations (instance, α), 20 were proven infeasible by model (F).

4.7.7 Summary of results

In summary, we have shown that the proposed upper bound U_h^σ is consistently better than bounds U_h^{RB} and U_h^C from the literature, and matched lower bounds found by our heuristic more often. For WEEE instances, U_h^σ was effective in allowing Algorithm 12 to skip its final iteration whenever it reached optimality, reducing running time by 58%. For study instances, no upper bound matched the lower bounds found heuristically. In an analysis of individual components we have found that the proposed ejection chain method 12 finds significantly better dispersion values than algorithms from the literature, which were optimal in 78% of cases, and that the proposed truncated branch-and-bound strategy is significantly better than a baseline VNS approach. An effective balancing strategy allows heuristic Hase to execute more iterations, and thus improve the dispersion further. Compared to the exact algorithm of Fernández et al. [61] on smaller instances, Hase is up to two orders of magnitude faster and finds more optimal solutions. In cases where Hase did not find optimal values, their values deviate less than 0.26% from optimality, on average. Hase is also able to solve instances with up to $n = 4000$ objects.

4.8 Conclusion

We have proposed a new, hybrid heuristic and an improved upper bound for the Maximum Dispersion Problem, and have shown experimentally that good solutions

can be found one to two orders of magnitude faster than with previous methods. These solutions are often provably optimal since they match the upper bounds.

Our strategy alternates between balancing the groups and increasing the maximum dispersion. We have applied a similar strategy in our heuristic of Chapter 3, and believe that such an approach may be generally useful for solving location problems with conflicting objectives of dispersion and balancing. The variable-depth ejection chain strategy we proposed to improve dispersion may also be useful in other location problems with similar max-min, or min-max objective functions with a polynomially large set of objective values which can lead to cascading conflicts, such as the diameter or p-center. Finally, our heuristic uses truncated and complete exact algorithms for balancing, a strategy we believe is of general interest, and we show that they lead to better results than simpler strategies. We believe that similar methods can be useful in related grouping problems with balancing or capacity constraints.

We have further shown that two previously proposed upper bounds are in fact the same, and that the new upper bound is considerably tighter and enables the heuristic to often prove optimality. Our new upper bound is based on finding k-vertex-critical subgraphs for a graph coloring subproblem, and in a separate work we have further extended this idea towards a general heuristic for finding k-vertex-critical subgraphs [79].

5 CONCLUSION

Districting problems have an extensive variety of applications, ranging from the design of electoral districts to the allocation of farmland into lots. In this thesis we have looked at districting from an application-independent point of view, as a pure mathematical problem. This approach has seldom been taken in the literature, as authors usually develop tailored solutions to a specific application.

In the first chapter of this thesis, we delved into various optimization criteria that are common across the districting applications. We have framed these within a generic single-objective model that considers the three most important districting criteria: compactness, balance and connectivity, as well as potentially additional, domain-specific criteria. As particular cases we have defined the p-Median, p-Center and Diameter Districting Problems, which, although they appear as subsets of many formulations, have previously received relatively little attention as standalone problems. In this first part, we also had a deeper look into the most common MIP formulations and metaheuristic approaches used to solve districting problems.

In the second chapter, we have presented a hybrid heuristic that can efficiently solve variants defined by this generic model. These include the p-Median, p-Center and Diameter Districting Problems, which we addressed specifically in our implementation and experiments. We showed that our heuristic was effective in handling all three variants, and compared favorably against existing methods that were targeted at only one. In separate studies we considered the extension of the p-Median Districting Problem with two common domain-specific criteria: routing costs and similarity to previous plans. We showed that our method can effectively handle these criteria with little change or loss of performance.

In the third chapter, we investigated the Maximum Dispersion Problem, a location problem closely related to districting that asks for dispersed rather than compact groups. We proposed a hybrid method to solve it, that follows the same structure as our general method for districting and combines several heuristic components which we show to be independently effective. We also proposed an improved upper bound

for the problem. Our heuristic is able to find solutions significantly faster than other methods, and in most cases proves optimality by matching values to upper bounds.

Algorithmically, the methods we proposed alternate between improving compactness (or dispersion) and a binary search strategy that solves a series of constrained balancing subproblems. Over a long development period we have implemented and tested several heuristics, and ultimately settled on this alternating approach which we believe is especially well-suited for location problems with conflicting optimization criteria, such as compactness and balance. A key advantage is its independence on the optimization algorithm used for each criterion. For example, in our general heuristic for districting we used tabu search to improve both compactness and balance, whereas for the Maximum Dispersion Problem we used an ejection chain algorithm to improve dispersion and a hybrid method based on truncated branch-and-bounds to improve balance.

Another major component of our methods was their heavy use of dynamic algorithms to cache and efficiently update optimization criteria, such as balance, compactness or connectivity, under neighboring operators. Since these updates are clear bottlenecks during neighborhood search, this led to a significant speedup and allowed greater flexibility in our heuristic design. These dynamic algorithms are of general interest and could be used in other methods, making them an independent contribution.

At the end of each chapter we have laid out our suggested directions for future research. Overall, our work in this thesis represents a step towards the study of districting from an application-independent perspective. We believe this is a worthwhile pursuit, as it fosters contributions to the area from researchers who are not in direct contact with a real-world application, and facilitates the reuse of methods across districting domains, or, more generally, location problems.

RESUMO EXPANDIDO EM PORTUGUÊS

Esta tese de doutorado se concentra na análise e resolução de problemas de setorização ou distritamento. Esses problemas envolvem a divisão de um grafo planar conectado G em p grupos chamados distritos ou territórios. O grafo G é composto por n vértices, que são geralmente referidos como unidades básicas ou unidades geográficas. Estas unidades podem representar entidades geográficas como quarteirões de cidades ou bairros, que devem ser agrupadas em distritos.

A setorização é utilizada em diversas aplicações do mundo real, desde a designação de áreas urbanas para prestadores de serviços (como policiamento, coleta de resíduos, distribuição de sal, assistência domiciliar e entregas de produtos), até a delimitação de distritos eleitorais ou políticos e a criação de lotes de fazendas com igual produtividade em áreas rurais.

Vários critérios definem o que torna um plano de setorização aceitável ou viável e, frequentemente, esses critérios variam de acordo com a aplicação. No entanto, os critérios de balanceamento, conectividade e compacidade são quase sempre presentes. O critério de balanceamento exige que os distritos sejam balanceados em relação a um conjunto de atributos associados às unidades, que podem representar diferentes aspectos das unidades, dependendo da aplicação. Em distritos de serviço, é comum usar atributos relacionados ao custo ou tempo de atendimento de uma unidade, enquanto em distritos eleitorais é desejado que os distritos tenham um número igual de eleitores. O critério de conectividade exige que cada distrito seja um subgrafo conectado de G, para facilitar o trânsito ao longo das unidades de um distrito. Por fim, o critério de compacidade exige que os distritos formem formas geometricamente compactas, que facilitam a travessia por agentes de serviço e podem fornecer pistas visuais sobre a manipulação de distritos para obter vantagem partidária, uma tática conhecida como *Gerrymandering*.

Além desses critérios, existem também critérios específicos do domínio. Alguns exemplos incluem a minimização das diferenças em relação aos planos de setorização anteriores, a proibição de enclaves (distritos dentro de distritos), a satisfação de relações de conflito entre unidades básicas e a minimização da soma dos custos de roteamento intra ou interdistrital. A combinação desses requisitos torna a setorização particularmente desafiadora, pois a maioria dos requisitos se traduz em subproblemas NP-difíceis cuja interação cria problemas de otimização difíceis.

Devido à dificuldade do districting, o foco das soluções computacionais até agora

tem sido em heurísticas. Algumas soluções exatas baseadas em branch-and-bound sobre modelos de programação inteira (IP) também foram propostas, mas com as capacidades computacionais atuais, elas se limitam a instâncias de 500 a 700 unidades e 2-10 distritos, enquanto as instâncias do mundo real têm milhares de unidades e dezenas de distritos.

Esta tese faz três contribuições principais.

A primeira contribuição é a proposta de uma heurística genérica para problemas de distritamento que pode lidar com uma grande combinação de critérios diferentes encontrados na literatura. A metodologia proposta alterna entre duas buscas de vizinhança, uma que otimiza o objetivo e outra que minimiza as violações de restrição. Este método tem a vantagem de ser um algoritmo multistart aleatório que começa a partir de várias soluções iniciais e tenta melhorar cada uma delas independentemente.

A segunda contribuição é a proposição de uma nova heurística para o Maximum Dispersion Problem, que busca particionar um conjunto de unidades ponderadas em grupos de tal forma que a dispersão, definida como a distância máxima entre dois objetos no mesmo grupo, seja máxima. Esta heurística é de particular relevância porque pode ser usada como um bloco de construção em vários outros problemas, incluindo problemas de distritamento.

A terceira contribuição é a criação de um benchmark público para problemas de distrito, que é atualmente uma das principais barreiras para a avaliação comparativa dos algoritmos de distrito. A existência de um benchmark público permitirá a comparação justa dos algoritmos e acelerará o progresso na resolução de problemas de distritamento.

REFERENCES

[1] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3): 75–102, 2002.

[2] M. Albareda-Sambola and J. Rodríguez-Pereira. Location-routing and location-arc routing. In G. Laporte, S. Nickel, and F. Saldanha da Gama, editors, *Location Science*, pages 431–451. Springer International Publishing, Cham, Switzerland, 2019.

[3] K. Andreev and H. Racke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

[4] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979.

[5] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Concorde TSP solver, 2006. URL http://www.tsp.gatech.edu/concorde.

[6] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, United States, 2006.

[7] T. Arani and V. Lotfi. A three phased approach to final exam scheduling. *IIE Transactions*, 21(1):86–96, 1989.

[8] V. Arredondo, M. Martínez-Panero, T. Peña, and F. Ricca. Mathematical political districting taking care of minority groups. *Annals of Operations Research*, 305(1):375–402, 2021.

[9] E. A. Autry, D. Carter, G. Herschlag, Z. Hunter, and J. C. Mattingly. Multi-scale merge-split Markov chain Monte Carlo for redistricting. *Preprint arXiv:2008.08054*, 2020.

[10] K. R. Baker and S. G. Powell. Methods for assigning students to groups: a study of alternative objective functions. *Journal of the Operational Research Society*, 53(4):397–404, 2002.

[11] R. Barnes and J. Solomon. Gerrymandering and compactness: Implementation, flexibility and abuse. *Preprint arXiv:1803.02857*, 2018.

[12] F. Bação, V. Lobo, and M. Painho. Applying genetic algorithms to zone design. *Soft Computing*, 9(5):341–348, 2005.

[13] J. Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55(4):299–327, 1959.

[14] R. Becker, I. Lari, M. Lucertini, and B. Simeone. Max-min partitioning of grid graphs into connected components. *Networks*, 32(2):115–125, 1998.

[15] G. Benade, N. Ho-Nguyen, and J. Hooker. Political districting without geography. *Operations Research Perspectives*, 9:100227, 2022.

[16] E. Benzarti, E. Sahin, and Y. Dallery. Operations management applied to home care services: Analysis of the districting problem. *Decision Support Systems*, 55 (2):587–598, 2013.

[17] P. K. Bergey, C. T. Ragsdale, and M. Hoskote. A simulated annealing genetic algorithm for the electrical power districting problem. *Annals of Operations Research*, 121(1):33–55, 2003.

[18] P. Bertolazzi, L. Bianco, and S. Ricciardelli. A method for determining the optimal districting in urban emergency services. *Computers & Operations Research*, 4(1):1–12, 1977.

[19] M. Blais, S. D. Lapierre, and G. Laporte. Solving a home-care districting problem in an urban setting. *Journal of the Operational Research Society*, 54(11): 1141–1147, 2003.

[20] L. Bodin and L. Levy. The arc partitioning problem. *European Journal of Operational Research*, 53(3):393–401, 1991.

[21] S. Borgwardt, A. Brieden, and P. Gritzmann. Geometric clustering for the consolidation of farmland and woodland. *The Mathematical Intelligencer*, 36 (2):37–44, 2014.

[22] B. Bozkaya, E. Erkut, and G. Laporte. A tabu search heuristic and adaptive memory procedure for political districting. *European Journal of Operational Research*, 144(1):12–26, 2003.

[23] B. Bozkaya, E. Erkut, D. Haight, and G. Laporte. Designing new electoral districts for the city of Edmonton. *Interfaces*, 41(6):534–547, 2011.

[24] A. Brieden, P. Gritzmann, and F. Klemm. Constrained clustering via diagrams: A unified theory and its application to electoral district design. *European Journal of Operational Research*, 263(1):18–34, 2017.

[25] J. Brimberg, N. Mladenović, and D. Urošević. Solving the maximally diverse grouping problem by skewed general variable neighborhood search. *Information Sciences*, 295:650–675, 2015.

[26] Bureau of the Census. Census tracts for the 2020 census. https://www.census.gov/data.html, 2018. Retrieved in 12-12-2022.

[27] A. Butsch. *Districting Problems - New Geometrically Motivated Approaches*. PhD thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, 2016.

[28] A. Butsch, J. Kalcsics, and G. Laporte. Districting for arc routing. *INFORMS Journal on Computing*, 26(4):809–824, 2014.

[29] S. I. Caballero-Hernández, R. Z. Ríos-Mercado, F. López, and S. E. Schaeffer. Empirical evaluation of a metaheuristic for commercial territory design with joint assignment constraints. In J. E. Fernandez, S. Noriega, A. Mital, S. E. Butt, and T. K. Fredericks, editors, *Proceedings of the 12th Annual International Conference on Industrial Engineering Theory, Applications, and Practice (IJIE)*, pages 422–427, Cancun, Mexico, November 2007. ISBN: 978-0-9654506-3-8.

[30] M. Camacho-Collados, F. Liberatore, and J. M. Angulo. A multi-criteria police districting problem for the efficient and effective design of patrol sector. *European Journal of Operational Research*, 246(2):674–684, 2015.

[31] D. Carter, G. Herschlag, Z. Hunter, and J. Mattingly. A merge-split proposal for reversible Monte Carlo Markov chain sampling of redistricting plans. *Preprint arXiv:1911.01503*, 2019.

[32] 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.

[33] C. Chou, S. Kimbrough, J. Sullivan-Fedock, C. J. Woodard, and F. H. Murphy. Using interactive evolutionary computation (IEC) with validated surrogate fitness functions for redistricting. In T. Soule, editor, *Proceedings of the 14th Annual Conference on Genetic and Evolutionary computation (GECCO'12)*, pages 1071–1078, Philadelphia, United States, July 2012.

[34] V. Cohen-Addad, P. N. Klein, and N. E. Young. Balanced centroidal power diagrams for redistricting. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'18)*, pages 389–396, Seattle, United States, November 2018.

[35] L. Cooper. Heuristic methods for location-allocation problems. *SIAM Review*, 6 (1):37–53, 1964.

[36] Á. Corberán and G. Laporte. *Arc Routing: Problems, Methods, and Applications*. SIAM, Philadelphia, United States, 2015.

[37] O. Coudert. Exact coloring of real-life graphs is easy. In *Proceedings of the 34th Annual Design Automation Conference*, pages 121–126, Anaheim, United States, June 1997. ACM Press.

[38] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[39] S. J. D'Amico, S.-J. Wang, R. Batta, and C. M. Rump. A simulated annealing approach to police district design. *Computers & Operations Research*, 29(6): 667–684, 2002.

[40] D. Datta, J. Malczewski, and J. R. Figueira. Spatial aggregation and compactness of census areas with a multiobjective genetic algorithm: A case study in Canada. *Environment and Planning B: Planning and Design*, 39(2):376–392, 2012.

[41] D. Datta, J. Figueira, A. Gourtani, and A. Morton. Optimal administrative geographies: An algorithmic approach. *Socio-Economic Planning Sciences*, 47 (3):247–257, 2013.

[42] L. S. De Assis, P. M. Franca, and F. L. Usberti. A redistricting problem applied to meter reading in power distribution networks. *Computers & Operations Research*, 41:65–75, 2014.

[43] A. M. P. de Mendonça. *Distritamento Aplicado ao Problema de Faturamento em Redes de Serviço*. PhD thesis, Centro Federal de Educaçao Tecnológica Celso Suckow da Fonseca, Rio de Janeiro, Brazil, 2020.

[44] R. F. Deckro. Multiple objective districting: A general heuristic approach using multiple criteria. *Journal of the Operational Research Society*, 28(4):953–961, 1977.

[45] D. DeFord, M. Duchin, and J. Solomon. Recombination: A family of Markov chains for redistricting. *Harvard Data Science Review*, 3:1–58, 2021.

[46] D. Demetriou, L. See, and J. Stillwell. A spatial genetic algorithm for automating land partitioning. *International Journal of Geographical Information Science*, 27(12):2391–2409, 2013.

[47] D. Demetriou, L. See, and J. Stillwell. A parcel shape index for use in land consolidation planning. *Transactions in GIS*, 17(6):861–882, 2013.

[48] G. A. Dirac. Some theorems on abstract graphs. *Proceedings of the London Mathematical Society*, s3-2(1):69–81, 1952.

[49] R. G. Dixon. Democratic representation: Reapportionment in law and politics. *American Political Science Review*, 63(2):567–568, 1969.

[50] A. Drexl and K. Haase. Fast approximation methods for sales force deployment. *Management Science*, 45(10):1307–1323, 1999.

[51] A. Duarte, J. Sánchez-Oro, M. G. C. Resende, F. Glover, and R. Martí. Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. *Information Science*, 296:46–60, 2015.

[52] M. Duchin and B. E. Tenner. Discrete geometry for electoral geography. *Preprint arXiv:1808.05860*, 2018.

[53] J. C. Duque, L. Anselin, and S. J. Rey. The Max-p-regions Problem. *Journal of Regional Science*, 52(3):397–419, 2012.

[54] M. G. Elizondo-Amaya, R. Z. Ríos-Mercado, and J. A. Díaz. A dual bounding scheme for a territory design problem. *Computers & Operations Research*, 44: 193–205, 2014.

[55] E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990.

[56] E. Erkut, Y. Ülküsal, and O. Yeniçerioğlu. A comparison of p-dispersion heuristics. *Computers & Operations research*, 21(10):1103–1113, 1994.

[57] R. Z. Farahani and M. Hekmatfar, editors. *Facility Location: Concepts, Models, Algorithms and Case Studies*. Springer-Verlag, Berlin, Germany, 2009.

[58] H. Farughi, M. Tavana, S. Mostafayi, and F. J. Santos Arteaga. A novel optimization model for designing compact, balanced, and contiguous healthcare districts. *Journal of the Operational Research Society*, 71(11):1740–1759, 2020.

[59] J. A. Ferland and G. Guénette. Decision support system for the school districting problem. *Operations Research*, 38(1):15–21, 1990.

[60] E. Fernández, J. Kalcsics, S. Nickel, and R. Z. Ríos-Mercado. A novel maximum dispersion territory design model arising in the implementation of the weee-directive. *Journal of the Operational Research Society*, 61(3):503–514, 2010.

[61] E. Fernández, J. Kalcsics, and S. Nickel. The maximum dispersion problem. *Omega*, 41(4):721–730, 2013.

[62] J. Ferreira Neto. Private communication, 2018.

[63] B. Fifield, M. Higgins, K. Imai, and A. Tarr. Automated redistricting simulation using Markov chain Monte Carlo. *Journal of Computational and Graphical Statistics*, 29(4):715–728, 2020.

[64] M. Fischetti, M. Leitner, I. Ljubić, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, and M. Sinnl. Thinning out steiner trees: a node-based model for uniform edge costs. *Mathematical Programming Computation*, 9(2):203–229, 2017.

[65] B. Fleischmann and J. N. Paraschis. Solving a large scale districting problem: a case report. *Computers & Operations Research*, 15(6):521–533, 1988.

[66] L. Ford and D. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[67] S. L. Forman and Y. Yue. Congressional districting using a TSP-based genetic algorithm. In E. Cantú-Paz, J. A. Foster, K. Deb, L. David, and R. Rajkumar, editors, *Genetic and Evolutionary Computation – GECCO 2003*, volume 2723 of *Lecture Notes in Computer Science*, pages 2072–2083. Springer, Berlin, Germany, 2003.

[68] G. García-Ayala, J. L. González-Velarde, R. Z. Ríos-Mercado, and E. Fernández. A novel model for arc territory design: promoting Eulerian districts. *International Transactions in Operational Research*, 23(3):433–458, 2016.

[69] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. Freeman, New York, United States, 1979.

[70] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C.-C. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.

[71] R. S. Garfinkel and G. L. Nemhauser. Optimal political districting by implicit enumeration techniques. *Management Science*, 16(8):B495–B508, 1970.

[72] A. Gelman. *Red State, Blue State, Rich State, Poor State: Why Americans Vote the Way They Do*. Princeton University Press, Princeton, United States, 2009.

[73] M. Gendreau and J.-Y. Potvin, editors. *Handbook of Metaheuristics*. Springer, New York, United States, 2nd edition, 2010.

[74] J. A. George, B. W. Lamar, and C. A. Wallace. Political district determination using large-scale network optimization. *Socio-Economic Planning Sciences*, 31 (1):11–28, 1997.

[75] A. F. Gil, M. G. Sánchez, C. Castro, and A. Pérez-Alonso. A mixed-integer linear programming model and a metaheuristic approach for the selection and allocation of land parcels problem. *International Transactions in Operational Research*, Forthcoming (DOI: 10.1111/itor.13115).

[76] B. E. Gillett and L. R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349, 1974.

[77] A. Gliesch and M. Ritt. A generic approach to districting with diameter or center-based objectives. In M. López-Ibáñez, editor, *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 249–257, Prague, Czech Republic, July 2019. ACM.

[78] A. Gliesch and M. Ritt. A hybrid heuristic for the maximum dispersion problem. *European Journal of Operational Research*, 288(3):721–735, 2021.

[79] A. Gliesch and M. Ritt. A new heuristic for finding verifiable k-vertex-critical subgraphs. *Journal of Heuristics*, 28:61–91, 2022.

[80] A. Gliesch, M. Ritt, and M. C. Moreira. A genetic algorithm for fair land allocation. In P. A. N. Bosman, editor, *GECCO '17: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 793–800, Berlin, Germany, July 2017. ACM.

[81] A. Gliesch, M. Ritt, and M. C. Moreira. A multistart alternating tabu search for commercial districting. In A. Liefooghe and M. López-Ibáñez, editors, *European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 10782 of *Lecture Notes in Computer Science*, pages 158–173, Cham, Switzerland, 2018. Springer.

[82] A. Gliesch, M. Ritt, A. H. S. Cruz, and M. C. O. Moreira. A heuristic algorithm for districting problems with similarity constraints. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, Glasgow, United Kingdom, July 2020. IEEE.

[83] A. Gliesch, M. Ritt, A. H. S. Cruz, and M. C. O. Moreira. A hybrid heuristic for districting problems with routing criteria. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–9, Glasgow, United Kingdom, July 2020. IEEE.

[84] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision S ciences*, 8(1):156–166, 1977.

[85] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[86] F. Glover and J.-K. Hao. The case for strategic oscillation. *Annals of Operations Research*, 183(1):163–173, 2011.

[87] S. Goderbauer and J. Winandy. Political districting problem: Literature review and discussion with regard to federal elections in Germany. Technical report, RWTH Aachen, Aachen, Germany, 2018.

[88] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, 1:69–93, 1991.

[89] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[90] E. C. Griffith. *The Rise and Development of the Gerrymander*. Scott, Foresman, United States, 1907.

[91] W. Gurnee. Scalable approximations of capacitated k-medians for political districting. Technical report, Cornell University, Ithaca, United States, 2020.

[92] W. Gurnee and D. B. Shmoys. Fairmandering: A column generation heuristic for fairness-optimized political districting. *Preprint arXiv:2103.11469*, 2021.

[93] G. Gutin and A. P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Springer, Kluwer, Dordrecht, The Netherlands, 2002.

[94] S. Hanafi, A. Freville, and P. Vaca. Municipal solid waste collection: An effective data structure for solving the sectorization problem with local search methods. *Information Systems and Operational Research*, 37(3):236–254, 1999.

[95] P. Hansen, N. Mladenović, J. Brimberg, and J. A. M. Pérez. Variable neighborhood search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 57–97. Springer, Cham, Switzerland, 2019.

[96] S. Har-Peled. On the expected complexity of random convex hulls. *Preprint arXiv:1111.53404*, 2011.

[97] D. Haugland, S. C. Ho, and G. Laporte. Designing delivery districts for the vehicle routing problem with stochastic demands. *European Journal of Operational Research*, 180(3):997–1010, 2007.

[98] A. Hertz and D. Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[99] 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.

[100] 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.

[101] C. Hierholzer and C. Wiener. Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

[102] A. K. Hirose, C. T. Scarpin, and J. E. P. Junior. Goal programming approach for political districting in Santa Catarina state: Brazil. *Annals of Operations Research*, 287(1):209–232, 2020.

[103] M. Hojati. Optimal political districting. *Computers & Operations Research*, 23 (12):1147–1161, 1996.

[104] C. Hojny, I. Joormann, H. Lüthen, and M. Schmidt. Mixed-integer programming techniques for the connected max-k-cut problem. *Mathematical Programming Computation*, 13(1):75–132, 2021.

[105] J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[106] D. L. Horn, C. R. Hampton, and A. J. Vandenberg. Practical application of district compactness. *Political Geography*, 12(2):103–120, 1993.

[107] D. L. Huerta-Muñoz, R. Z. Ríos-Mercado, and R. Ruiz. An iterated greedy heuristic for a market segmentation problem with multiple attributes. *European Journal of Operational Research*, 261(1):75–87, 2017.

[108] A. I. Jarrah and J. F. Bard. Large-scale pickup and delivery work area design. *Computers & Operations Research*, 39(12):3102–3118, 2012.

[109] T. R. Jensen and B. Toft. *Graph Coloring Problems*. Wiley, New York, United States, 2005.

[110] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. Wiley, Chichester, United Kingdom, 1997.

[111] J. Kalcsics and R. Z. Ríos-Mercado. Districting problems. In G. Laporte, S. Nickel, and F. Saldanha da Gama, editors, *Location Science*, chapter 25, pages 703–741. Springer, Cham, Switzerland, 2nd edition, 2019.

[112] J. Kalcsics, T. Melo, S. Nickel, and H. Gündra. Planning sales territories – A facility location approach. In P. Chamoni, R. Leisten, A. Martin, J. Minnemann, and H. Stadtler, editors, *Operations Research Proceedings*, pages 141–148, Germany, 2002. Springer.

[113] J. Kalcsics, S. Nickel, and M. Schröder. Towards a unified territorial design approach – Applications, algorithms and GIS integration. *Top*, 13(1):1–56, 2005.

[114] J. Kalcsics, S. Nickel, and M. Schröder. A generic geometric approach to territory design and districting. Technical report, Karlsruhe Institute of Technology, Karlsruhe, Germany, 2009.

[115] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. *SIAM Journal on Applied Mathematics*, 37(3):513–560, 1979.

[116] A. R. Kaufman, G. King, and M. Komisarchik. How to measure legislative district compactness if you only know it when you see it. *American Journal of Political Science*, 65(3):533–550, 2021.

[117] J. M. Keil. Polygon decomposition. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 11, pages 491–518. Elsevier, Amsterdam, The Netherlands, 2000.

[118] D. M. King, S. H. Jacobson, E. C. Sewell, and W. K. T. Cho. Geo-graphs: An efficient model for enforcing contiguity and hole constraints in planar graph partitioning. *Operations Research*, 60(5):1213–1228, 2012.

[119] D. M. King, S. H. Jacobson, and E. C. Sewell. Efficient geo-graph contiguity and hole algorithms for geographic zoning and dynamic plane graph partitioning. *Mathematical Programming*, 149(1-2):425–457, 2014.

[120] D. M. King, S. H. Jacobson, and E. C. Sewell. The geo-graph in practice: creating united states congressional districts from census blocks. *Computational Optimization and Applications*, 69(1):25–49, 2018.

[121] S. M. Korman. *Graph colouring and related problems in operations research*. PhD thesis, Imperial College London, London, United Kingdom, 1975.

[122] J. Łącki and P. Sankowski. Optimal decremental connectivity in planar graphs. *Theory of Computing Systems*, 61(4):1037–1053, 2017.

[123] J. Łącki, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Single source–all sinks max flows in planar digraphs. In *IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 599–608, New Brunswick, Canada, 2012. IEEE.

[124] X. Lai and J. K. Hao. Iterated maxima search for the maximally diverse grouping problem. *European Journal of Operational Research*, 254(3):780–800, 2016.

[125] G. Laporte and I. H. Osman. Routing problems: A bibliography. *Annals of Operations Research*, 61(1):227–262, 1995.

[126] G. Laporte, S. Nickel, and F. Saldanha da Gama, editors. *Location Science*. Springer, Cham, Switzerland, 2nd edition, 2019.

[127] H. Lei, G. Laporte, and B. Guo. Districting for routing with stochastic customers. *EURO Journal on Transportation and Logistics*, 1(1):67–85, 2012.

[128] H. Lei, G. Laporte, Y. Liu, and T. Zhang. Dynamic design of sales territories. *Computers & Operations Research*, 56:84–92, 2015.

[129] H. Lei, R. Wang, and G. Laporte. Solving a multi-objective dynamic stochastic districting and routing problem with a co-evolutionary algorithm. *Computers & Operations Research*, 67:12–24, 2016.

[130] H. A. Levin and S. A. Friedler. Automated congressional redistricting. *Journal of Experimental Algorithmics*, 24:1–24, 2019.

[131] R. Lewis, J. Thompson, C. Mumford, and J. Gillard. A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers & Operations Research*, 39(9):1933–1950, 2012.

[132] W. Li, R. L. Church, and M. F. Goodchild. The p-compact-regions problem. *Geographical Analysis*, 46(3):250–273, 2014.

[133] F. Liberatore, M. Camacho-Collados, and L. Quijano-Sánchez. Equity in the police districting problem: Balancing territorial and racial fairness in patrolling operations. *Journal of Quantitative Criminology*, 38:1–25, 2022.

[134] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):5–55, 1932.

[135] H.-Y. Lin and J.-J. Kao. Subregion districting analysis for municipal solid waste collection privatization. *Journal of the Air & Waste Management Association*, 58 (1):104–111, 2008.

[136] M. Lin, K.-S. Chin, C. Fu, and K.-L. Tsui. An effective greedy method for the meals-on-wheels service districting problem. *Computers & Industrial Engineering*, 106:1–19, 2017.

[137] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the Traveling-Salesman Problem. *Operations Research*, 21(2):498–516, 1973.

[138] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[139] J. F. López-Pérez and R. Z. Ríos-Mercado. Embotelladoras ARCA uses operations research to improve territory design plans. *Interfaces*, 43(3):209–220, 2013.

[140] T. Lotan, D. Catrysse, and D. Van Oudheusden. Winter gritting in the province of antwerp: a combined location and routing problem. *Belgian Journal of Operations Research, Statistics, and Computer Science*, 36(2-3):141–157, 1996.

[141] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, chapter 12, pages 129–168. Springer, New York, United States, 2nd edition, 2019.

[142] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, volume 1, pages 281–297, Oakland, United States, 1967.

[143] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, New York, United States, 1990.

[144] C. McCartan and K. Imai. Sequential Monte Carlo for sampling balanced and compact redistricting plans. *Preprint arXiv:2008.06131*, 2020.

[145] M. W. McConnell. The redistricting cases: Original mistakes and current consequences. *Harvard Journal of Law and Public Policy*, 24:103, 2000.

[146] A. Mehrotra, E. L. Johnson, and G. L. Nemhauser. An optimization based heuristic for political districting. *Management Science*, 44(8):1100–1114, 1998.

[147] J. Min and D. Savage. Why do american indians vote democratic? *The Social Science Journal*, 51(2):167–180, 2014.

[148] F. K. Miyazawa, P. F. Moura, M. J. Ota, and Y. Wakabayashi. Partitioning a graph into balanced connected classes: Formulations, separation and experiments. *European Journal of Operational Research*, 293(3):826–836, 2021.

[149] M. Moeini and O. Wendt. A heuristic for solving the maximum dispersion problem. In A. Fink, A. Fügenschuh, and M. J. Geiger, editors, *Operations Research Proceedings 2016*, pages 405–410, Cham, Switzerland, 2018. Springer.

[150] M. Moeini, D. Goerzen, and O. Wendt. A local search heuristic for solving the maximum dispersion problem. In N. T. Nguyen, D. H. Hoang, T.-P. Hong, H. Pham, and B. Trawiński, editors, *Intelligent Information and Database Systems*, volume 10751 of *Lecture Notes in Computer Science*, pages 362–371, Cham, Switzerland, 2018. Springer.

[151] M. C. O. Moreira, J. A. Ferreira Neto, C. J. Einloft, and N. T. C. Silva. *Desenvolvimento Rural, Sustentabilidade e Prdenamento Territorial*, chapter O uso da busca tabu no ordenamento territorial em assentamentos rurais: reconfigurando o SOTER-PA (Sistema de Ordenamento Territorial da Reforma Agrária e Planejamento Ambiental), pages 265–272. Suprema, Visconde do Rio Branco, Brazil, 2011.

[152] S. Moreno, J. Pereira, and W. Yushimito. A hybrid k-means and integer programming method for commercial territory design: a case study in meat distribution. *Annals of Operations Research*, 286(1):87–117, 2020.

[153] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. C3P Report 826, California Institute of Technology, Pasadena, United States, 1989.

[154] J. G. Moya-García and M. A. Salazar-Aguilar. Territory design for sales force sizing. In R. Z. Ríos-Mercado, editor, *Optimal Districting and Territory Design*, volume 284 of *International Series in Operations Research and Management Science*, chapter 10, pages 191–206. Springer, Cham, Switzerland, 2020.

[155] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7(1):65–85, 1988.

[156] J. M. Mulvey and M. P. Beck. Solving capacitated clustering problems. *European Journal of Operational Research*, 18(3):339–348, 1984.

[157] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[158] F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[159] 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.

[160] L. Muyldermans, D. Cattrysse, and D. V. Oudheusden. District design for arc-routing applications. *Journal of the Operational Research Society*, 54(11):1209–1221, 2003.

[161] R. G. Niemi, B. Grofman, C. Carlucci, and T. Hofeller. Measuring compactness and the role of a compactness standard in a test for partisan and racial gerrymandering. *The Journal of Politics*, 52(4):1155–1181, 1990.

[162] B. Nygreen. European assembly constituencies for Wales – comparing of methods for solving a political districting problem. *Mathematical Programming*, 42 (1):159–169, 1988.

[163] J. Oehrlein and J.-H. Haunert. A cutting-plane method for contiguity-constrained spatial aggregation. *Journal of Spatial Information Science*, 2017 (15):89–120, 2017.

[164] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166–204, 1981.

[165] E. Parliament and T. C. of the European Union. Directive 2012/19/EU on waste electrical and electronic equipment (WEEE), Jul. 2012. URL http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2012:197:0038:0071:EN:PDF.

[166] F. Pezzella, R. Bonanno, and B. Nicoletti. A system approach to the optimal health-care districting. *European Journal of Operational Research*, 8(2):139–146, 1981.

[167] D. D. Polsby and R. D. Popper. The third criterion: Compactness as a procedural safeguard against partisan gerrymandering. *Yale Law & Policy Review*, 9(2): 301–353, 1991.

[168] J. Poot, O. Alimi, M. P. Cameron, and D. C. Maré. The gravity model of migration: The successful comeback of an ageing superstar in regional science. Technical Report IZA DP No. 10329, IZA Institute of Labor Economics, Bonn, Germany, October 2016.

[169] C. Prodhon and C. Prins. A survey of recent research on location-routing problems. *European Journal of Operational Research*, 238(1):1–17, 2014.

[170] O. A. Prokopyev, N. Kong, and D. L. Martinez-Torres. The equitable dispersion problem. *European Journal of Operational Research*, 197(1):59–67, 2009.

[171] E. C. Reock. A note: Measuring compactness as a requirement of legislative apportionment. *Midwest Journal of Political Science*, 5(1):70–74, 1961.

[172] M. G. C. Resende and C. C. Ribeiro. *Optimization by GRASP: Greedy Randomized Adaptive Search Procedure*. Springer, New York, United States, 2016.

[173] M. G. C. Resende and R. F. Werneck. A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10(1):59–88, 2004.

[174] F. Ricca and B. Simeone. Local search algorithms for political districting. *European Journal of Operational Research*, 189(3):1409–1426, 2008.

[175] F. Ricca, A. Scozzari, and B. Simeone. Weighted Voronoi region algorithms for political districting. *Mathematical and Computer Modelling*, 48(9-10):1468–1477, 2008.

[176] F. Ricca, A. Scozzari, and B. Simeone. Political districting: From classical models to recent approaches. *Annals of Operations Research*, 204(1):271–299, 2013.

[177] R. Z. Ríos-Mercado. Assessing a metaheuristic for large-scale commercial districting. *Cybernetics and Systems*, 47(4):321–338, 2016.

[178] R. Z. Ríos-Mercado, editor. *Optimal Districting and Territory Design*, volume 284 of *International Series in Operations Research and Management Science*. Springer, Cham, Switzerland, 2020.

[179] R. Z. Ríos-Mercado and J. F. Bard. An exact algorithm for designing optimal districts in the collection of waste electric and electronic equipment through an improved reformulation. *European Journal of Operational Research*, 276(1): 259–271, 2019.

[180] R. Z. Ríos-Mercado and H. J. Escalante. GRASP with path relinking for commercial districting. *Expert Systems with Applications*, 44:102–113, 2016.

[181] R. Z. Ríos-Mercado and E. Fernández. A reactive GRASP for a commercial territory design problem with multiple balancing requirements. *Computers & Operations Research*, 36(3):755–776, 2009.

[182] R. Z. Ríos-Mercado and J. F. López-Pérez. Commercial territory design planning with realignment and disjoint assignment requirements. *Omega*, 41(3): 525–535, 2013.

[183] R. Z. Ríos-Mercado and J. C. Salazar-Acosta. A GRASP with strategic oscillation for a commercial territory design problem with a routing budget constraint. In I. Batyrshin and G. Sidorov, editors, *Advances in Soft Computing*, volume 7095 of *Lecture Notes in Artificial Intelligence*, pages 307–318. Springer, Heidelberg, Germany, 2011.

[184] R. Z. Ríos-Mercado, A. M. Álvarez-Socarrás, A. Castrillón, and M. C. López-Locés. A location-allocation-improvement heuristic for districting with multiple-activity balancing constraints and p-median-based dispersion minimization. *Computers & Operations Research*, 126:105106, 2021.

[185] R. Z. Ríos-Mercado, J. L. González-Velarde, and J. R. Maldonado-Flores. Tabu search with strategic oscillation for improving collection assignment plans of waste electric and electronic equipment. *International Transactions of Operational Research*, 30(2):1002–1030, 2023.

[186] M. Ritt and J. Pereira. Heuristic and exact algorithms for minimum-weight non-spanning arborescences. *European Journal of Operational Research*, 287 (1):61–75, 2020.

[187] M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and M. Cabrera-Ríos. New models for commercial territory design. *Networks and Spatial Economics*, 11(3):487–507, 2011.

[188] M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and J. L. González-Velarde. A bi-objective programming model for designing compact and balanced territories in commercial districting. *Transportation Research Part C: Emerging Technologies*, 19(5):885–895, 2011.

[189] M. A. Salazar-Aguilar, J. L. González-Velarde, and R. Z. Ríos-Mercado. A divide-and-conquer approach to commercial territory design. *Computación y Sistemas*, 16(3):309–320, 2012.

[190] M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, J. L. González-Velarde, and J. Molina. Multiobjective scatter search for a commercial territory design problem. *Annals of Operations Research*, 199(1):343–360, 2012.

[191] M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and J. L. González-Velarde. Grasp strategies for a bi-objective commercial territory design problem. *Journal of Heuristics*, 19(2):179–200, 2013.

[192] M. G. Sandoval, J. A. Díaz, and R. Z. Ríos-Mercado. An improved exact algorithm for a territory design problem with p-center-based dispersion minimization. *Expert Systems with Applications*, 146:113150, 2020.

[193] M. G. Sandoval, E. Álvarez-Miranda, J. Pereira, R. Z. Ríos-Mercado, and J. A. Díaz. A novel districting design approach for on-time last-mile delivery: An application on an express postal company. *Omega*, 113:102687, 2022.

[194] O. B. Schoepfle and R. L. Church. A new network representation of a "classic" school districting problem. *Socio-Economic Planning Sciences*, 25(3):189–197, 1991.

[195] M. Schröder. *Gebiete optimal aufteilen*. PhD thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, 2001.

[196] J. Schwartz. Cracked, stacked and packed: Initial redistricting maps met with skepticism and dismay. https://indyweek.com/news/northcarolina/cracked-stacked-packed-initial-redistricting-maps-met-skepticism-di 2011. Retrieved in 12-12-2022.

[197] J. A. Segura-Ramiro, R. Z. Ríos-Mercado, A. M. Álvarez-Socarrás, and K. de Alba Romenus. A location-allocation heuristic for a territory design problem in a beverage distribution firm. In *International Conference on Industrial Engineering Theory, Applications, and Practice*, pages 428–434, 2007.

[198] M. I. Shamos. *Computational Geometry*. PhD thesis, Yale University, New Haven, United States, 1978.

[199] T. Shirabe. Districting modeling with exact contiguity constraints. *Environment and Planning B: Planning and Design*, 36(6):1053–1066, 2009.

[200] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, 2014.

[201] M. T. A. Steiner, D. Datta, P. J. S. Neto, C. T. Scarpin, and J. R. Figueira. Multi-objective optimization in partitioning the healthcare system of Paraná state in Brazil. *Omega*, 52:53–64, 2015.

[202] N. O. Stephanopoulos and E. M. McGhee. Partisan gerrymandering and the efficiency gap. *University of Chicago Law Review*, 82:831, 2015.

[203] Q. J. Stewart. The development of social physics. *American Journal of Physics*, 18(5):239–253, 1950.

[204] G. Syswerda et al. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, volume 3, pages 2–9, George Mason University, United States, 1989.

[205] M. Takahashi and H. Kita. A crossover operator using independent component analysis for real-coded genetic algorithms. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 1, pages 643–649, Seoul, Korea, 2001. IEEE.

[206] R. E. Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161, 1974.

[207] F. Tavares-Pereira, J. R. Figueira, V. Mousseau, and B. Roy. Multiple criteria districting problems: The public transportation network pricing system of the paris region. *Annals of Operations Research*, 154(1):69–92, 2007.

[208] M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of Computing*, pages 112–119, Baltimore, United States, May 2005.

[209] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications*. SIAM, Philadephia, United States, 2nd edition, 2014.

[210] User “cmglee” at English Wikipedia. Gerrymandering 9-6. https://commons.wikimedia.org/wiki/File:Gerrymandering_9-6.png, 2010. Retrieved in 12-12-2022.

[211] H. Validi and A. Buchanan. Political districting to minimize cut edges. *Mathematical Programming Computation*, 14:623–672, 2022.

[212] H. Validi, A. Buchanan, and E. Lykhovyd. Imposing contiguity constraints in political districting models. *Operations Research*, 70(2):867–892, 2022.

[213] B. Vangerven, D. Briskorn, D. R. Goossens, and F. C. Spieksma. Parliament seating assignment problems. *European Journal of Operational Research*, 296 (3):914–926, 2022.

[214] G. S. Warrington. Quantifying gerrymandering using the vote distribution. *Election Law Journal*, 17(1):39–57, 2018.

[215] G. R. Webster. Reflections on current criteria to evaluate redistricting plans. *Political Geography*, 32:3–14, 2013.

[216] I. K. White and C. N. Laird. *Steadfast Democrats: How Social Forces Shape Black Political Behavior*. Princeton University Press, Princeton, United States, 2020.

[217] J. C. Williams, Jr. Political districting: A review. *Regional Science*, 74(1):13–40, 1995.

[218] S. Yanik, Ö. Sürer, and B. Öztayşi. Designing sustainable energy regions using genetic algorithms and location-allocation approach. *Energy*, 97:161–172, 2016.

[219] H. P. Young. Measuring the compactness of legislative districts. *Legislative Studies Quarterly*, 13(1):105–115, 1988.

[220] A. A. Zoltners and P. Sinha. Sales territory alignment: A review and model. *Management Science*, 29(11):1237–1256, 1983.

[221] A. A. Zoltners, P. Sinha, and S. E. Lorimer. Sales force effectiveness: a framework for researchers and practitioners. *Journal of Personal Selling & Sales Management*, 28(2):115–131, 2008.

[222] B. Çavdar and J. Sokol. A distribution-free TSP tour length estimation model for random graphs. *European Journal of Operational Research*, 243(2):588–598, 2015.