Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik

Dissertation

Competitive and Voting Location

Joachim Spoerhase

December 2009

Supervisor: Prof. Dr. Hartmut Noltemeier

Referees: Prof. Dr. Hartmut Noltemeier
Prof. Dr. Alexander Wolff
Prof. Dr. Horst A. Eiselt

Zusammenfassung

Gegenstand dieser Arbeit sind bestimmte Klassen von *Standortproblemen*. Bei einem Standortproblem geht es darum, einen geeigneten Standort für eine neu zu eröffnende Einrichtung zu finden. Beispiele für solche Einrichtungen sind Geschäfte, Warenlager, Fabriken, Krankenhäuser, aber auch Server in einem Computer-Netzwerk. Welche Standorte geeignet sind, wird durch ein numerisches Qualitätsmaß vorgegeben, das auf den Abständen des jeweiligen Standorts zu den Kunden oder Benutzern der Einrichtung beruht. Aufgrund der Vielzahl möglicher Anwendungen gibt es verschiedenste Arten solcher Qualitätsmaße.

In dieser Arbeit werden zwei Klassen von Standortproblemen, sogenannte *kompetitive Standortprobleme* und *Voting-Standortprobleme*, untersucht. Im Folgenden werden beide Problemklassen kurz dargestellt.

Für viele in der Fachliteratur betrachtete Standortprobleme unterstellt man einen einzelnen, monopolistischen Anbieter, der einen Standort für eine neu zu eröffnende Einrichtung sucht. Im Gegensatz dazu geht man bei *kompetitiven Standortproblemen* von zwei oder mehr Anbietern aus, die um Kundschaft konkurrieren. Die vorliegende Arbeit konzentriert sich auf Modelle mit genau zwei Wettbewerbern, genannt *Leader* und *Follower*. Es wird angenommen, dass diese Anbieter ähnliche Produkte zu ähnlichen Preisen anbieten. Aus Sicht der Kunden unterscheiden sich die Anbieter dann lediglich durch ihre Standorte. Jeder Kunde wählt den ihm nächstgelegenen Anbieter. Die Platzierung der Einrichtungen von Leader und Follower erfolgt sequentiell: Zunächst wählt der Leader seinen Standort. Die Kenntnis dieses Standorts ermöglicht es dem Follower, einen Standort für seine Einrichtung zu bestimmen, der seinen eigenen Ertrag (Gesamtbedarf seiner Kunden) maximiert. Dadurch wird das Verhalten des Followers für den Leader vorhersehbar, was dieser bei seiner initialen Entscheidung berücksichtigen kann, um einen für ihn optimalen Standort zu finden. In den hier betrachteten Modellen agieren die Wettbewerber rein eigennützig.

Bei *Voting-Standortproblemen* geht es hingegen um soziale Gesichtspunkte: Ein Anbieter versucht, einen Standort zu bestimmen, der die Benutzer oder Kunden soweit wie möglich zufrieden stellt. Solche Fragestellungen sind beispielsweise bei der Planung öffentlicher Einrichtungen relevant.

In den meisten Fällen gibt es keinen Standort, der von allen Benutzern favorisiert wird. Daher muss ein Kompromiss gefunden werden. Hierzu werden Kriterien betrachtet, die auch in Wahlsystemen eingesetzt werden: Ein geeigneter Standort wird als Sieger einer gedachten Wahl verstanden, bei der die möglichen Standorte die zur Wahl stehenden Kandidaten und die Kunden die Wähler darstellen.

Der Zusammenhang zwischen kompetitiven Standortproblemen und Voting-Standortproblemen besteht darin, dass Standorte, die vorteilhaft für den Leader im kompetitiven Modell sind, auch gute Kompromisse im Sinne der Voting-Standortprobleme sind.

Die zentrale Fragestellung dieser Arbeit ist, wie schnell sich kompetitive Standortprobleme und Voting-Standortprobleme algorithmisch lösen lassen. Als Maß für die Effizienz eines Algorithmus dient seine asymptotische Laufzeit. Als Entscheidungsraum werden Graphen verwendet. Das heißt, die potentiellen Standorte und die Kunden werden durch Knoten oder Punkte auf Kanten eines Graphen modelliert. Die Abstände zwischen Kunden und Standorten ergeben sich durch Kürzeste-Wege-Distanzen im zugrundeliegenden Graphen.

Im Folgenden werden wesentliche Ergebnisse der vorliegenden Arbeit zusammengefasst.

Im ersten Teil der Arbeit werden *relaxierte Einfachstandortprobleme* behandelt. Bei einem Einfachstandortproblem platziert jeder Anbieter genau eine Einrichtung. Durch die Relaxierung wird eine gewisse Zurückhaltung der Kunden modelliert, wie sie in der realen Welt beobachtet werden kann: Ein Kunde ist erst dann bereit, zu einem neuen Anbieter (Follower) zu wechseln, wenn dieser signifikant besser als der bereits existierende ist. Im hier untersuchten Modell muss dazu die Differenz der Abstände des Kunden zu Leader bzw. Follower eine vorgegebene Schranke überschreiten.

Zuerst werden *monotone Gewinnfunktionen* als ein neues und allgemeines Modell für relaxierte kompetitive Standortprobleme und Voting-Standortprobleme eingeführt (Kapitel 3). Alle aus der Fachliteratur bekannten Einfachstandortprobleme, die in dieser Arbeit betrachtet werden, lassen sich durch monotone Gewinnfunktionen beschreiben. Es ergeben sich jedoch auch neue, bislang noch nicht untersuchte Probleme.

Im ersten Teil der Arbeit werden monotone Gewinnfunktionen vorrangig auf *Baumgraphen* betrachtet. Wir entwickeln einen Linearzeit-Algorithmus zur Bestimmung einer optimalen Lösung für eine monotone Gewinnfunktion auf einem Baum (Kapitel 4).

Das Problem, die Menge *aller* Lösungen einer monotonen Gewinnfunktion auf einem Baum zu bestimmen, erweist sich als schwieriger (Kapi-

tel 5). Wir beweisen (für bestimmte Rechnermodelle) eine untere Laufzeitschranke von $\Omega(n \log n)$, wobei n die Anzahl der Knoten im Baum ist. Auf der anderen Seite wird ein Algorithmus mit einer optimalen Laufzeit von $O(n \log n)$ zur Bestimmung der Lösungsmenge vorgestellt. In bestimmten Spezialfällen ist sogar lineare Laufzeit möglich. Als Nebenprodukt ergibt sich ein Linearzeit-Algorithmus für das sogenannte *Stackelbergproblem mit parametrischen Preisen*. Dies ist eine wesentliche Verbesserung gegenüber dem bekannten Algorithmus mit einer Laufzeit von $O(n^3 \log n)$.

Der zweite Teil der Arbeit befasst sich mit kompetitiven *Mehrfachstandortproblemen*. Hier platzieren Leader und Follower eine vorgegebene Anzahl von Einrichtungen. Kompetitive Mehrfachstandortprobleme erweisen sich als wesentlich schwieriger als die entsprechenden Einfachstandortprobleme.

Zunächst wird die Komplexität kompetitiver Mehrfachstandortprobleme auf *allgemeinen* Graphen untersucht (Kapitel 7). Wir betrachten das Problem des Leaders (Leader-Problem), eine optimale Platzierung zu finden, und zeigen, dass das korrespondierende Entscheidungsproblem Σ_2^p-vollständig ist. Dies verschärft das bekannte NP-Schwere-Resultat und legt nahe, dass das Leader-Problem schwieriger ist als viele typische, in der Fachliteratur untersuchte Optimierungsprobleme.

Wir untersuchen auch die Approximierbarkeit kompetitiver Mehrfachstandortprobleme. Wir geben eine untere Schranke von $n^{1-\varepsilon}$ für die Approximierbarkeit des Leader-Problems an. Diese zeigt, dass es (unter der Annahme $P \neq NP$) keinen Approximationsalgorithmus für das Leader-Problem gibt, der auch im Worst-Case zufriedenstellend approximiert. Für das Follower-Problem ergibt sich ein Verfahren mit konstanter Approximationsgüte. Wir zeigen, dass diese Güte die bestmögliche ist.

In den verbleibenden Kapiteln werden die kompetitiven Mehrfachstandortprobleme für Bäume und Pfade untersucht.

Der Komplexitätsstatus des *Leader-Problems* auf Bäumen ist eine seit langem offene Frage (Hakimi, 1990). Wir zeigen, dass das Leader-Problem sogar auf Pfaden NP-schwer ist, was diese Frage somit beantwortet (Kapitel 9). Auf der positiven Seite wird ein vollpolynomielles Approximationsschema (FPTAS) für Pfade entwickelt. Für verschiedene Spezialfälle des Leader-Problems auf Bäumen stellen wir effiziente Algorithmen vor oder beweisen die NP-Schwere.

Für das *Follower-Problem* auf Bäumen ist bereits ein Polynomialzeit-Algorithmus bekannt. Wir konzentrieren uns auf einen Spezialfall, in dem der Follower nur eine Einrichtung platziert, der Leader jedoch mehrere (Kapitel 8). Wir entwickeln einen Algorithmus mit $O(n \log n)$ Laufzeit, der das

entsprechende Follower-Problem für Bäume löst. Dieser schlägt das bislang schnellste Verfahren, welches eine Laufzeit von $O(n \log^2 n)$ hat.

Abstract

The subject of this thesis is the investigation of certain classes of *location problems*. A location problem aims at finding a suitable location for a new facility that is to be opened. Examples for such facilities are shops, warehouses, plants, hospitals, but also servers in a computer network. The adequacy of a location is specified in terms of a given numerical quality measure that is based on the distances between the respective location and the customers or users of the facility. Due to the multitude of possible applications there is also a variety of different quality measures.

This thesis focuses on two classes of location problems: so-called *competitive location problems* and *voting location problems*. In the following both problem classes are briefly described.

Many location problems dealt with in the literature assume the existence of a single monopolistic provider who wants to open a new facility and looks for a good location. In contrast, *competitive location* investigates scenarios where two or more competing providers place their facilities and users can decide between the providers. In this thesis, models with two sequentially acting competitors, *leader* and *follower*, are considered. It is assumed that both competitors offer the same type of good or service at the same price. Hence the user preference can be expressed solely in terms of distances to the locations of the facilities. Every customer chooses the closest competitor. Once the leader has chosen a location, it is the follower's turn to determine a location maximizing his own revenue (the total demand of his customers). Hence the follower's reaction is predictable, which the leader can take into account when making the initial decision. It is assumed that the competitors act non-cooperatively.

Voting location, in contrast, aims at identifying locations that meet *social criteria*. The provider wants to satisfy the *users* (customers) of the facility to be opened. In general, there is no location that is favored by all users. Therefore, a satisfactory compromise has to be found. To this end, criteria arising from voting theory are considered. The solution of the location problem is understood as the winner of a virtual election among the users of the facilities, in which the potential locations play the role of the candidates and the users represent the voters.

The link between competitive and voting location is that a location which is advantageous for the leader in competitive location is also a good compromise in terms of voting location.

The main question of this thesis is how fast competitive and voting location problems can be solved algorithmically. The measure for the efficiency of an algorithm is its asymptotic running time. The decision space is always a graph. That is, the potential locations and the customers are modeled by nodes or points on edges of a graph. The distances between customers and locations arise from the shortest-path distances in the underlying graph.

In the following we summarize some important results of this work.

The first part of this thesis examines *relaxed single location problems*. In a single location problem each provider places exactly one facility. The relaxation is a means of modeling some kind of reluctance of the customers as it can be observed in the real world: A customer is willing to change his provider only if the new provider (the follower) is significantly better. In the model investigated here, a customer changes to the follower if the difference of the distances of this customer to leader and follower, respectively, exceeds a given threshold.

First, we introduce *monotonic gain functions* as a new and general model for relaxed competitive and voting location problems. (Chapter 3). All particular single location problems that are known from literature and treated in this thesis can be described by such a monotonic gain function. But also new problems arise.

In the first part of this thesis we examine monotonic gain functions primarily on tree graphs. We develop a linear time algorithm for determining an optimal solution of a monotonic gain function on a tree (Chapter 4).

The problem of computing the set of *all* solutions of a monotonic gain function on a tree turns out to be more difficult. We prove a lower bound of $\Omega(n \log n)$ on the running time of any algorithm (in terms of certain computer models), where n is the number of nodes in the tree. On the other hand, an algorithm with an optimal running time of $O(n \log n)$ is proposed. In certain special cases we can even achieve linear time. As a byproduct we obtain a linear time algorithm for the so-called *Stackelberg problem with parametric prices*. This is a considerable improvement upon the existing algorithm that has a running time of $O(n^3 \log n)$.

The second part of this thesis deals with *multiple* competitive location. Here, leader and follower place a given number of facilities each. It turns out that multiple competitive location problems are much harder than their counterparts in single competitive location.

First, the complexity of multiple competitive location problems on *general* graphs is analyzed (Chapter 7). We consider the problem of the leader (leader problem, for short) to find an optimal placement and show that the corresponding decision problem is Σ_2^p-complete. This sharpens the known NP-hardness result and suggests that the leader problem is harder than many typical optimization problems known from the literature.

We investigate also the approximability of multiple competitive location problems. We prove a lower bound of $n^{1-\varepsilon}$ on the approximability of the leader problem. This shows (assuming $P \neq NP$) that there is no approximation algorithm which approximates satisfactorily also in the worst case. For the follower problem an algorithm with a constant approximation ratio is given. It is also shown that this factor is best possible.

In the remaining chapters multiple competitive location problems are investigated on trees and paths.

The complexity status of the *leader problem* on trees has been a long-standing open question (Hakimi, 1990). We show that the leader problem is NP-hard even on paths thereby answering this question (Chapter 9). On the positive side a fully polynomial time approximation scheme (FPTAS) for paths is developed. Additionally, we consider several special cases of the leader problem on trees for which we either propose efficient algorithms or provide NP-hardness proofs.

For the *follower problem* on trees a polynomial time algorithm is already known. We focus on the special case where the follower places only one facility while the leader locates a plurality of facilities (Chapter 8). We present an algorithm that runs in $O(n \log n)$ time. This beats the best existing algorithm for this problem, which has a running time of $O(n \log^2 n)$.

Contents

1. Introduction and Preliminaries

1.1. Introduction

Location Problems

An old saying states that the three attributes of property that matter most are location, location, location. In fact, the choice of adequate locations for *facilities* is one of the fundamental problems arising from economic and public planning. Here, we use the notion "facility" in the broadest sense. It refers to any object for which it makes sense to speak of the quality of its location. Examples are shops, warehouses, plants, hospitals, schools, libraries, air and marine ports, but also servers in a computer network.

The origins of *location theory*, the field of locational issues, date back to the beginning of the 19th century when locational decision were mainly examined from the viewpoint of economic geography. Today, location has become a well-established, interdisciplinary field with a large body of publications in economy, geography, game theory, and computer science.

There are several reasons why location theory plays such a vital role. First, locational decisions are ubiquitous in human life. This ranges from the choice of well located flats by private individuals to the strategic placement of department industrial plants, or even the political and economic decisions for the location of a new airport. Second, opening a facility often involves considerable setup costs and is thus inherently long term or even irrevocable. This makes it especially important to select such a location carefully. And finally, locational decisions can have significant impact on private, economic, or public life. For example, one may think of the manifold effects for the citizens and the economy of a given town when a new department store is opened.

There are two main approaches in location theory.

The first direction deals with *descriptive* models and is mainly treated by economists and geographers. The goal is to explain existing locational configurations, for example, why some branches of industry are spatially concentrated around certain centers. In this area the focus is placed on devel-

oping realistic models that are able to describe the real world.

The second approach aims at providing quantitative methods that support the actual process of *decision making*. For example, we might ask at what specific place in a given town we should locate a new school such that a certain, well-defined quality criterion is met. Such problems are tackled by computer scientists and operations researchers when they deal with locational problems. And also this work belongs to that line of research.

Since our problems require the formulation of objective quality measures, we will *not* be concerned with locational decisions of single individuals, say, how to find a good flat in a given town. Such decisions are often dominated by subjective factors and considerations which are only valid in that particular case. Rather, we will investigate *large-scale scenarios* (such as finding locations for schools, libraries or fast food stores) that affect a plurality of users and where subjective factors of single individuals may be neglected.

Due to the wide-ranging applicability of this kind of questioning there is, however, no universal quality criterion of a location. Rather, an appropriate placement of facilities depends heavily on the respective scenario.

- A public library may be located so as to minimize the *sum* of distances to its readers (social welfare).
- The placement of hospitals requires that *each* potential patient have a sufficiently quick access route.
- The placement of servers in a computer network could maximize the quality of service.
- Transmission antennas could be placed so that as many clients as possible are covered.
- Ambulance or tow truck stations along major motorways (interstate highways) could require quick routes to key stretches.

Besides the quality criteria, these examples also differ in the structure of underlying decision space (that is, where the facilities can be placed): The example of the transmission antennas allows the provider to place within a given planar region. In contrast, the remaining examples involve some kind of *network* (road or computer network). The tow truck example differs from the other network scenarios in that the underlying network (motorways) has a particularly simple and sparse structure. In fact, a large part of this thesis is devoted to location problems on such simple network structures.

If one abstracts from the concrete case of application, the main concern is to select a location from a given space of potential positions such that the demands of the users of the facility are served according to a certain quality measure such as cost, travel distance or gain. We speak of a *location problem*. One often encounters scenarios where a multitude of facilities are to be placed all at once. Such problems are named *multiple location problems*. For example, we may imagine a fast food chain establishing a number of stores in a given city. The chain might aim at scattering its stores over the city so that as many customers as possible find at least one store in their vicinity.

There are two main steps when solving a locational problem in the aforementioned sense.

The first step consists in formulating an appropriate model and includes in particular the identification of a space of potential locations as well as the definition of an adequate quality measure.

In this work, we will mainly address the second step, namely, finding good or even optimal solutions to given locational models. That is, we are looking for *algorithms* that compute optimal or approximate solutions for certain location problems and use only a reasonable amount of computational resources such as time or space. As location problems, especially multiple ones, are often extremely difficult to solve we will also prove that some of the problems in question do not admit efficient algorithms at all. We remark that our analyses are of theoretical nature, that is, we concentrate on provable results on the asymptotic running time and the approximation performance of algorithms and the complexity of problems. We do *not* provide experimental evaluations or heuristic approaches.

We will focus on two sorts of location problems: *voting location problems* and *competitive location problems*.

Voting Location

Many location problems in literature are formulated from the viewpoint of the provider which locates the facility. We remember the example of the fast food chain that wants to locate a number of new stores in a given city. The goal of this company could be, for example, to minimize the setup costs of the stores, to maximize the revenue generated by the customers, or to maximize the net profit.

Voting location aims at satisfying the *users* (customers) of the facilities to be opened. The vital problem is that in general, different users have differ-

ent perceptions of what a good location is. Therefore, the goal is to find a satisfactory compromise that is tolerated by a majority of the users; much as in elections where a group of individuals decides on a candidate that may be considered acceptable by that group as a whole. To put it another way, voting location aims at identifying locations that meet *social criteria*. The solution of a location problem is understood as the winner of a virtual election among the users of the facilities. Therefore, we will consider criteria arising from voting theory.

The generic situation of an election, independent from location problems, is as follows. A number of *candidates* are up for election. There are *voters* each of which has *preferences* concerning the candidates. Based on those preferences a *winner* of the election is then determined by means of a suitable social choice function.

For this thesis, the *Condorcet criterion* is of special importance. A candidate A of an election is a *Condorcet winner* if there is no candidate B that is preferred over A by an absolute majority of the voters. Figure 1.1 depicts

Voter 1	Voter 2	Voter 3
Candidate B	Candidate C	Candidate D
Candidate A	Candidate A	Candidate A
Candidate D	Candidate B	Candidate C
Candidate C	Candidate D	Candidate B

FIG. 1.1.: Example of an election illustrating the Condorcet criterion. Each column specifies the preferences of the respective voter where a candidate is preferred over another one if he occurs further up in the column.

an example of an election. Here, each column specifies the preferences of the respective voter. For example, Voter 1 prefers Candidate B over Candidate A and Candidate A over Candidate C. In this scenario, Candidate D is *no* Condorcet winner since Candidate A is preferred over D by Voters 1 and 2, which is an absolute majority. Neither are Candidates B and C Condorcet winners. In contrast, Candidate A *is* a Condorcet winner since every opposing candidate is preferred by at most one voter.

We remark that the Condorcet criterion takes into consideration the complete preference structure of the voters. This has to be compared with electoral systems in which each voter casts a vote only for his favorite candidate. Electoral systems of this kind are often found in real political systems. Making use of such a system, Candidate A in Figure 1.1 would not win since he were the only Candidate who has not got any vote. In fact Candidate A

seems to be a "suitable" compromise since, albeit not the favorite of anyone, he is tolerated by all voters and may be considered as a "stable" solution.

A critical lack of the Condorcet principle is that a Condorcet winner does not need to exist. For an example, consider the scenario depicted in Figure 1.2. For this reason we will consider several *relaxations* of the Con-

Voter 1	Voter 2	Voter 3
Candidate A	Candidate C	Candidate B
Candidate B	Candidate A	Candidate C
Candidate C	Candidate B	Candidate A

FIG. 1.2.: Example of an election where no Condorcet winner exists.

dorcet criterion. One popular relaxation is the *Simpson criterion* where we require that the strongest opposition of the winner be preferred by a minimum number of voters.

In this thesis, we will apply voting to location theory in the following way. The users of the facilities play the role of the voters and the potential locations are considered as the candidates being up for election. The outcome of the location problem is then the location that corresponds to the winner of the corresponding virtual election. The preferences of the users arise from the distances between the users and the potential locations. That is, the users want to live as near to the facility as possible. The Condorcet principle can then be reformulated for location problems as follows. A location x is a *Condorcet location* if there is no location y such that more than half of the users is closer to y than to x.

Competitive Location

As we have seen, location theory deals with problems of optimally placing facilities so as to serve the individual demands of a given set of users. Many problems, including those we have discussed in the previous sections, consider the case where *one monopolistic* provider places all of the facilities. To put it another way, it is often assumed that all facilities are placed with respect to a uniform quality criterion.

Modern market economies are usually governed by *competitive mechanisms* whereas monopolies, often highly regulated, is desired only in a limited number of specific market ranges. The interaction of different autonomous actors or players, which is also a main aspect of competition,

is the domain of the flourishing field of *game theory*. To combine location theory and game theory by introducing competition in locational models might thus be seen as a contemporary aim.

Competitive location investigates scenarios where two (or more) providers place their facilities and users can decide between the providers.

In this thesis, we primarily assume that all competitors provide the same type of good or service, hence the user preference can be expressed solely in terms of distances to the locations of the servers. We will also assume that the competitors act non-cooperatively. That is, every competitor tries to maximize his own benefit.

Let's take up the example of the fast food chain and assume that the company has found adequate locations for its stores. As the business is going well and the chain's success becomes known, a second, competing company enters the market. Suppose that the differences regarding the choice of products, quality and price are negligible between both companies. Then it seems natural that a customer is willing to switch to the new company if the access route is shorter. Since we assume that the competitors act non-cooperatively, the new company aims only at maximizing its own revenue, which reduces to the maximization of sales as prices are fixed. More precisely, it seeks to locate its stores so that there are as many customers as possible for which the nearest store belongs to the new company.

Another example of competition for good locations are the aforementioned tow truck stations along major motorways. In case of a breakdown a car driver will prefer the closest station in order to get help as quickly as possible. A special feature of this scenario is the particularly simple structure of the underlying road network. This makes it easier to tackle the problem by means of efficient algorithms. In fact, identifying simple network structures that allow to solve competitive location problems efficiently is one of the main aspect of this thesis.

We will consider competitive location problems with two, sequentially moving players. In our terminology, the company (or more generally the provider) that comes first will be called *leader* and the second one *follower*.

The behavior of leader and follower depends heavily on their *knowledge*. In the foregoing example we assumed that the leader is not aware of the follower's plans of entering the market. The situation changes dramatically if the leader knows in advance the number of facilities to be placed by the follower. Once the leader has chosen his location, it is the follower's turn to determine an optimal location. Hence the follower's reaction is predictable, which the leader can take into account when he makes the initial decision. We call the resulting problem the *leader problem*. The *follower problem* consists

in determining an optimal placement for the follower under the presence of existing (leader) facilities.

It will be a recurring theme of this thesis that the leader problem is substantially harder to solve than that of the follower. But just for this very reason, the leader problem will also turn out to be a lot more interesting with respect to the techniques used to settle its computational complexity.

Overview

This thesis is divided into two parts.

The first part deals with *single* voting and competitive location. That is, each provider places a single facility. In this part, the focus will be on particularly fast algorithms on tree graphs.

The second part examines *multiple* voting and competitive location problems where a given number of facilities is to be placed. Here, we will concentrate on the complexity and approximability of those location problem. That is, we will systematically identify classes of graphs for which the problems are efficiently solvable and more complex classes for which they are hard. We will also investigate conditions under which we can efficiently compute approximate solutions that come close to the optimum.

Acknowledgments

I want to thank my adviser Professor Hartmut Noltemeier for his guidance and many valuable suggestions without which this thesis would not have been possible. I also thank PD Dr. Hans-Christoph Wirth for numerous, fruitful research discussions and for proofreading this thesis; Professor Alexander Wolff for giving me the opportunity to finish this work, for proofreading and many valuable improvement suggestions; Professor H.A. Eiselt for many valuable improvement suggestions; Marc Spisländer for proofreading; and Lisa Zauner, my colleagues, my family and my friends for their backing and patience during my research.

Credits

The results on the complexity (Section 7.3.2) and approximability (Section 7.4) of the (r, p)-centroid and the (r, X_p)-medianoid problem and the algorithms for computing the set of φ_0-bounded, the set of φ_0-tolerant and

the set of efficient Φ-solutions of leader independent monotonic gain functions on trees (parts of Chapter 5) are based on joint work with Hartmut Noltemeier and Hans-Christoph Wirth [NSW07]. The result on incremental (r, p)-centroid on degree-bounded graphs (Section 10.2) was jointly obtained with Hartmut Noltemeier, Jan Wiefel and Hans-Christoph Wirth [Wie08]. All other results (except for Section 8.2.4) were developed with Hans-Christoph Wirth [SW10, SW09b, SW09a, SW09c, SW08, SW07]. The algorithm of Section 8.2.4 has not been published yet.

1.2. Preliminaries

We assume that the reader is already familiar with the basics in theoretical computer science and graph theory. The aim of this section is to fix notations and terminology informally. Papadimitriou [Pap94], Ausiello et al. [ACG+99] and Even [Eve79] give more formal and comprehensive introductions to these fields.

The knowledgeable reader may skip this section confidently and use it as a reference when needed.

1.2.1. Decision and Optimization Problems

The subject of this thesis is the analysis of several *computational problems* from the area of competitive and voting location.

One of the most basic sort of computational problems consists in deciding whether a given *instance* of the problem satisfies a certain property.

Definition 1.2.1 A *decision problem* Π is characterized by a countable set D of instances and a partition $Y \cup N$ of D into a set Y of *positive* and a set N of *negative instances*.

In practical scenarios, we often encounter problems where we have to compute a solution which is as good as possible according to some given quality measure.

Definition 1.2.2 An *optimization problem* Π is a quadruple $(D, SOL, m, goal)$ satisfying the following properties.

1. D is some countable set of *instances*.
2. SOL is a mapping that assigns to each instance $I \in D$ a non-empty set $S(I)$ of *feasible solutions*.

3. m is a function, called *goal function*, that assigns to each pair (I, s) with $I \in D$ and $s \in SOL(I)$ a positive rational number $m(I, s)$ representing the quality of the feasible solution s with respect to I.

4. goal $\in \{\min, \max\}$ specifies whether Π is a *minimization* or a *maximization problem*.

By $SOL^*(I)$ we denote the set of *optimal solutions* of instance I, that is, all solutions from $SOL(I)$ such that $m(I, s)$ is maximum if goal $= \max$ and minimum if goal $= \min$. The corresponding value of the goal function is denoted by $m^*(I)$.

There are several ways to consider an optimization problem Π as a computational problem:

Construction problem Given an input instance $I \in D$ compute some optimal solution $s \in SOL^*(I)$.

Evaluation problem Given an input instance $I \in D$ compute the value $m^*(I)$.

Decision problem Given an input instance $I \in D$ and a positive number K decide whether $m^*(I) \geq K$ if goal $= \max$. If goal $= \min$ decide whether $m^*(I) \leq K$.

1.2.2. Algorithms

We will mainly analyze problems from an *algorithmic* point of view. That is, we will investigate the existence of *algorithms* meeting certain criteria.

In our context, an algorithm is what can be expressed as a program on a *RAM* (random access machine [Pap94]). A RAM is a simple abstract machine capable of modeling essential aspects of real computers. A RAM owns a countably infinite number of *registers*. Each of these register can store an integer number. The machine can perform the following *operations*: arithmetic operations $(+, -, \times, /)$, direct and indirect addressing and branching. A *program* is a sequence of *instructions* each of them referring to an operation of the above type.

Since arithmetic operations on the rationals can easily be simulated by integer arithmetic we will assume that our algorithms can also cope with rational numbers.

In order to allow our algorithms to solve problems we designate some of the registers to specify the *input* and the *output* of the algorithm, respectively.

For all problems we will deal with, it is possible to represent the input and the output by sequences of integer values.

1.2.3. Running Time of Algorithms

In the simplest case, the running time of an algorithm under a given input is the number of operations performed during the execution of the corresponding program. We call this model *unit-cost RAM*.

Following common practice, we will not be interested in the running time of an algorithm for a concrete input but rather in the *asymptotic growth* of the running time as a function of the *input size*. To this end we will make use of the well known big-oh notation.

Definition 1.2.3 Let $f, g\colon \mathbb{N} \to \mathbb{R}^+$ be two functions.

1. We say that $f \in O(g)$ if there are positive numbers c, n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
2. We say that $f \in \Omega(g)$ if $g \in O(f)$.
3. We say that $f \in \Theta(g)$ if $f \in O(g)$ and $g \in O(f)$ holds.
4. We say that $f \in o(g)$ if $f \in O(g)$ but not $f \in \Omega(g)$.
5. We say that $f \in \omega(g)$ if $f \in \Omega(g)$ but not $f \in O(g)$.

As it is the case for real computers, we imagine that the values stored in the registers of a given RAM are represented as a sequence of bits. The number of bits used to represent a value $n \in \mathbb{N}$ is called the *size* of n. Using the standard binary representation of integers we may assume that the size of a number n is $O(\log n)$. The size of the input can be represented in several ways. For example, we may simply count the number of integer values the input consists of. A more refined analysis may additionally take into account the size of those values.

We call an algorithm *efficient* if its running time is polynomially bounded. More formally, its running time is $O(n^k)$ for some fixed k where n represents the size of the input. For example, we can sort a set of n integers in time $O(n \log n)$ using the well-known merge sort algorithm. This is certainly a polynomial running time and therefore merge sort is an efficient algorithm.

It has been observed that the unit-cost RAM model, in its full generality, is not realistic since it allows for the multiplication of arbitrarily large numbers in a single step of computation [Pap94].

A more realistic model, called *log-cost RAM* assigns to each operation a time which is logarithmic in its operands. This is motivated by the fact that real processors use $O(\log n)$ bits to represent a number n and only a constant number of bits are manipulated during a single clock cycle.

For the sake of simplicity, we will use the *unit-cost model* avoiding "misuse" in the above sense, that is, we will take care that the registers always store values whose size is polynomially bounded in the input size.

1.2.4. Complexity Classes

We say that a decision problem Π *reduces* to some decision problem Π' if there is an efficient algorithm that computes for each instance I of Π some instance $f(I)$ of Π' such that I is a positive instance for Π if and only if $f(I)$ is a positive instance for Π'. Intuitively, this says that Π is not harder than Π' since we can decide Π efficiently if we can do so for Π'.

A *complexity class* is a class of decision problems. A problem Π (not necessarily a decision problem) is *hard* for a complexity class $\mathcal{C}$ (also called *$\mathcal{C}$-hard*) if every problem in $\mathcal{C}$ reduces to Π. The problem Π is called *complete* for $\mathcal{C}$ (also called $\mathcal{C}$-complete) if Π is additionally in $\mathcal{C}$; this means in particular that Π is a decision problem. The problems that are complete for some complexity class $\mathcal{C}$ are often considered as the "most difficult" problems in that class.

The complexity class P consists of all decision problems that admit efficient algorithms. The class NP contains all decision problems for which there is an efficient algorithm A (that is, a program of a RAM) such that for any instance I the following holds.

- If I is a positive instance then there is some s such that A accepts the pair (I, s) and the size of s is polynomially bounded in the size of I.
- If I is a negative instance then A rejects (I, s) for any s.

The class NP is a superset of P. In fact, it is widely believed that $P \neq NP$, that is, NP is supposed to be a *proper* superset of P. Papadimitriou [Pap94] provides a detailed discussion of this issue.

A common way to justify that a problem is not efficiently solvable consists in showing that it is NP-hard. This implies that there is no polynomial time algorithm for this problem unless $P = NP$.

Let $\mathcal{C}$ be a complexity class that contains $\mathcal{C}$-complete problems. A *RAM augmented by an oracle for $\mathcal{C}$* is defined like the ordinary RAM but furnished with an additional operation allowing to decide some $\mathcal{C}$-complete problem

in constant time. If $\mathcal{D}$ is a complexity class whose definition is based on the RAM model then $\mathcal{D}^{\mathcal{C}}$ denotes the complexity class that is defined like $\mathcal{D}$ with the only difference that we use a RAM augmented by an oracle for $\mathcal{C}$ instead of an ordinary RAM.

The *polynomial time hierarchy* [SU02] is a natural way to generalize the classes P and NP by means of oracle machines. In particular, we set $\Sigma_0^P := P$ and $\Sigma_{i+1}^P := NP^{\Sigma_i^P}$. It is clear that the family $\Sigma_0^P \subseteq \Sigma_1^P \subseteq \Sigma_2^P \subseteq \ldots$ of this inductively defined complexity classes actually forms a hierarchy. Moreover, it follows immediately from the definition that $\Sigma_0^P = P$ and $\Sigma_1^P = NP$. Similar to the $P \neq NP$ conjecture many researchers believe that all of the above inclusions are proper. Again, we are able to bound the complexity of a decision problem from below by using the concept of completeness. More precisely, if we could show a Σ_{i+1}^P-complete problem to be also in Σ_i^P, this would imply that $\Sigma_i^P = \Sigma_{i+1}^P = \ldots$ holds. In other words, the polynomial time hierarchy would collapse to the ith level.

1.2.5. Approximation Algorithms

For many optimization problems we do not currently know efficient algorithms. Therefore, one tries to develop *approximation algorithms* computing not always optimal but yet good solutions. An approximation algorithm A for a optimization problem Π is an efficient algorithm that computes for every instance $I \in D$ a feasible solution $s \in SOL(I)$. The *ratio* of an approximation algorithm is given by its relative deviation from the measure $m^*(I)$ of an optimal solution.

Definition 1.2.4 Let Π be an optimization problem and r be a mapping that assigns to each instance I a positive number $r(I)$. An approximation algorithm A is said to be an r-*approximate* if for all instances

$$\max\left\{\frac{m^*(I)}{m(I,A(I))}, \frac{m(I,A(I))}{m^*(I)}\right\} \leq r(I)\,.$$

1.2.6. Graph Theory

Most of the problems considered in this thesis can be described by means of *graphs*. We distinguish between *directed* and *undirected* graphs.

Directed Graphs

A *directed graph* $G = (V, R)$ is characterized by a finite, non-empty set V of *nodes* and a finite set R of *arcs*. Here, an arc is a pair (u, v) of distinct nodes. We say that u is the *source* and v is the *target* of arc (u, v).

A directed graph $G' = (V', R')$ is a *subgraph* of graph $G = (V, R)$ if $V' \subseteq V$ and $R' \subseteq R$ holds. We write $G' \subseteq G$. The graph G' is a *proper subgraph* of G if $G \neq G'$ holds additionally.

Given some directed graph $G = (V, R)$ the *out-degree* $\deg^+(u)$ of a node u is the number of arcs $(u, v) \in R$ emanating from u. The *in-degree* $\deg^-(v)$ of node v is the number of arcs $(u, v) \in R$ reaching v. The *degree* $\deg(u) := \deg^+(u) + \deg^-(u)$ of node u is the sum of the in-degree and the out-degree of u.

A path is a finite sequence $P := (v_1, \ldots, v_k)$ of pairwisely distinct nodes such that for any pair v_i, v_{i+1} of consecutive nodes there is an arc (v_i, v_{i+1}). We say that P *meets* v_i for any v_i in P. We call v_1 start node and v_k end node of P. We say also that v_k is *reachable* from v_k. We point out that a path may consist of a single node.

Two nodes are called *connected* if they are reachable from one another. A graph is *connected* if any pair of nodes in it is connected. A *(strongly) connected component* of a given graph G is a connected subgraph G' of G such that there is no connected subgraph G'' satisfying $G' \subsetneq G'' \subseteq G$.

Consider a directed graph $G = (V, R)$. Assume further that we associate with each arc $(u, v) \in R$ some *arc length* $c(u, v)$. That is, we define a function $c\colon R \to \mathbb{Q}$. Then the *length* $c(P)$ of some path $P = (v_1, \ldots, v_k)$ is the sum $\sum_{i=1}^{k-1} c(v_i, v_{i+1})$ of lengths of arcs in P. If P consists of a single node then $c(P) = 0$. Let u, v be (not necessarily distinct) nodes. Then the *distance* $d(u, v)$ induced by c is defined by the length

$$d(u, v) := \inf\{\, c(P) \mid P \text{ is a path with start node } u \text{ and end node } v \,\}$$

of some shortest path from u to v. Note that $d(u, v) = +\infty$ if there v is not reachable from u. It is easy to verify that the distance function d is a metric on V if all arc lengths are non-negative.

Undirected Graphs

An *undirected graph* $G = (V, E)$ consists of a finite and non-empty node set V and a finite set E of *edges*. An *edge* is an *unordered* pair (u, v) of distinct nodes. That is, we do not distinguish between (u, v) and (v, u). We call u and v *end nodes* of the edge and say that u and v are *incident to* (u, v).

The subgraph relation is defined analogously to the directed case. Let $G = (V, E)$ be an undirected graph and $V' \subseteq V$ be some node set. Then the *subgraph* $G[V']$ *induced by* V' is defined to be the graph (V', E') where E' contains all edges from E where both end nodes are in V'. By $G - V'$ we denote the induced subgraph $G[V - V']$.

Two distinct nodes u, v are called *adjacent to one another* if there is some edge joining u and v. We call v neighbor of u and vice versa. The set of neighbors of u is denoted by $N(u)$. The *degree* $\deg(u)$ of node u is the number $|N(u)|$ of neighbors of u.

Given some undirected graph, a *path* is a finite sequence $P := (v_1, \dots, v_k)$ of pairwisely distinct nodes such that any pair $v_i, \dots, v_{i+1}$ of consecutive nodes is joined by some edge. As for directed graphs a path may consist of a single node. We call v_1, v_k *end nodes* of P and say that they are connected.

A undirected graph is called *connected* if every pair of nodes is. A *connected component* is a connected subgraph G' such that any subgraph G'' satisfying $G' \subsetneq G''$ is *not* connected.

A *cycle* is a sequence $(v_1, v_2, \dots, v_k, v_1)$ such that $(v_1, \dots, v_k)$ is a path, $k \geq 3$, and v_k, v_1 are adjacent.

A graph that does not contain cycles is a *forest*. A *tree* is a connected forest. Given some tree, it is a well-known and basic fact that there is exactly one path for any node pair u, v with end nodes u and v [Har72]. We denote this path by $P(u, v)$.

An r-*rooted tree* is a tree $T = (V, E)$ where $r \in V$ is some distinguished node called *root*. Let T be r-rooted. Node v is called *descendant of* u if the path $P(r, v)$ meets u. If v is a descendant of u and u and v are adjacent then v is a *child* of u and u is the *parent* of v. We remark that u is a descendant of itself. We use the notation T_u to denote the subtree of T induced by the set of descendants of u. We call T_u the *subtree hanging from* u.

Let T be an tree and u, v be distinct nodes. Then $T_u(v)$ denotes the subtree hanging from v when T is considered as u-rooted.

Let $G = (V, E)$ be an undirected graph and $c \colon E \to \mathbb{Q}_0^+$ be a function assigning to each edge some non-negative length. As in the directed case we define the *length of a path* $P = (v_1, \dots, v_k)$ by $c(P) := \sum_{i=1}^{k-1} c(v_i, v_{i+1})$ and the *distance* $d(u, v)$ of two nodes u, v by

$$d(u, v) := \inf\{\, c(P) \mid P \text{ is a path with end nodes } u \text{ and } v \,\}.$$

It is easy to observe that d defines a metric on the node set.

Let u be a node. Then the *eccentricity* $\operatorname{ecc}(u)$ is given by $\max_{v \in V} d(u, v)$. The *diameter* $\operatorname{diam}(G)$ is defined by $\max_{u \in V} \operatorname{ecc}(u)$. The *radius* $\operatorname{rad}(G)$ is $\min_{u \in V} \operatorname{ecc}(u)$.

Consider an undirected graph $G = (V, E)$ with positive edge lengths $c\colon E \to \mathbb{Q}^+$. An edge of the graph can be considered as an infinite set of *points*. A point x on edge $e = (u, v)$ is specified by the distance from one of the endpoints of e, and the remaining distance is derived from the invariant $c(u, x) + c(x, v) = c(e)$. Notice that the set of points of a graph includes the set of nodes. All points which are not nodes are called *inner points*. If x is an inner point of an edge (u, v) we call u, v *neighbors* of x and set $N(x) := \{u, v\}$. In what follows, we will use G (and e) both for denoting the graph (the edge) and for denoting all of its points, as the meaning will become clear from the context. In the sense of these considerations the edge length function d is extended to a distance function $d\colon G \times G \to \mathbb{Q}_0^+$ defined on all pairs of points. The distance of a point x to a finite *point set* M is given by $d(u, M) := \inf_{m \in M} d(u, m)$.

Now assume that $w\colon V \to \mathbb{Q}_0^+$ is a function which assigns to each node v some weight $w(v)$. If $G' = (V', E')$ is a subgraph of G, then $w(G')$ denotes the weighted sum $\sum_{v \in V'} w(v)$ of the nodes of G'.

If T is a tree we make use of the notation $w_u(v) := w(T_u(v))$. The notations $T_u(v)$ and $w_u(v)$ are easily extended to the case where u or v are points, namely by temporarily adding a new node at the position of the point. The *α-ball around point* x is the point set $S_\alpha(x) := \{y \in T \mid d(x, y) \leq \alpha\}$.

Unless otherwise stated, we will use n to denote the number $|V|$ of nodes of a given graph $G = (V, E)$.

2. State of the Art and Research Objectives

This chapter embeds the topics covered by this thesis in the existing research on location, in particular on competitive and voting location. The goal is to enable the reader to classify the results developed in this work. To this end, we briefly review some of the results that are related to this work. Based on that, we formulate our main research questions.

2.1. A Brief Overview

The rigorous mathematical analysis of locational issues was initiated in the book "Der isolierte Staat" [vT26] by von Thünen in 1826. This, and also the subsequent works were mainly considered from the viewpoint of economic geography. The models employed were of descriptive nature and operated mostly on continuous spaces, for example, the Euclidean plane or the real line.

Weber was the first to consider location problems from our optimization point of view. In his seminal work "Theory of Location of Industries" [Web29] he introduced the problem of determining a point (facility) in the plane that minimizes the sum of distances to a given finite set of points (customers). This problem is called *Fermat-Weber* or also *geometric median problem*. *Competitive location* has been introduced by Hotelling in his milestone work "Stability in Competition" [Hot29] where he investigates competing providers placing facilities on a line segment.

In contrast to the aforementioned works, this thesis will not deal with continuous decision spaces. Rather, we will analyze our problems on *networks* (graphs), which is common in combinatorial optimization and operations research. This is due to the fact that in the majority of applications the facilities and customers are located at some supply network and the distances are induced by shortest paths rather than by the euclidean metric.

The first to consider such *discrete location problems* was Hakimi. He introduced the famous p-*median* and p-*center* problems on networks [Hak64].

The p-median problem asks for a p-element set of points (facilities) in a given graph such that the sum of distances from the nodes (customers) of the network is minimized. Here, the distance equals the length of some shortest path to the closest facility. The p-*center* problem seeks to minimize the maximum among those distances. A p-median solution is desirable in an application in which the costs for getting connected to some facility are borne by customers themselves. A p-median set simultaneously minimizes the *average* connection cost but also the overall financial burden for the customers as a community (social welfare). A p-center solution is appropriate in scenarios in which there is an upper bound on the cost imposed on any customer. A striking example is the location of one or more hospitals where it must be guaranteed that the access route for each potential patient is sufficiently quick. The work of Hakimi lead to the rapidly growing field of *network location*, also called *discrete* or *facility location*, which is today firmly embedded in combinatorial optimization and operations research. Mirchandani, Francis [MF90], Hamacher and Drezner [Dre09, HD01] provide comprehensive introductions and surveys to this field.

It was also Hakimi who brought competition into network location [Hak83]. He introduced the most basic competitive network location problems called (r, X_p)-medianoid and (r, p)-centroid, which are formally defined in Chapter 7 starting from page 111. In his model two providers, called *leader* and *follower*, place facilities sequentially. After both competitors have placed their facilities each customer connects to the facility that is closest to him. The aim of both competitors is to maximize their own revenue, which is measured by the total number of customers they serve. The (r, X_p)-medianoid problem is the one of the follower. It starts from the premise that the leader has already chosen some p-element set X_p for his facilities. Then, the follower chooses a set Y_r of r positions such that as many users as possible are closer to Y_r than to X_p. The leader problem, (r, p)-centroid, consists in determining a set X_p of p points such that the maximum gain of the follower is minimized. In terms of competitive location, this thesis is concerned with exactly these two basic problems (r, X_p)-medianoid and (r, p)-centroid and some of their variants.

The concept of voting location, in particular that of a Condorcet location, has been introduced by Hansen and Thisse [HT81]. Recall that in voting location the customers are regarded as voters and the potential placements for the facilities play the role of candidates being up for election. The problem asks for the placement that corresponds to the winner of the election according to some suitable social choice function. The preferences of the users are modeled by the distances in the underlying graph. More precisely,

a user u prefers a location x over location y if and only if x is closer to u than y is. Hansen et al. [HTW90] provide a comprehensive overview on voting location.

In this thesis, we will examine a *relaxed preference model* that has been introduced by Campos and Moreno [CM03]. In that model, a user is *indifferent* between two locations if the difference in the distances is less than a fixed bound. For exact definitions we refer the reader to Chapter 3 on page 39.

2.2. A Classification of the Problems Under Investigation

Hamacher and Nickel [HN98] suggest a classification scheme for location models. Eiselt and Laporte [ELT93] classify particularly competitive location problems. In the following, we briefly discuss both systems. Our aim is not only to classify the problems examined in this thesis, but also to give the reader some feeling for the degrees of freedom in the model and also the diversity in locational analysis.

It is a well-known fact that the distinction between competitive and voting location arises rather from the different motivations than from mathematical viewpoints. (At least, this is the case for the sort of problem we are going to discuss.) We will often see that one and the same problem has meaningful interpretations in both models. Thus, if we make a general statement about, say, competitive location the same will mostly hold also for the voting model and vice versa.

2.2.1. The Scheme of Hamacher and Nickel

The classification scheme of Hamacher and Nickel [HN98] encodes a location problem by a 5-tuple of the form

$$\text{Number}/\text{Space}/\text{Constraints}/\text{Customers}/\text{Objective}\,.$$

For each position some specific symbol indicates the respective choice. For example, the p-median problem on a graph is represented by the string "$N/G/\cdot/d(V,G)/\Sigma$".

Number refers to the number and type of facilities to be opened. In almost all problems that we consider a given number of points has to

be placed on a graph, that is, either at its vertices or at inner points of edges.

In Part I of this thesis we deal with single location problems where this number equals one (in the classification scheme that is symbolized by the character "1"). In Part II multiple location problems are examined, that is, the number of facilities is part of the input (symbol "N").

In the literature there are many models in which the facilities are not merely points, but rather more complex structures such as lines, paths or trees. In fact, we consider in Section 8.2.2 one such example where we place a *tree-shaped facility*. That can be symbolized by "1T".

We remark that there are also location models in which the number of facilities is not given in advance (symbol "#"). For example, the famous *facility location problem* [BK67, BK77] assigns to each node some *setup cost* and asks for a finite set of nodes minimizing the sum of setup cost and transportation cost.

Space specifies the decision space, that is, the underlying topology of the model. Here, Hamacher and Nickel distinguish between *continuous*, *network* and *discrete* location models. Continuous location deals with d-dimensional euclidean spaces $\mathbb{R}^d$ or even general Hilbert spaces. In *network location* points (that is, nodes or inner points) of a given graph form the potential locations and shortest paths induce distances. In *discrete location* the set of potential locations is finite and arbitrary distance functions may be used. We investigate network location problems. In particular, we consider general graphs and trees (symbols "G" and "T") as decision spaces.

Constraints refers to particularities of the location model such as feasible solutions or capacity constraints. We do not use such restrictions (symbol "$\cdot$"). An example where such constraints are specified is the *capacitated facility location problem* in which the total setup cost is limited by a given budget.

Customers specifies the relation between the customers (or more generally existing facilities) and the new facilities.

In our case, the customers are located at the nodes of the graph.

Concerning the set of potential locations, we distinguish between the *absolute* and the *discrete* version of a network location problem. In the absolute version, facilities may be placed at nodes and at inner points

of edges as well. The discrete version restricts the set of potential locations to the node set of the graph. Of course, the discrete version can also be viewed as a *discrete location problem* in the aforementioned sense. To denote the absolute version we may use the symbols "$d(V, G)$", or "$d(V, T)$" in the tree case. Here, $d(\cdot, \cdot)$ indicates that we are using shortest path distances, the first argument represents the set of customers and the second one the set of potential locations. Consequently, the discrete version is symbolized by "$d(V, V)$".

Objective The fifth and last position expresses the kind of *objective function* employed by the location model. In our case, we are dealing with a competitive location model, which is symbolized by "Σ_{comp}".

2.2.2. The Scheme of Eiselt, Laporte, and Thisse

Regarding competitive location the scheme of Hamacher and Nickel distinguishes only between competitive and non-competitive problems. Here the taxonomy of Eiselt, Laporte, and Thisse comes into play. Their naming scheme is especially designed for competitive location problems and thus allows for more details of the model. The scheme consists of five positions, too. The general form is

$$\text{Space}/\text{Number}/\text{Pricing}/\text{Rules}/\text{Customers}\,.$$

Space specifies the *decision space* and the symbols "N" and "T" are used to denote graphs ("networks") and trees, respectively.

Number refers to the *number of players*, which is two in our case. We remark that there are also competitive models in which there is a *free market entry*, that is, the number of players is not known in advance.

Pricing specifies the *pricing policy*. In this thesis, we assume that the competitors sell their goods or services at fixed prices. In other words, the price is no decision variable. In most cases, the competitors offer their good even at the *same* price. Therefore, the prices do not influence the competitive process at all and are thus neglected. The only model where we discuss prices is the *Stackelberg problem with parametric prices* in Section 3.4. Here, the prices of the competitors may vary but are still given as part of the input. The taxonomy uses the symbol "$\emptyset$" for non-existing or fixed prices.

For the sake of completeness we briefly outline possible policies for variable prices. Eiselt et al. distinguish between *mill pricing*, *uniform delivered pricing*, and *perfect spatial discriminatory pricing*. In all three models the customers pay the price for the good, but they differ in what way the customers bear the transportation cost.

The *mill pricing* policy models the scenario where the good is sold at the facilities and the transportation costs are imposed on the customers. In the other two models those costs are born by the respective competitor himself.

In the *uniform delivered pricing* model the prices for the goods are the same for all customers.

The *spatial discriminatory pricing* model allows the competitors to sell the good at a price that is dependent on where the respective customer is located.

Rules is concerned with the *rules of the game*. Eiselt et al. distinguish three types of *equilibria*. As common in game theory, an equilibrium is a combination of strategies pursued by the players such that no player has an incentive to change his strategy unilaterally.

The first type of equilibrium considers the case where both players move *simultaneously* and is called *Cournot-Nash equilibrium*. Situations like this occur when the competitors plan *independently* from each other. It is important that the leader-follower model we introduced informally does *not* have this property since the follower knows the placement of the leader.

The second type of equilibrium is of interest when prices are included. The *subgame perfect Nash equilibrium* assumes that the decision process of the competitors is divided into two stages. First, the locations are chosen and second, the prices are determined. This allows the competitors to calculate their prices with the knowledge of the opponent's locations.

In the third model, called *Stackelberg equilibrium*, the competitors move *sequentially*. In fact, our leader-follower model fits into this context, which is abbreviated by "V". Competitive location models that are based on Stackelberg equilibria are also called *sequential location problems*. A comprehensive overview [EL96] over that topic is provided by Eiselt and Laporte.

Customers characterizes the *behavior of the customers*. In our scenario, we assume that every customer minimizes his or her distance to the facilities, which is denoted by "D". Other models consider abstract *utility functions* which need not be based purely on distances. Examples are the sum of price and transportation cost or accessibility criteria. Furthermore, one can distinguish between *deterministic* and *probabilistic* models.

Besides what follows from the above classification we will assume the following.

- The *customers* choose to which facility they connect. This is in contrast to models in which customer allocation is part of the decision making.
- Customer preferences are based on proximity. We do not employ attraction or utility functions that respect further factors.
- The facilities are desirable, that is, customers prefer to be close to them.
- The competitors charge fixed or equal prices.
- The demand of the customers is inelastic, that is, the demand is independent from distance (essential goods).

It should also be mentioned that we do not rule out co-location, that is, we allow multiple facilities to be placed at the same location. However, the general framework of so-called *monotonic gain functions*, which we introduce in Chapter 3, allows us to modify problems so that co-location is unprofitable (strong Φ-solutions in Section 4.5) and therefore avoided by the competitors. It is well-known that permitting or prohibiting co-location can have dramatic impact.

2.3. Previous Results and Research Objectives

As we indicated in the introduction, we focus on *provable* results on efficient, exact or approximation algorithms. Thus we do not consider heuristic approaches or experimental evaluations of algorithms, for which a great deal of literature already exists; especially in the field of competitive location.

The first part of this thesis deals with single competitive and voting location problems that fall into the category $1/G/\cdot/d(V,G)/\Sigma_{\text{comp}}$ in the Hamacher-Nickel taxonomy for general location problems and

$N/2/\emptyset/V/D$ in the Eiselt-Laporte-Thisse system for competitive location. In particular, we examine the $(1,1)$-centroid problem and relaxed variants.

For the understanding of the following considerations it is crucial that the competitive $(1,1)$-centroid problem is equivalent to the Simpson problem from voting location (confer Section 3.3).

For general graphs, the fastest known algorithm for $(1,1)$-centroid and the equivalent Simpson problem is due to Hansen [HL88] and has a running time of $O(|V|^4|E|^2 \log(|V||E|) \log w(G))$. Although this running time is polynomial it is not satisfactory when handling large instances. As we shall see in Section 6.2, there are indications that it might be quite difficult to come up with "very fast" algorithms for general graphs.

Recently, Campos and Moreno introduced a *relaxed user preference model* in voting location [CM03, CM08]. It is assumed that a user prefers some facility over another one only if the former is *significantly* closer to the user. If the difference between the distances is below a given threshold the user is *indifferent* between both locations. Campos and Moreno generalize the concept of Condorcet and Simpson locations to this relaxed model and suggest polynomial time algorithms. Their algorithms are of enumerative nature and we suspect, as we do for the unrelaxed model, that it is difficult to devise substantially faster algorithms.

The relaxed preference model has originally been formulated for the Simpson and Condorcet location problems. There are, however, many other voting location problems that bear resemblances [HTW90] and that allow this concept to be applied. Moreover, the relaxed preference model can also be interpreted in the competitive location model. In practice, users show a kind of *reluctance* against new providers. It is hence natural to assume that a user is only willing to change his provider if the new provider is significantly closer. Another interpretation assumes a difference in prices [Eis92] and hence a customer changes only if the difference in distance compensates the price difference. Our first aim is thus to work out the *common properties* of voting and location problems that allow the relaxed model to be applied. This is accomplished in Chapter 3 by means of so called *monotonic gain functions*, which enable us to formulate general results and algorithms subsequently.

The aforementioned difficulty of those problems on *general* graphs motivates to investigate *simpler graph structures*. Competitive and voting location problems are often examined on *tree graphs* [MZH83, HTW86, Hak90, HTW90, Eis92, EL93, EL96, GP03]. Tree graphs appear quite often in practical applications as they constitute the "cheapest" graphs connecting a given set of nodes (for example, one may think of backbone networks in com-

puter networks). Moreover, it has turned out that the absence of cycles in the network often dramatically reduces the complexity of sequential location problems, which are rather "messy" on general graphs. In fact it is known [HTW86] that the $(1,1)$-centroid and the Condorcet problem can be solved in linear time on trees by means of Goldman's algorithm [Gol71] for computing a median of a tree.

This and other strong positive results for the unrelaxed model suggest to investigate also the relaxed model or even monotonic gain functions on tree graphs.

Eiselt [Eis92] investigates the Stackelberg problem with parametric prices on trees (which fits into the relaxed model and is essentially equivalent to the relaxed Simpson problem, confer Section 3.4). He shows that, having the prices fixed, it is generally beneficial for each competitor to act as the *follower*. This phenomenon is called the *first entry paradox*.

Garcia and Pelegrin [GP03] analyze the same problem from an algorithmic viewpoint and provide an $O(n^3 \log n)$ time algorithm for trees. However, such a running time might be considered as rather impractical.

Therefore, in Chapters 4 and 5, we investigate the existence of *fast* algorithms for general monotonic gain functions on trees. The super-cubic algorithm of Garcia and Pelegrin [GP03] for the Stackelberg problem with parametric prices (that is, for the relaxed model) on trees and the linear time algorithm of Goldman [Gol71] for the unrelaxed case may serve as "yardsticks".

In the second part of this thesis, we analyze *multiple* competitive and voting location problems. These problems can be classified by the string $n/G/\cdot/d(V,G)/\Sigma_{comp}$ in the Hamacher-Nickel system and again by $N/2/\emptyset/V/D$ in the Eiselt-Laporte-Thisse system. In particular, we examine the (r,p)-centroid and the (r,X_p)-medianoid problem. There is a recent overview [SSD07] of results on these problems. A survey on *sequential location* [EL96] is provided by Eiselt and Laporte.

In the multiple location case, we are mainly concerned with complexity theoretic and approximability questions. The focus is *not* on particularly fast algorithms. Rather, we examine the existence of polynomial time (approximation) algorithms and hardness results, respectively.

Both problems, (r,X_p)-medianoid and (r,p)-centroid, are known to be NP-hard on general graphs [Hak83]. The (r,p)-centroid problem is not approximable within a constant factor [Hak90]. These results, however, leave some important questions open. While the decision problem of (r,X_p)-medianoid is in NP, the same is *not* clear for the more complicated (r,p)-centroid. In fact, Hakimi [Hak90] conjectures the (r,p)-centroid problem

to be "exceedingly difficult". Moreover, the non-approximability result for (r, p)-centroid does not rule out algorithms with a good input-dependent ratio such as $O(\log n)$. The approximability of the follower problem is open, too.

These questions are tackled in Chapter 7. First, we will discuss the exact complexity of the (r, p)-centroid problem on general graphs giving justification to Hakimi's conjecture that it is extremely difficult. Then we examine the existence of approximation algorithms with good input dependent performances and the approximability of the follower problem.

We remark that Campos and Moreno [CM08] investigate the p-Simpson problem, which is a special case of the discrete (r, p)-centroid problem. They provide exact exponential time algorithms by means of linear integer programming.

The negative results for general graphs motivate the investigation of the problems on trees as in the single location case.

In fact, the (r, X_p)-medianoid problem is known to be efficiently solvable on trees in $O(rn^2)$ time [Tam96]. Tamir's algorithm also solves the p-median problem on trees in time $O(pn^2)$, which is the fastest algorithm known for this problem.

Whether the *leader problem* is polynomially solvable on trees is a long-standing open question and has first been asked by Hakimi [Hak90]. The question is repeated in the overview of sequential location problems [EL96] of Eiselt et al. and Benati [Ben00].

In Chapter 9 we answer Hakimi's open question concerning the complexity of (r, p)-centroid on trees. We shed also some light on the complexity and approximability of the problem on paths and on the case where the leader locates one facility and the follower locates multiple facilities.

Let's return to the follower problem on trees. In the single location case ($(1, X_1)$-medianoid) the problem is trivially solvable in linear time. Is such an improvement in running time still possible if the number of facilities of the leader, whose locations are part of the input and thus no decision variables, is unrestricted? Applying Tamir's algorithm [Tam96] to this case yields a quadratic running time, which is also achieved by a naive approach (see Section 8.2). Kim et al. [KLTW96] suggest an algorithm that solves the problem in $O(n \log^2 n)$ time. Their approach applies even to a much more general scenario where a tree-shaped facility is to be located on a tree.

In Chapter 8 we show how to improve the algorithm of Kim et al. even in the general case of a tree-shaped facility.

Part I.

Single Location

3. Monotonic Gain Functions

The aim of this chapter is to introduce a novel framework, called *monotonic gain function* (MGF), which constitutes a generalization and unification of all particular location problems investigated within the first part of this thesis. The problems we are going to model with the help of MGFs are single location problems, that is, each provider always places one single facility or server. In particular we examine single competitive (Section 3.2) and single voting location problems (Section 3.3) on graphs.

As we have seen in the introduction, competitive location addresses the situation where two or more competing providers offer the same type of goods or service to a finite set of customers located at the nodes of the graph. Different competitors may supply their product from different sites. Each customer is assumed to bear transportation costs thereby incurred and hence to select the provider whose facility is closest to him. In a nutshell: Providers compete for location but not for quality or prices of goods. In our particular problem setting we are given two suppliers, leader and follower, each of which places exactly one facility and tries to serve as much demand as possible. Here two crucial points must be stressed: First, competitors make their decisions in a sequential manner, that is, one after the other. And second, we rule out any kind of cooperation.

In contrast to the non-cooperative behavior of suppliers in competitive location, voting location implies one central and *social* planning instance. Here the process of decision making may be understood as a collective action which yields a compromise respecting the wishes and preferences of the single users as far as possible.

Surprisingly, it turns out that both of these seemingly contrary paradigms lead to the same complex of mathematical problems [Hak90, HTW90]. In both settings, we deal with the preferences of users basing on the distance to potential locations of a facility. Moreover, we shall see that the locational decision of the leader in competitive location can also be viewed as a particularly fair compromise in voting location: There is no alternative placement that is preferred by a large portion of the users.

From a mathematical point of view, the problems we shall examine differ not so much in whether they belong to competitive or voting location as in

how they deal with the users' preferences.

The first question is on *what basis* a user prefers one location over the other. In the simplest case a user chooses a facility x if it is closer to him than its opponent y is, and he is *undecided* if and only if both distances are equal (unrelaxed user preference). But it might also be argued that users *disregard* very small differences in distance, that is, they only affect significantly better sites (relaxed user preference).

The other question is how undecided users *react*. For example, we could assume that they always select the leader facility or alternatively, that they divide their demand equally between leader and follower. We shall see that those degrees of freedom bring about a variety of location criteria and problems.

This chapter is organized as follows. We first discuss in Sections 3.1, 3.2 and 3.3 the standard unrelaxed user preference model and proceed in Section 3.4 with an *additive indifference relation* [CM03] and a *mill pricing model* [GP03] as extensions of the standard model. The reader will be faced with a multitude of problem definitions, which often bear resemblances, sometimes differ merely in details or are patently reformulations of each other arising from different application domains. The reason for this "tangle" is a lack of systematics in literature for this special sort of sequential location problems. As a way out we finally introduce the aforementioned framework of monotonic gain functions in section 3.5, which enables us to formulate general results and algorithms in subsequent chapters. It is to be hoped that the notion of monotonic gain functions will result in a clearer, more systematic and deeper understanding and maybe also in new insights into the investigated complex of problems.

3.1. User Preference

In the sequel we are given an undirected graph $G = (V, E)$ with positive edge lengths $c\colon E \to \mathbb{Q}^+$ and non-negative node weights $w\colon V \to \mathbb{Q}_0^+$. We assume that the competitors may place their facilities or servers on points of the graph, that is, on nodes or inner points of edges. The users, on the other hand, are located only at the nodes of the graph. The weight $w(u)$ of node u represents the *demand* of the user located at u. Both competitors sell the same good at the same price. The good is *essential* for users, that is, each users is supposed to meet his full demand. Transportation costs, modeled by edge lengths, are born by the users themselves. Thus it is reasonable to assume that

- users always use shortest paths and
- connect to the closest server or facility.

As described in Section 1.2 the edge lengths c induce a distance function $d\colon G \times G \to \mathbb{Q}_0^+$ on the graph G viewed as a point set. Recall that $d(x,y)$ denotes the length of a shortest path between points $x, y \in G$ according to c. This complies with the above assumption of users moving along shortest paths.

The user behavior is formalized as follows.

Definition 3.1.1 (User preference) Let $x, y \in G$ be points of a graph $G = (V, E)$, that is, either nodes or inner points of edges of G. A user $u \in V$ *prefers* point x over point y, denoted by $x \prec_u y$, if

$$d(u,x) < d(u,y)\,.$$

The user u *is indifferent*, $x \sim_u y$, if $d(u,x) = d(u,y)$.

In our competitive scenario we assume that if user u prefers x over y then the full demand $w(u)$ of u is served by x. Thus it is favorable for both competitors to be preferred by as many users as possible.

We use the following notation: The set of users preferring x over y is denoted by $U(x \prec y) := \{u \in V \mid x \prec_u y\}$, and its weight by $w(x \prec y) := w(U(x \prec y))$. Analogously we define the set $U(x \sim y) := \{u \in V \mid x \sim_u y\}$ of undecided users and its weight $w(x \sim y) := w(U(x \sim y))$. Now we are able to formulate the main problems under investigation in this piece of work.

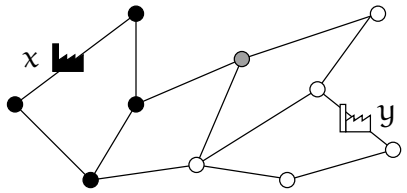

FIG. 3.1.: Two competing facilities x and y are placed at the middle of edges. Every vertex hosts a user with unit demand. All edges have unit length. Nodes preferring x are colored black. White nodes prefer y. The gray node is undecided. We obtain $w(x \prec y) = 4$, $w(x \sim y) = 1$ and $w(y \prec x) = 5$.

3.2. Competitive Location Problems

In this and in the following section we are going to introduce and discuss competitive and voting location models based on the standard (unrelaxed) user preference. We consciously avoided to start with the more general relaxed user preference model, which will be considered in Section 3.4.

Centroid and x-Medianoid

Centroid and x-medianoid belong to the most prominent representatives of sequential location problems [EL96], which constitute an important field in competitive location. They are characterized by competitors locating their servers one after the other. In our case we are given two competitors, leader and follower, placing one facility each.

The user preference model in Definition 3.1.1 describes the static situation where the competing providers have already placed their facilities. Now we are going to specify how the competitors make decisions and what rules they obey. Since it is easier to motivate let us first assume that the leader has already chosen his position and that the follower is trying to determine a location maximizing his own revenue (follower problem).

Definition 3.2.1 (x-Medianoid [Hak83]) Let $x \in G$ be a leader placement and let

$$\Gamma(x) := \max_{y \in G} w(y \prec x)$$

be the maximum influence any follower placement can gain over the fixed leader placement x. An *absolute x-medianoid* of the graph is any point $y \in G$ where $w(y \prec x) = \Gamma(x)$ is attained.

For an illustrative example of an absolute x-medianoid, see Figure 3.2.

Surely choosing an x-medianoid as server location may be considered as a rational follower strategy. It complies with our assumption of selfish players. On the other hand this makes the follower's reaction predictable for the leader: Once he has chosen a position x the follower will achieve a gain of $\Gamma(x)$; and clearly he has no option but to serve the remaining demand $w(G) - \Gamma(x)$. Thus maximizing the gain of the leader simply amounts to minimizing that of the follower.

Definition 3.2.2 (Centroid [Hak83]) Let

$$\Gamma^* := \min_{x \in G} \Gamma(x)\,.$$

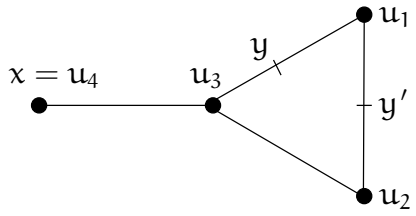

FIG. 3.2.: Assume that the leader has located his facility x at node u_4. If the follower selects y', he serves users $U(y' \prec x) = \{u_1, u_2\}$, that is, $w(y' \prec x) = 2$. But y is a better choice since $w(y \prec x) = 3$. In fact y is an x-medianoid and so are all points $\neq x$ not lying at edge (u_1, u_2). We obtain $\Gamma(x) = 3$.

An *absolute centroid* of the graph is any point $x \in G$ where $\Gamma(x) = \Gamma^*$ is attained.

For an illustrative example of an absolute centroid see Figure 3.3.

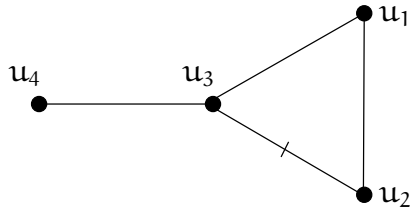

FIG. 3.3.: Consider the same graph as in Figure 3.2. Assume, however, that the leader has not located his facility yet. We have seen that $\Gamma(u_4) = 3$. The same holds for inner points of edge (u_4, u_3). If x is located at the middle of edge (u_3, u_2) then the follower achieves $\Gamma(x) = 3$, too, by placing at adjacent node u_3. Similarly we have $\Gamma(x) = 3$ for any inner point $x \in G$. For node u_1 we obtain $\Gamma(u_1) = 3$ if the follower will select the midpoint of edge (u_3, u_2). Analogously $\Gamma(u_2) = 3$. On the other hand, if x is placed at u_3 the follower has no chance of serving more than two users. Hence $\Gamma^* = \Gamma(u_3) = 2$ and u_3 is the unique centroid of the graph.

The notions *discrete* x*-medianoid* and *discrete centroid* are defined similarly, with the server points restricted to nodes $x, y \in V$ rather than points.

Centroid and x-medianoid have originally been introduced in Hakimi's seminal paper [Hak83] for *multiple* competitive location problems (confer Section 7.1). In the original terminology of Hakimi centroid or x-medianoid would be called $(1, 1)$-centroid or $(1, X_1)$-medianoid respectively. For the

time being we focus on single leader and follower placements and therefore use the simplified terms.

Stackelberg Location

The reader may have noticed that the users treat leader and follower asymmetrically in the above definitions. If a user is indifferent, he is implicitly assumed to be served by the leader. This could be justified by, for example, an additional effort needed for reconnecting from leader to follower or by implying users of conservative nature.

But we may also suppose more active users treating leader and follower as fully equivalent providers. We model this behavior by ascribing one half of the demand $w(x \sim y)$ of the undecided users to each competitor. This leads immediately to the notion of *Stackelberg locations*:

Definition 3.2.3 (Stackelberg [HTW90]) Let $x \in G$ be a leader placement and

$$\Sigma(x) := \max_{y \in G} w(y \prec x) + \frac{1}{2} w(y \sim x).$$

Then an *absolute Stackelberg location* is a point that minimizes $\Sigma(\cdot)$. We write

$$\Sigma^* := \min_{x \in G} \Sigma(x)\,.$$

As in the case of centroid and x-medianoid we introduce a *discrete* variant of Stackelberg location by restricting the set of potential locations for leader and follower to the node set.

Nash Solution

Assume that leader and follower have already chosen their locations according to the Stackelberg model. In this situation we may ask whether both competitors are actually content with the resulting locational pattern in the sense that neither of them has any incentive to change his position unilaterally. While this condition is true for the follower it does not need to hold for the leader position (confer Figure 3.4).

A pair (x, y) of points meeting the above stability criterion will be called a *Nash solution* named after and inspired by the eponymous equilibrium in game theory.

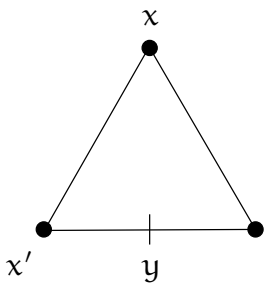

FIG. 3.4.: Points x and y are optimal Stackelberg leader and follower positions, respectively. By switching from x to x' the leader can increase his payoff.

Definition 3.2.4 (Nash solution [HTW90]) A pair (x, y) of points in a graph is called an (absolute) *Nash solution* if no party can increase its payoff by moving to another location where the payoff is given as $w(y \prec x) + \frac{1}{2}w(y \sim x)$. Formally, the pair (x, y) must satisfy

$$\begin{aligned} w(y \prec x) + \frac{1}{2}w(y \sim x) &\geq w(y' \prec x) + \frac{1}{2}w(y' \sim x) \quad \text{and} \\ w(x \prec y) + \frac{1}{2}w(x \sim y) &\geq w(x' \prec y) + \frac{1}{2}w(x' \sim y) \quad \text{for all } x', y'. \end{aligned}$$

The reader familiar with the general concept of Nash *equilibria* may wonder why we define Nash *solutions* just for Stackelberg payoff $w(y \prec x) + \frac{1}{2}w(y \sim x)$ but not for the simpler centroid payoff $w(y \prec x)$. (The original notion of a Nash equilibrium does not impose any restriction on the payoff function.) The reason is, that there is no temporal element in the definition of a Nash solution: Basically, it is not specified in which order the locations x and y have been selected or if they have been placed even simultaneously. So there is no reasonable argument to include any asymmetry in payoff into this definition as it were the case for centroid and x-medianoid (confer motivation of Stackelberg model).

3.3. Voting Location Problems

Condorcet and Simpson

The notion of a Condorcet location is inspired by the eponymous criterion known from voting theory. A candidate of an election is called a *Condorcet winner*, if there is no opponent which is preferred by more than one half of the voters' weight. In order to apply this criterion to our location setting, we

identify users with voters and potential locations with the candidates being up for the election. We suppose that a voter (user) u prefers a candidate (location) x over y if and only if $d(u,x) < d(u,y)$, that is, $x \prec_u y$. Using our standard notation we are able to define the Condorcet location formally.

Definition 3.3.1 (Condorcet location [HT81]) Let $x, y \in G$ be points. We say that y *dominates* x if $w(y \prec x) > \frac{1}{2}w(G)$. Moreover x is called a *Condorcet location* if there is no point dominating x.

A main drawback of this concept is, that a Condorcet location does not always exist. Consider for example an equilateral triangle with one user located at each vertex (confer Figure 3.4). Then any vertex is dominated by each of the inner points of the opposite edge. On the other hand each inner point of an edge is dominated by its adjacent vertices.

In order to overcome these difficulties Campos and Moreno [CM03] introduce the notion of a γ-*Condorcet location*, where $0 \leq \gamma \leq 1$. Here the requirement of the Condorcet criterion is relaxed. A point y dominates x only if $w(y \prec x) > \gamma \cdot w(G)$. Since each point is a 1-Condorcet location we are now able to always guarantee the existence of an γ admitting a γ-Condorcet location.

Definition 3.3.2 (γ-Condorcet location [CM03]) Let $0 \leq \gamma \leq 1$. A point x is called a γ-Condorcet location if there is no y such that $w(y \prec x) > \gamma \cdot w(G)$.

It appears reasonable to understand the parameter γ as a degree of *stability* of the solution. More precisely we consider locations with smaller γ as more stable than those with a larger one. Finally we are interested in the *most stable location* admitted by the problem instance which leads to the following definition.

Definition 3.3.3 (Simpson location [HL88]) Let γ^* be the smallest γ such that a γ-Condorcet location exists. Then each γ^*-Condorcet location is called a *Simpson location*.

In this definition, the minimum γ^* always exists since there are only finitely many user parties (subsets of the node set).

Clearly a Condorcet location exists if and only if $\gamma^* \leq \frac{1}{2}$. In this case each Simpson location is also a Condorcet location.

Plurality and Security

In the definition of the Condorcet location x, only voters against x are taken into account while all undecided users are actually treated as if they voted for x. This can pretend a high stability of a solution which does not actually exist. The concept of plurality location is a way to deal with this situation as only decided users are respected.

Definition 3.3.4 (Plurality location [WM81]) A point x is called a *plurality location* if there is no point y such that $w(y \prec x) > w(x \prec y)$.

Clearly, each plurality location is also a Condorcet location. Thus a plurality location does not need to exist either. An obvious way to relax the plurality criterion is to minimize the difference between the weights of the follower and the leader party. This leads to the following definition.

Definition 3.3.5 (Security location [Sla75]) Let x be a point and

$$\Delta(x) := \max_{y \in G}(w(y \prec x) - w(x \prec y)) .$$

Then a *security location* is a point that minimizes $\Delta(\cdot)$. We write

$$\Delta^* := \min_{x \in G} \Delta(x) .$$

Clearly a plurality location exists if and only if $\Delta^* \leq 0$. In this case each security location is also a plurality location.

Comparison Between Voting and Competitive Location Models

As mentioned in the introduction of this chapter, from a mathematical point of view the concepts of competitive and voting location bear resemblances to each other or are even equivalent. In particular it is immediately clear that the notion of a Simpson location and a centroid are mathematically equivalent. If we take into account that the user parties always form a partition of the node set of the input graph we obtain the relation $w(x \prec y) + w(y \prec x) + w(x \sim y) = w(G)$. Plugging this into Definition 3.3.5 we obtain

$$\Delta(x) = \max_{y \in G} 2(w(y \prec x) + \frac{1}{2} w(y \sim x)) - w(G) = 2 \cdot \Sigma(x) - w(G) .$$

Since $w(G)$ is constant over all feasible solutions each security location is also a Stackelberg location and vice versa.

Not quite so obvious is the following relation between plurality and Nash locations.

Theorem 3.3.6 (Nash and plurality [HTW90]) *The set of Nash solutions is equal to the set of pairs of plurality locations.* □

3.4. Relaxed User Preferences

Relaxed User Preferences in Voting Location

The relaxed user preference model was introduced by Sloss [Slo78] in the context of voting theory. Campos and Moreno [CM08, CM03] tied in with that model and investigated relaxed variants of γ-Condorcet and Simpson locations for which they provided enumerative polynomial-time algorithms.

Relaxed user preferences in voting location can be motivated by the observation that the standard preference model does not take into account *how much* but only *if* a user prefers one location over the other. Therefore even very small differences in distance can prevent a location from being a Condorcet or plurality solution. Clearly this property is rather undesirable not least because in reality small distance differences would be ignored by users or may arise from measurement errors.

By contrast the relaxed preference model is based on the assumption that a user only prefers one of the locations x or y if their distance difference exceeds a given threshold α; otherwise the user is undecided.

Definition 3.4.1 (Relaxed user preference [Slo78, CM03]) Let $\alpha \geq 0$. A user u *α-prefers* point x over point y if

$$d(u, x) < d(u, y) - \alpha\,. \tag{3.1}$$

The user u *is α-undecided* if $|d(u, x) - d(u, y)| \leq \alpha$.

We use the following notation: The set of users α-preferring x over y is denoted by $U_\alpha(x \prec y)$ and called the *x-party*. We use $w_\alpha(x \prec y) := w(U_\alpha(x \prec y))$ to denote its weight. Similar notations apply to the set $U_\alpha(x \sim y)$ of undecided users and the y-party $U_\alpha(y \prec x)$. As in many cases the indifference parameter α becomes clear from the context we often suppress α in the notation. We simply write $U(x \prec y), w(x \prec y), U(x \sim y)$ or $w(x \sim y)$ and use the terms "to prefer" and "undecided". For an illustration of relaxed user preferences confer Figure 3.5.

A straightforward application of the relaxed preference model to the voting location problems introduced above leads to the following definitions.

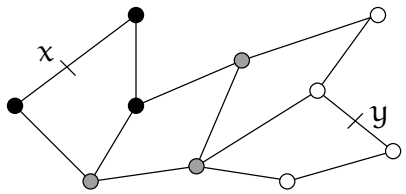

FIG. 3.5.: Preferences under the relaxed model with $\alpha := 1$. In comparison to the unrelaxed case (cf. Figure 3.1) the set $U_\alpha(x \sim y)$ of undecided users grows. We have $w_\alpha(x \prec y) = 3$ and $w_\alpha(y \prec x) = 4$.

Definition 3.4.2 ((α, γ)-Condorcet [CM03]) A location x is an (α, γ)-*Condorcet location* if there is no $y \in G$ such that $w_\alpha(y \prec x) > \gamma \cdot w(G)$.

For given α and γ let $C(\alpha, \gamma)$ denote the set of all (α, γ)-Condorcet locations. Campos and Moreno [CM03] prove some *basic properties* of the (α, γ)-Condorcet criterion for which we introduce the following terms:

Monotonicity The family $(C(\alpha, \gamma))_{\alpha,\gamma}$ of (α, γ)-Condorcet sets is inclusion-wise increasing with respect to both parameters α and γ.

γ-Minimality For any $\alpha \geq 0$ there is a smallest γ, denoted by $\gamma^*(\alpha)$, such that $C(\alpha, \gamma)$ is not empty.

α-Minimality For any $0 \leq \gamma \leq 1$ there is a smallest α, denoted by $\alpha^*(\gamma)$, such that $C(\alpha, \gamma)$ is not empty.

Those observations allow us to formulate the following definitions.

Definition 3.4.3 (α-Simpson [CM03]) Let $\alpha \geq 0$ be given. An *α-Simpson location* is an $(\alpha, \gamma^*(\alpha))$-Condorcet location.

Figure 3.6 shows an example where the set of unrelaxed Simpson locations and the set of (relaxed) 1-Simpson locations are disjoint.

Definition 3.4.4 (γ-Tolerant and Efficient Condorcet [CM03]) Let $0 \leq \gamma \leq 1$. Then any $(\alpha^*(\gamma), \gamma)$-Condorcet location is called a *γ-tolerant Condorcet location*. An (α, γ)-Condorcet location is called *efficient* if $\alpha = \alpha^*(\gamma)$ and $\gamma = \gamma^*(\alpha)$.

The relaxed pendants of plurality and security location are defined as follows.

Definition 3.4.5 (α-Plurality and α-Security [CM08]) A point x is an *α-plurality location* if $w_\alpha(y \prec x) \leq w_\alpha(x \prec y)$ for all $y \in G$. A location is called an *α-Security location* if it minimizes $\Delta_\alpha(z) := \max_{y \in G}(w_\alpha(y \prec z) - w_\alpha(z \prec y))$ among all $z \in G$.

FIG. 3.6.: Consider the α-Simpson problem for $\alpha := 1$. Unmarked nodes and edges are unit weighted. The center node is the sole α-Simpson location. Thus $\gamma^*(\alpha) = 0$. If weight Ω is sufficiently large then the leftmost node is the single (unrelaxed) Simpson solution. We obtain $\gamma^*(0) = 2$. This demonstrates that relaxed and unrelaxed solutions can form disjoint location sets.

Relaxed User Preferences in Competitive Location

The relaxed user preference model can also be interpreted in the setting of competitive location: Once the leader has placed his facility and serves the whole demand users are supposed to exhibit a reluctance against a new provider (the follower) entering the market. A user is only willing to reconnect to the follower if his location is significantly better, that is, closer to the user. This behavior is also modeled by means of relaxed user preferences. The following definitions are straightforward generalizations of their unrelaxed counterparts.

Definition 3.4.6 (α-x-Medianoid) Let $x \in G$ be a leader placement and

$$\Gamma_\alpha(x) := \max_{y \in G} w_\alpha(y \prec x).$$

An *absolute α-x-medianoid* of the graph is any point $y \in G$ where $w_\alpha(y \prec x) = \Gamma_\alpha(x)$ is attained.

Definition 3.4.7 (α-Centroid) Let

$$\Gamma_\alpha^* := \min_{x \in G} \Gamma_\alpha(x)\,.$$

An *absolute α-centroid* of the graph is any point $x \in G$ where $\Gamma_\alpha(x) = \Gamma_\alpha^*$ is attained.

Definition 3.4.8 (α-Stackelberg) Let $x \in G$ be a leader placement and

$$\Sigma_\alpha(x) := \max_{y \in G} w_\alpha(y \prec x) + \frac{1}{2} w_\alpha(y \sim x)\,.$$

Then an *absolute α-Stackelberg location* is a point that minimizes $\Sigma_\alpha(\cdot)$. We write

$$\Sigma_\alpha^* := \min_{x \in G} \Sigma_\alpha(x)\,.$$

The relaxation parameter α can also be interpreted in a scenario known as *mill pricing*: here each competitor sells the good at a fixed price specific to this competitor and the individual user costs are determined by the sum of the user-server distance and the service price of the server to which the user is connected. Input parameters $p, q \in \mathbb{Q}_0^+$ specify the *prices* when a user connects to the leader or the follower: The total serving cost of user u when connecting to the leader x or the follower y, respectively, are defined by $L(u) := p + d(u, x)$ or $F(u) := q + d(u, y)$. Each user u decides to connect to one of the servers according to the following rule set [GP03].

1. If $L(u) \neq F(u)$, then user u will choose the server with smaller total serving costs.
2. If $L(u) = F(u)$ and $d(u, x) \neq d(u, y)$, then user u chooses the nearest server.
3. If $L(u) = F(u)$ and $d(u, x) = d(u, y)$, then user u splits his demand $w(u)$: the leader serves $(1 - f(u)) \cdot w(u)$ and the follower serves $f(u) \cdot w(u)$, where $f\colon V \to [0, 1]$ is an additional function specified by the input instance.

Again leader and follower move sequentially and try to maximize the demand they serve. Garcia and Pelegrin [GP03] call the resulting optimization problem *Stackelberg location with parametric prices*. They state that for the case $p > q$ the problem becomes trivial: the follower can place right at the leader's position and thus gain all users.

Assume that the prices satisfy $p < q$. As in this case rule 3 never applies, the user preference rules are greatly simplified: a user u prefers the follower y over leader x if and only if $d(u, x) - d(u, y) \geq \alpha$ where $\alpha := q - p$. Except for the equality in this condition this yields the α-centroid problem. Note that the problem is *not* equivalent to the α-Stackelberg location problem (confer Definition 3.4.8) due to the different treatment of undecided users.

Now let $p = q$: If $d(u, x) = d(u, y)$ then Rule 3 applies, that is, user u assigns a portion $f(u) \cdot w(u)$ of his demand to the follower and the rest $(1 - f(u) \cdot w(u))$ to the leader. Thus the demand served by the follower can be expressed by $w(y \prec x) + (f \cdot w)(U(y \sim x))$ which has some similarities with the unrelaxed preference model (confer Section 4.6).

3.5. Generalization to Monotonic Gain Functions

Monotonic Gain Functions and Φ-Solutions

If we set the indifference α to zero we arrive at the unrelaxed preference model. Therefore all relaxed problems defined in Section 3.4 are generalizations of their unrelaxed equivalents in Sections 3.2 and 3.3. In the sequel we are going to develop a framework, called monotonic gain functions, which generalizes all relaxed (and therefore also unrelaxed) competitive and voting location models introduced so far. As we are always dealing with the more general relaxed model we will omit the parameter α from all notations (for the sake of simpler notations).

A *gain function* $\Phi\colon G \times G \to \mathbb{Q}$ maps a point pair (y, x) to the value $\Phi(y, x)$ which measures in some sense the influence of a follower point y after leader point x has already been placed into the graph. In order to emphasize the intuitive meaning of $\Phi(y, x)$ we use the notation $\Phi(y \prec x) := \Phi(y, x)$. We require that any gain function have a finite image.

Given a gain function, the notions *absolute score* and *absolute solution* are defined as follows:

Definition 3.5.1 (Absolute Φ-score and Φ-solution) For any gain function Φ, the *absolute* Φ*-score* of a leader point x is defined as

$$\Phi(x) := \max_{y \in G} \Phi(y \prec x)\,.$$

The *absolute* Φ*-score* of a graph is defined as $\Phi^* := \min_{x \in G} \Phi(x)$. An *absolute* Φ*-solution* of a graph is a point x with $\Phi(x) = \Phi^*$.

We notice that the maxima and minima in this definition always exist since Φ has a finite image.

One might imagine the Φ-score $\Phi(x)$ of point x as the *instability* of x since it measures the maximum influence the follower can gain in presence of x. In this sense the Φ-score Φ^* of a graph is the minimum instability achievable by the leader. Briefly speaking, the smaller Φ^* the better placements are possible for the leader.

Many economical models known from the literature see stability as a mere 0–1 concept. In contrast, the notion of stability induced by monotonic gain functions is of gradual nature. We mention that there are other stability concepts with this property such as the stability continuum introduced by Bhadury and Eiselt [BE95].

Definition 3.5.2 (Discrete Φ-score and Φ-solution) *Discrete Φ-score and solution* are defined as in 3.5.1, except that x and y are restricted to *nodes* of the input graph.

Definition 3.5.3 (Witness) A *witness* of leader x is a point y where

$$\Phi(y \prec x) = \Phi(x).$$

If $\Phi(y \prec x)$ measures the instability of the leader x when confronted with follower y then there is a natural monotonicity requirement: Φ should not decrease if we move the points x, y in such a way that the follower influence $w(y \prec x)$ increases while the leader influence $w(x \prec y)$ decreases.

Definition 3.5.4 (Monotonic gain function) A gain function $\Phi(y \prec x)$ is called *monotonic*, if there is a function $\varphi\colon \mathbb{Q}_0^+ \times \mathbb{Q}_0^+ \to \mathbb{Q}$ such that

1. $\Phi(y \prec x) = \varphi(w(y \prec x), w(x \prec y))$ for all points $x, y \in G$
2. φ is monotonically increasing in the first parameter and monotonically decreasing in the second parameter
3. φ can be evaluated in constant time.

Note that the last requirement can be also achieved after a preprocessing step which itself may need more than constant time. This may be necessary if the evaluation of $\varphi(\cdot,\cdot)$ comprises graph-dependent parameters like the total weight $w(G)$ of all users.

Once again, we emphasize that we are always operating with *relaxed user preferences*. Thus $w(y \prec x)$ is a shorthand notation for $w_\alpha(y \prec x)$ etc. We remark that this does not lead to conflicts with notation used for the unrelaxed model in Section 3.2 and 3.3 as the latter is obtained as a special case when setting $\alpha := 0$. In this sense we are *extending* our notation instead of redefining it.

Observation 3.5.5 (Simpson, security, Stackelberg) *The monotonic gain functions*

$$\begin{aligned}\Gamma(y \prec x) &:= w(y \prec x)\\ \Delta(y \prec x) &:= w(y \prec x) - w(x \prec y)\\ \Sigma(y \prec x) &:= w(y \prec x) + \tfrac{1}{2}w(x \sim y)\end{aligned}$$

induce the α-Simpson *problem, the* α-security *problem, and the* α-Stackelberg *problem, respectively.*

Observation 3.5.6 (Further examples) *Let $\lambda \in [0,1]$. Then the following gain functions are monotonic:*

$$\begin{aligned} R(y \prec x) &:= \frac{w(y \prec x)}{w(x \prec y)} \\ N(y \prec x) &:= w(y \prec x) + w(y \sim x) \\ \Psi(y \prec x) &:= w(y \prec x) + \lambda \cdot w(y \sim x) \end{aligned}$$

The gain function R is may be considered as a variation of security location where difference is replaced by ratio (the case $w(x \prec y) = 0$ is to be handled appropriately). While for Simpson location it is assumed that users are conservative, the gain function N implies novelty oriented users: If a user is indifferent between the competitors he always connects to the new facility, that is, to the follower. Unfortunately, this brings about a disproportional disadvantage of the leader: If the follower chooses the same location as the leader does he serves the whole demand $w(G)$. Hence all $x \in G$ are N-solutions. Finally the MGF Ψ is a generalization of Γ, Σ and N: Setting $\lambda := 0$, $\lambda := \frac{1}{2}$ or $\lambda := 1$ we obtain the MGF $\Psi = \Gamma$, $\Psi = \Sigma$ and $\Psi = N$, respectively. We call Ψ *generalized Stackelberg function*.

Leader Independent MGFs

It is noteworthy that the Simpson MGF Γ differs from all other above MGFs in that it only depends on the weight of the follower party. We call such MGFs *leader independent.*

Definition 3.5.7 (Leader independent MGF) A MGF Φ induced by function φ is called *leader independent* if φ only depends on the first parameter.

An arbitrary leader independent MGF Φ may be considered as a coarsening of the Simpson MGF Γ since each Simpson location is also a Φ-solution. Moreover if φ is *strictly increasing* in the first parameter then Φ becomes equivalent to the Simpson MGF.

Tolerant and Efficient Φ-Solutions

Recall that a point x is an (α, γ)-Condorcet location if $\Gamma(x) \leq \gamma \cdot w(G)$. The following definition is a straightforward generalization of this concept.

Definition 3.5.8 (φ_0-bounded Φ-solution) Let Φ be an MGF and α an indifference threshold. Let further $\varphi_0 \in \mathbb{Q}$. A point x is called a *φ_0-bounded Φ-solution* if $\Phi(x) \leq \varphi_0$. The set of all φ_0-bounded Φ-solutions for parameter α is denoted by $S(\alpha, \varphi_0)$.

Note that if $\varphi_0 < \varphi(0,0)$ then any point is dominated by itself and $S(\alpha, \varphi_0)$ is empty. Therefore we assume in the sequel that $\varphi_0 \geq \varphi(0,0)$.

It is not hard to prove that the basic properties of the (α,γ)-Condorcet sets $C(\alpha,\gamma)$ carry over to *leader independent MGFs*.

Lemma 3.5.9 (Monotonicity) *Let Φ be a MGF. Then the family of sets $(S(\alpha, \varphi_0))_{\alpha,\varphi_0}$ is monotonically increasing with respect to φ_0. If Φ is leader independent this holds for parameter α, too.*

In what follows we use the notation $\varphi(w_y) := \varphi(w_y, 0)$ for leader independent MGFs.

Proof (Lemma 3.5.9). The monotonicity with respect to φ_0 is trivial.

To prove the monotonicity with respect to α consider two fixed points $x, y \in G$ and a user u preferring y over x, that is, $d(u,x) - d(u,y) > \alpha$. If we increase the indifference parameter α this inequality may become untrue. In other words u may leave follower party $U(y \prec x)$ and enter the party $U(y \sim x)$ of undecided users. Clearly, no users enters $U(y \prec x)$ and hence $w(y \prec x)$ cannot increase. Therefore if we increase α then $w(y \prec x)$ and thus also $\varphi(w(y \prec x))$ cannot increase as well. Here, we exploit the monotonicity and leader independency of $\varphi(\cdot,\cdot)$. □

Figure 3.7 demonstrates that for leader dependent MGFs the monotonicity with respect to α does not need to hold.

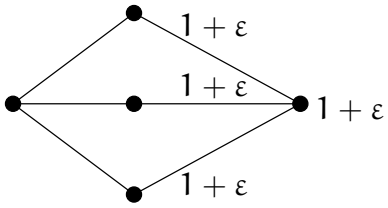

FIG. 3.7.: Consider the security MGF Δ. The leftmost node is in $S(0,0)$ but not in $S(\varepsilon,0)$ for $\varepsilon > 0$ since it is then dominated by the rightmost node. Unmarked nodes and edges are unit weighted.

The analogue to γ-minimality, called *Φ-minimality*, holds also for general MGFs and follows immediately from the existence of the minimum Φ^* in Definition 3.5.1.

Observation 3.5.10 (Φ-Minimality) *Let Φ be a MGF and $\alpha \geq 0$ an indifference threshold. Then there is a smallest φ_0, denoted by $\Phi^*(\alpha)$, such that $S(\alpha, \varphi_0)$ is non-empty.* □

Clearly, $\Phi^*(\alpha)$ equals the Φ-score Φ^* but we sometimes use the former notation to emphasize its dependency from α.

Recall that $S(\alpha, \varphi_0)$ is empty for any $\varphi_0 < \varphi(0,0)$ and $\alpha \geq 0$. On the other hand, if $\varphi_0 \geq \varphi(0,0)$ and $\alpha \geq \operatorname{diam}(G)$ then $S(\alpha, \varphi_0)$ contains all points of G. Together with the monotonicity property we conclude.

Observation 3.5.11 (α-Minimality) *Let Φ be a MGF and $\varphi_0 \geq \varphi(0,0)$. Then there is a smallest $\alpha \geq 0$, denoted by $\alpha^*(\varphi_0)$, with non-empty $S(\alpha, \varphi_0)$.* □

The validity of the basic properties for leader independent MGFs enables us to generalize the notions of γ-tolerant and efficient Condorcet locations.

Definition 3.5.12 (φ_0-Tolerant Φ-solution) Let Φ be a leader independent MGF and $\varphi_0 \geq \varphi(0)$. Then each $x \in S(\alpha^*(\varphi_0), \varphi_0)$ is called a φ_0-*tolerant* Φ-*solution*.

Definition 3.5.13 (Efficient Φ-solution) Let Φ be a leader independent MGF and $\varphi_0, \alpha \in \mathbb{Q}$. Then $x \in S(\alpha, \varphi_0)$ is called an *efficient* Φ-*solution*, if $\alpha = \alpha^*(\varphi_0)$ and $\varphi_0 = \Phi^*(\alpha)$.

3.6. Concluding Remarks

In this chapter we have introduced a new model called monontonic gain functions which generalizes several problems from the area of competitive and voting location such as Stackelberg, $(1,1)$-centroid, Simpson, Condorcet, Security and Plurality. Monotonic gain functions emerge from two basic properties inherent in all those problems: First, they are based on distance-based user preferences. Second, the underlying optimization process is two-stage (leader and follower in competitive location, candidate and opposition in the voting scenario). Without any difficulties, monotonic gain functions cope with the relaxed user preference model introduced by Campos and Moreno [CM03]. Besides the most obvious notion a Φ-solution on a given graph for some monotonic gain function Φ, we have also introduced related concepts such as φ_0-bounded, φ_0-tolerant and efficient Φ-solutions.

The following two chapters will be devoted to algorithmic aspects related with monotonic gain functions. Specifically, we will develop fast algorithms for tree graphs. As we shall see, the framework of monotonic gain functions

helps formulate general algorithms and results for competitive and voting location problems.

4. Computing a Φ-Solution of a Tree

4.1. Introduction

As mentioned in Chapter 2 there are three main reasons why we focus on tree graphs: First of all, the problems under consideration seem to be quite "unapproachable" on general graphs, which motivates to study simpler graph structures. Second, tree graphs appear quite often in practical applications. And third, a whole series of positive results and fast algorithms already exist for the unrelaxed variants of those problems on a tree. It is therefore an interesting question how far such results carry over to the more general relaxed user preference model or even monotonic gain functions.

4.1.1. General Graphs

Hansen and Labbe [HL88] developed a polynomial-time algorithm for the (unrelaxed) absolute centroid problem on a general graph $G = (V, E)$ with integral node weights w. The running time of this quite involved algorithm is $O(|V|^4|E|^2 \log(|V||E|) \log w(G))$. Although centroid is therefore efficiently solvable the above asymptotic running time involves large exponents and might be considered as rather impracticable.

Even if we restrict ourselves to the *discrete* centroid (discrete Simpson problem) the situation does not become much easier. Campos and Moreno [CM03] suggested an enumerative algorithm for computing the Condorcet set $C(\alpha, \gamma)$ in cubic time. We will argue later in Section 6.2 that computing the set $C(\alpha, \gamma)$ is as least as hard as solving the *vector maximization problem* [KLP75], which is a problem of fundamental importance and has already been investigated thoroughly in literature. From our result it follows that an algorithm which is essentially faster than the enumerative approach of Campos and Moreno [CM03] would lead to an algorithm for the vector maximization problem that supersedes the fastest algorithms known today.

4.1.2. Trees

There is a great deal of literature investigating sequential and voting location problems on tree graphs [Sla75, WM81, HTW90, Eis92, GP03]. One of the most impressive results is that on trees *unrelaxed* (but *not* relaxed, confer Figure 4.1) centroid, Condorcet, Simpson, plurality and security all coincide with the set of medians [HTW90]. One could say that central planning, voting and competition lead to the same locational outcome in this particular setting. On the algorithmic side this leads to a linear time algorithm for all these problems using the well known algorithm of Goldman [Gol71] for computing the set of medians of a tree.

It is natural to ask whether this is true for all MGFs. Indeed we can prove that the following weakening of the above observation holds for all MGFs.

Theorem 4.1.1 *For the* unrelaxed *user preference $\alpha = 0$ and for any monotonic gain function Φ, any absolute weighted median of a tree is also a Φ-solution.*

Proof. Let x be an arbitrary point of a tree T. We assume w. l. o. g. that x is a node for otherwise we could create a zero weighted node at x. By $T - x$ we denote the forest obtained by deleting x and all edges incident to x from T.

Now let $y \neq x$ be any point. It is then obvious that the y-party $U(y \prec x)$ is always contained in one single connected component C_y of $T - x$. Moreover the x-party $U(x \prec y)$ contains at least all nodes of T that are not in C_y.

It is a basic fact known from location theory that a point $x \in T$ is a median of T if and only if all connected components of $T - x$ have weight at most $\frac{1}{2}w(T)$ [Gol71]. Thus if m is a weighted median and $y \neq m$ then $w(y \prec m) \leq \frac{1}{2}w(T) \leq w(m \prec y)$ (here we exploit that $\alpha = 0$). Hence, if y is a witness of m and $x \neq m$ is an alternative point, we obtain:

$$\begin{aligned}
\Phi(m) &= \Phi(y \prec m) \\
&= \varphi(w(y \prec m), w(m \prec y)) \\
&\leq \varphi\left(\tfrac{1}{2}w(T), \tfrac{1}{2}w(T)\right) \\
&\leq \varphi(w(m \prec x), w(x \prec m)) \\
&= \Phi(m \prec x) \\
&\leq \Phi(x)\,.
\end{aligned}$$

It remains to consider the case where m itself is the only witness of m. Then $\Phi(m) = \varphi(0,0) = \Phi(x \prec x) \leq \Phi(x)$. □

We conclude that an unrelaxed Φ-solution can always be computed in linear time.

Clearly if we set $\Phi :\equiv 0$ each point of the tree is a Φ-solution. Hence there can be unrelaxed Φ-solutions that are not a median.

Unfortunately Theorem 4.1.1 does not hold for the *relaxed* user preference: We see in Figure 4.1 an example which demonstrates that in the case $\alpha > 0$ the sets of all medians, α-Simpson and (strong) α-Security solutions may actually be pairwise disjoint. Hence finding a relaxed Φ-solution of a tree

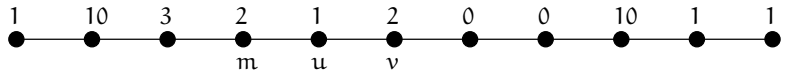

FIG. 4.1.: Example where median $\{m\}$, relaxed strong[1] security solution $\{u\}$ and relaxed Simpson solution $\{v\}$ are unique and pairwise different. All edges have unit length and the indifference parameter is $\alpha = 4$.

seems to be a nontrivial task.

Algorithms for relaxed sequential location problems on trees have been investigated by Garcia and Pelegrin [GP03]. They investigate the problem of determining all Stackelberg solutions with parametric prices (akin to α-Simpson problem) of a tree. The running time of their quite complicated algorithm is $O(n^3 \log n)$. Recall that we use n to denote the number $|V|$ of vertices of the input graph $G = (V, E)$, which is a tree in this case.

Eiselt [Eis92] examines a competitive location problem that is similar to the relaxed Nash location problem. Also sequential variants are considered but not from an algorithmic viewpoint.

Apart from the equivalence between medians and unrelaxed sequential location on trees there is another interesting special case relating MGFs with central planning: Assume that we increase α until it attains the radius $\mathrm{rad}(G)$ of G. Then each α-Simpson location becomes an absolute *center* of the tree and vice versa.

We conclude from the above observations that any algorithm which is able to compute a Φ-solution for arbitrary Φ and $\alpha \geq 0$ must be so general that it "includes" especially the computation of medians, centers and also of fixed-priced Stackelberg solutions of trees. Computing a median or a center of a tree are already nontrivial problems but can be solved in linear time by the well known algorithms of Goldman [Gol71] and Handler [Han73], respectively. Indeed we are able to generalize both results: In this chapter,

[1] A strong solution excludes the slightly pathological case of all users being undecided. See Definition 4.5.1.

we will present a *linear time algorithm* computing a relaxed Φ-solution (and thus also the score Φ^*) for an arbitrary MGF Φ. Hence our algorithm may be understood as a general procedure for solving not only voting and competitive location but also certain planning problems (like median and center) on a tree in linear and hence optimal time. As a byproduct, we shall see in Chapter 5 that our algorithm enables us to compute the entire set of Stackelberg solutions with parametric prices in $O(n)$ time which is significantly faster than Garcia and Pelegrin's $O(n^3 \log n)$ algorithm.

4.2. Computing the Absolute Φ-Score of a Point

We first investigate the problem of computing the relaxed Φ-score of a leader point x in a tree. Consider the case where two points x, y are given. By definition the three sets $U(x \prec y)$, $U(x \sim y)$, and $U(y \prec x)$ form a partition of the tree T. We show in the following lemma that each of the three sets is actually a connected subtree (in the case $d(x,y) \leq \alpha$ the sets $U(x \prec y)$ and $U(y \prec x)$ are empty as all users are undecided). Confer Figure 4.2 for an illustration.

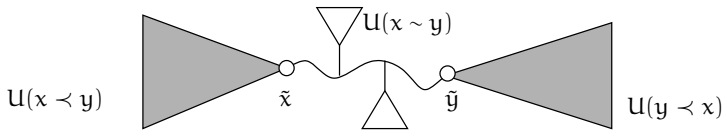

FIG. 4.2.: Any tree is partitioned in x-party, y-party, and the set of undecided users. Each of the three parties induces a subtree unless it is empty. Notice that x, y do not necessarily fall into their respective party if they are inner points of edges.

Lemma 4.2.1 (Front nodes) *Let $x, y \in T$ be points such that $d(x,y) > \alpha$ and let P be the shortest node terminated path containing $P(x,y)$ as a subpath. Then there are nodes $\tilde{x}, \tilde{y}$ on P such that $U(x \prec y) = T_{\tilde{y}}(\tilde{x})$ and $U(y \prec x) = T_{\tilde{x}}(\tilde{y})$.*

The nodes $\tilde{x}, \tilde{y}$ are called *front nodes*. These nodes are of special importance since they completely characterize the above mentioned partition of the tree, from which the desired value $\Phi(y \prec x)$ can be derived. We remark that $\tilde{x} = x$ or $\tilde{y} = y$ can occur. It is even possible that $x \notin T_{\tilde{y}}(\tilde{x})$ or $y \notin T_{\tilde{x}}(\tilde{y})$ (confer Figure 4.3).

Proof (of Lemma 4.2.1). Let x_0 be the end node of the path P that is closer to x than to y. For any node z, let the *projection* of z be the node $\bar{z}$ on P where

$d(z, \bar{z})$ is minimal. Since $d(z, x) - d(z, y) = d(\bar{z}, x) - d(\bar{z}, y)$ it follows that the preferences $x \prec_z y$ and $x \prec_{\bar{z}} y$ are identical. Choose node $\tilde{x} \in P$ such that $x \prec_{\tilde{x}} y$ and $d(x_0, \tilde{x})$ is maximal. Then the nodes on path P that prefer x are exactly those on the subpath $P(x_0, \tilde{x})$. If $v \in T$ is an arbitrary node, then $x \prec_v y$ if and only if its projection $\bar{v}$ prefers x, that is, $\bar{v} \in P(x_0, \tilde{x})$. This is equivalent with $v \in T_y(\tilde{x}) = T_{\tilde{y}}(\tilde{x})$. The situation for $\tilde{y}$ is symmetric. □

Consider a leader x, a follower y and the path $P(x, y)$ with endpoints x and y. Now assume that y moves along $P(x, y)$ approaching the leader. As long as $d(x, y) > \alpha$, the movement of y can affect the weights of the parties in two ways: On the one hand, the influence $w(y \prec x)$ of the follower can increase because it retains all users in $T_x(y)$ and comes closer to all other users. On the other hand, the influence $w(x \prec y)$ of the leader can decrease since y approaches all users of the leader party. By the monotonicity property, Φ increases with decreasing $d(x, y)$. We formulate our considerations in the following lemma:

Lemma 4.2.2 *Let x, x', y, y' be (not necessarily distinct) points such that x and y' lie on the path $P(x', y)$ and $d(x', y') > \alpha$ and $d(x, y) > \alpha$ holds. Then*

$$\Phi(y' \prec x') \geq \Phi(y \prec x)\,.$$

Proof. Let's start with the leader-follower configuration (x', y) and assume that we move the follower towards x' until we reach the configuration (x', y'). According to our above considerations we have that $w(y' \prec x') \geq w(y \prec x')$ and $w(x' \prec y') \leq w(x' \prec y)$. Hence $\Phi(y' \prec x') \geq \Phi(y \prec x')$ by the monotonicity property of Φ.

Now assume that we, starting from configuration (x', y), shift the *leader* towards y until we reach the configuration (x, y). By symmetry we conclude that $w(x \prec y) \geq w(x' \prec y)$ and $w(y \prec x) \leq w(y \prec x')$ holds. Hence $\Phi(y \prec x') \geq \Phi(y \prec x)$, which completes the proof. □

Let's go back to our above thought experiment of a follower y approaching the leader x. As long as the follower remains outside the α-ball $S_\alpha(x)$ it increases $\Phi(y \prec x)$. However, when y arrives at the border of $S_\alpha(x)$ all users become suddenly undecided since then $|d(u, x) - d(u, y)| \leq \alpha$ for all users u. Hence a local maximum of $\Phi(y \prec x)$ is attained when the follower has reached a distance $d(x, y)$ "slightly larger than α".

To formalize the expression "slightly larger than α", observe that any user within the α-ball $S_\alpha(x)$ around x either belongs to the x-party or is undecided. The same holds for the α-ball around y. Now assume that y has been

moved so close to x that all inner nodes on the path $P(x,y)$ are part of the intersection $S_\alpha(x) \cap S_\alpha(y)$. Then it is clear that the follower can not increase his gain by further approaching x, since all inner nodes on $P(x,y)$ are and remain undecided.

The above condition is met if there is no node u on the path $P(x,y)$ such that $0 < d(x,u) < d(x,y) - \alpha$ or $\alpha < d(x,u) < d(x,y)$.

Now we define

$$\varepsilon(x) := \min\big(\{\, d(x,u),\ d(x,u) - \alpha \mid u \in V \,\} \cap \mathbb{Q}^+\big)$$

and observe that any point y at distance $\alpha + \varepsilon(x)$ from x satisfies the above condition.

Definition 4.2.3 (α-neighborhood) Let x be a leader point. Any point y with $d(x,y) = \alpha + \varepsilon(x)$ is called an *α-neighbor* of x. The set of all α-neighbors of x is denoted by $N_\alpha(x)$.

We may imagine an α-neighbor of some point x as a point whose distance to x is "slightly larger" than α or that is "infinitesimally close" to the α-ball $S_\alpha(x)$.

We remark that the auxiliary definition of $\varepsilon(x)$ helps us construct a well-defined and finite set of α-neighbors for the purpose of theoretical investigations. Later in Lemma 4.2.5 we will give a simple characterization of front nodes that avoids an explicit calculation of α-neighbors.

From the above observations we can conclude that, in order to compute $\Phi(x)$, it suffices to consider nodes in the α-neighborhood of x:

Lemma 4.2.4 (Witness) *For each leader x there is a witness $y \in N_\alpha(x) \cup \{x\}$.*

Proof. Let y' be a follower node such that $\Phi(x) = \Phi(y' \prec x)$. If $d(x,y') \le \alpha$, then all nodes are undecided, that is, $U(y' \sim x) = V$. Since also $U(x \sim x) = V$ we can conclude $\Phi(x \prec x) = \Phi(y' \prec x) = \Phi(x)$ and hence x itself is a witness.

If $d(x,y') > \alpha$, then consider the α-neighbor y of x on path $P(x,y')$. According to Lemma 4.2.2 we have that $\Phi(y \prec x) \ge \Phi(y' \prec x)$. Since $\Phi(y' \prec x)$ was maximal this shows $\Phi(y \prec x) = \Phi(x)$. Note that if $P(x,y')$ does not contain an α-neighbor then there is an α-neighbor y such that $P(x,y)$ contains y'. Due to the definition of an α-neighbor, $\Phi(y' \prec x)$ cannot be greater than $\Phi(y \prec x)$. □

Consider a pair (x,y) of leader and follower. Once the pair $(\tilde{x},\tilde{y})$ of associated front nodes is known, the value $\Phi(y \prec x) = \varphi(w_{\tilde{x}}(\tilde{y}), w_{\tilde{y}}(\tilde{x}))$

can be easily computed as it only depends on the weights of the subtrees hanging from the front nodes. For the sake of easier presentation we write $\varphi(y, x) := \varphi(w_x(y), w_y(x))$ in the sequel. If we assume that the weights of all possible subtrees $\{ T_u(v), T_v(u) \mid (u, v) \in E \}$ have already been computed as a preprocessing step (which needs two DFS traversals and therefore takes $O(n)$ time) the function value $\Phi(y \prec x) = \varphi(\tilde{y}, \tilde{x})$ can be evaluated in $O(1)$ time when the front nodes $(\tilde{x}, \tilde{y})$ are known. The following lemma provides a tool for constructing front nodes (see Figure 4.3 for an illustration).

Lemma 4.2.5 (Characterization of front nodes) *Let x be a point and y be an α-neighbor of x. Then the front nodes $\tilde{x}, \tilde{y}$ are characterized as: The path $P(\tilde{x}, \tilde{y})$ is the shortest node-terminated path that contains $P(x, y)$ as a subpath.*

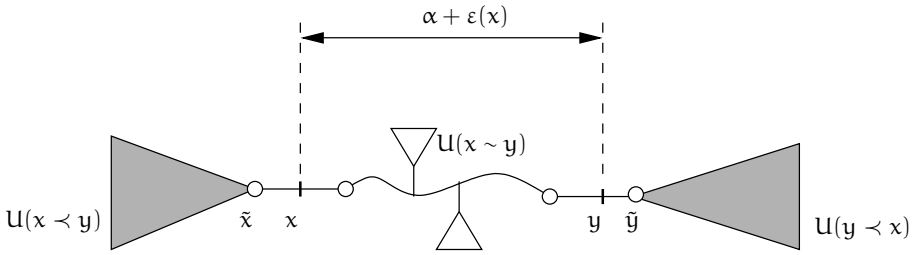

FIG. 4.3.: Characterization of front nodes $\tilde{x}$ and $\tilde{y}$ if y is an α-neighbor of x.

Proof. Consider an arbitrary node z and its projection $\bar{z}$ onto $P(\tilde{x}, \tilde{y})$. If $\bar{z}$ is an inner node on $P(\tilde{x}, \tilde{y})$, it is undecided since y is an α-neighbor of x. Otherwise, z is part of one of the subtrees $T_{\tilde{x}}(\tilde{y})$ or $T_{\tilde{y}}(\tilde{x})$ and clearly prefers x or y, respectively. □

We now argue that during the process of determining front nodes it is not necessary to actually execute a computation of α-neighbors using the function $\varepsilon(x)$ outlined above. Observe that for a given point x, any edge (u, v) contains an α-neighbor of x if and only if $d(x, u) \leq \alpha < d(x, v)$; if that condition is met, v is the desired front node. Obviously the set of all front nodes can be enumerated by a simple depth-first-search traversal starting at x.

Theorem 4.2.6 (Absolute Φ-score of a point) *For any MGF Φ on a tree, the absolute score $\Phi(x)$ of a given point x can be computed in time $O(n)$.* □

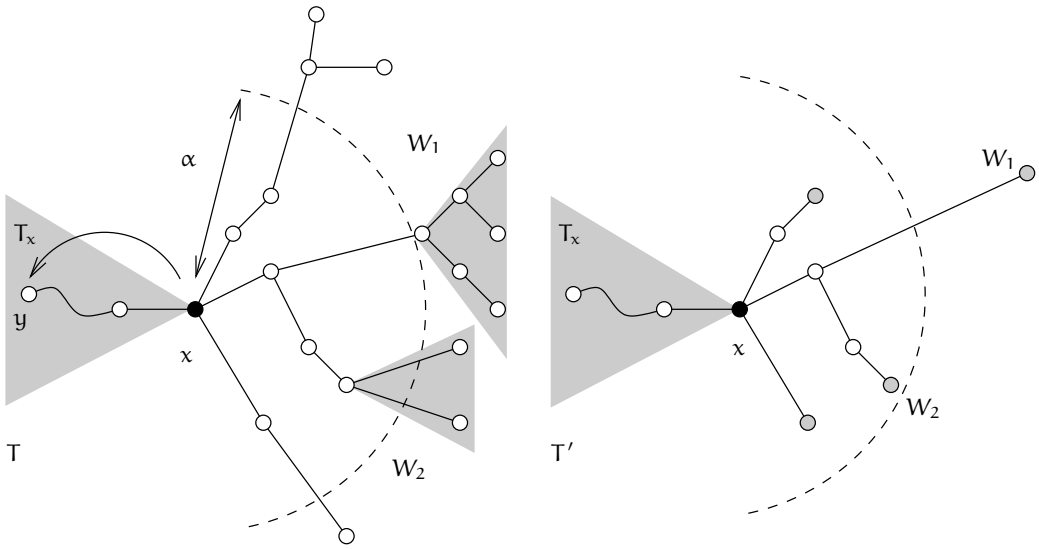

FIG. 4.4.: Example for collapsing subtrees. If y is a witness of x then the subtree T_x contains at least one optimum. Moreover, if we simplify T to T', we can guarantee that any optimum of T' that lies in T_x is also an optimum in the original tree T.

4.3. Computing an Absolute Φ-Solution

In this section we develop an algorithm that computes for any monotonic gain function Φ on a tree an absolute Φ-solution in linear time. The rough idea of the algorithm is to maintain a sparsified version of the input tree that is divided into a so-called *leader tree* and two *follower trees*. The leader tree is always a subtree of the input tree and is guaranteed to contain a Φ-solution, that is, an optimal placement for the leader. The leader tree and the follower trees are iteratively shrunken during the execution of the algorithm. When the number of remaining nodes has reached $O(1)$, the iteration stops and a solution can be found and output in constant time.

4.3.1. An Introductory Example

Consider the example depicted in Figure 4.4 showing a point x with witness y of distance $d(x, y) > \alpha$. First, we claim that the subtree T_x towards witness y (that is, the subtree induced by the node set $(V - T_y(x)) \cup x$) contains at least one optimum. This follows from the fact that for any point $x' \notin T_x$ we have that $\Phi(x') \geq \Phi(y \prec x') \geq \Phi(y \prec x) = \Phi(x)$ by Lemma 4.2.2. Therefore, either x itself is optimal or all optima must lie in T_x. This obser-

vation will be called "guide rule" since it shows the direction to the optimal leader placements and allows us to narrow down our further search to T_x. In fact, the guide rule will be used in our algorithms to reduce the size of the leader tree (confer procedure HALVELEADER in Section 4.3.3).

Although we can restrict our search to T_x, we can, of course, not simply delete the nodes outside T_x since they still could serve as potential locations for the follower ("follower tree"). However, we can considerably simplify the part of the tree that lies outside T_x *and* outside the α-ball $S_\alpha(x)$.

To this end consider an arbitrary point $x' \in T_x$ and assume that its witness y' does not belong to T_x.

If y' is inside $S_\alpha(x)$, say, the root node of the gray colored subtree of weight W_2 in Figure 4.4, then the actual structure of the tree outside S_α is not crucial for computing the Φ-score of x'. It would thus be sufficient to retain only one node of that subtree and to assign to it weight W_2 as it is done in the tree T' on the right side of the picture. To put it more algorithmically, we may merge all nodes outside $S_\alpha(x)$ with the closest node inside.

But what if the witness y' lies outside $S_\alpha(x)$? To allow for this case we could keep one single representative node whose weight is that of the heaviest subtree outside $S_\alpha(x)$. In our example this is W_1.

To summarize, a situation like the one in Figure 4.4 allows us to discard all nodes (except one) that are contained neither in T_x nor in $S_\alpha(x)$. Our transformation guarantees that any optimum of T' that lies in T_x is also an optimum of T. Thus it suffices to solve the problem on T' rather than to search the whole tree T. We will use this idea in the procedure DISCARDFAR (confer Section 4.3.6) to reduce the size of the tree iteratively.

We learn from this example that it is possible to maintain a sparsified version of the input tree that essentially keeps aggregated weights in certain nodes organized in so-called follower trees. This way we can guarantee the invariant that any point with minimum Φ-score in the current leader tree has also minimum Φ-score in the initial tree and both scores are identical.

4.3.2. Terminal Trees

Let T be a tree and u, v two designated nodes called *terminal nodes*. The *two terminal subtree (2TS)* T_{uv} *induced by* u *and* v is the maximal subtree of T containing both u and v as leaves. The actual data structure employed by our algorithm is called a *terminal tree* and defined as follows (see Figure 4.5 for an illustration):

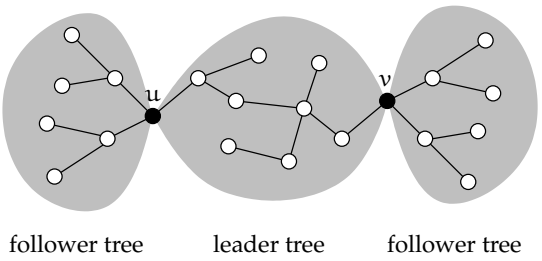

FIG. 4.5.: Example of a terminal tree with terminals u, v.

Definition 4.3.1 (Terminal tree) A *terminal tree* is a tree $T = (V, E)$ with two distinguished terminal nodes $u, v \in V$. The 2TS T_{uv} induced by u and v is called the *leader tree* of T. The trees $F_u := T_v(u)$ and $F_v := T_u(v)$ are called *follower trees* of T.

Our algorithms maintains a terminal tree that is shrunken iteratively until the number of nodes reaches a constant. The linear running time of our algorithm is achieved since first, each iteration takes a time linear in the size of the current terminal tree and second, this size is guaranteed to decrease by a constant factor after a constant number of iterations. To this end we employ two procedures (confer Figure 4.6): Procedure HALVELEADER reduces the size of the leader tree by identifying new terminals and thus moving some nodes from the leader tree into the follower trees. In contrast, procedure HALVEFOLLOWER discards nodes from a follower tree, maintaining the weights accordingly. Note that only HALVEFOLLOWER actually reduces the size of the current terminal tree.

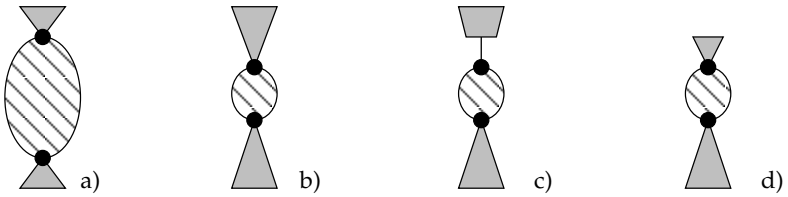

FIG. 4.6.: a) current terminal tree, follower trees shaded gray. b) terminal tree after HALVELEADER. c) terminal tree after HALVEFOLLOWER (discard near). d) terminal tree after HALVEFOLLOWER (discard far).

4.3.3. Sparsifying the Leader Tree

The first ingredient of our algorithm is the subroutine HALVELEADER that decreases the size of the leader tree by almost half of its nodes. Before we

describe this procedure, we formulate the aforementioned guide rule.

To this end let x be a node in a tree T and $y \neq x$ be some point of T. The *y-component of T at x*, denoted by $C_x(y)$, is the subtree induced by the node set $(V - T_y(x)) \cup x$. For an illustration confer Figure 4.7.

Lemma 4.3.2 (Guide rule) *Let T be a tree, let $x \in T$ be a node of T, and let $y \in T$ be a witness of x such that $y \notin S_\alpha(x)$. If there is an $x' \in T$ with $\Phi(x') < \Phi(x)$ then $x' \in C_x(y)$. On the other hand, if x is a witness of itself then $\Phi(x) = \Phi^*$.*

Proof. Let $x' \in T_y(x)$ be an arbitrary point. Then

$$\Phi(x') \geq \Phi(y \prec x') \geq \Phi(y \prec x) = \Phi(x)$$

by Lemma 4.2.2.

For the second claim observe that if x is a witness of itself then all nodes are undecided. Thus $\Phi(x) = \Phi(x \prec x) = \varphi(0,0)$. Moreover, $\varphi(0,0) = \Phi(x' \prec x') \leq \Phi(x')$ for all other points x', hence $\Phi(x)$ is optimal. □

As a consequence of the guide rule, observe that if we have identified a witness y of a point x then we can narrow down the search for optima to the component $C_x(y)$.

An *x-split of T* is the collection $\{ C_x(y) \mid y \in N(x) \}$ of components induced by the neighborhood of x (confer Figure 4.7). For a finite set S of points of tree T, the *S-split* is the result of iteratively splitting at all $x \in S$. Observe that any S-split forms a partition of the edge set of the tree T, and nodes of S always appear as leaves in the resulting subtrees. We call a node set S a *valid split* if none of the resulting subtrees contains more than two nodes of S (confer Figure 4.7).

A valid split set is helpful for the development of a divide and conquer algorithm: given a current leader tree and a valid split set, any of the generated subtrees can serve as a new leader tree since it is surrounded by at most two split nodes which can be used to define a new 2TS. If such a subtree contains only one split node then choose an arbitrary leaf as the second terminal to obtain a valid 2TS.

The following lemma shows that an appropriate valid split set can be formed with the help of unweighted medians.

Lemma 4.3.3 (Valid subdivision by unweighted median) *Given a tree with terminals u, v, let n' be the number of nodes in the 2TS T_{uv}. Now let m be the unweighted median of T_{uv} and m' be the projection of m onto path $P(u,v)$. Then the set $\{m, m', u, v\}$ is a valid split set and each 2TS induced has at most $n'/2 + 1$ nodes. This split can be computed in $O(n')$ time.*

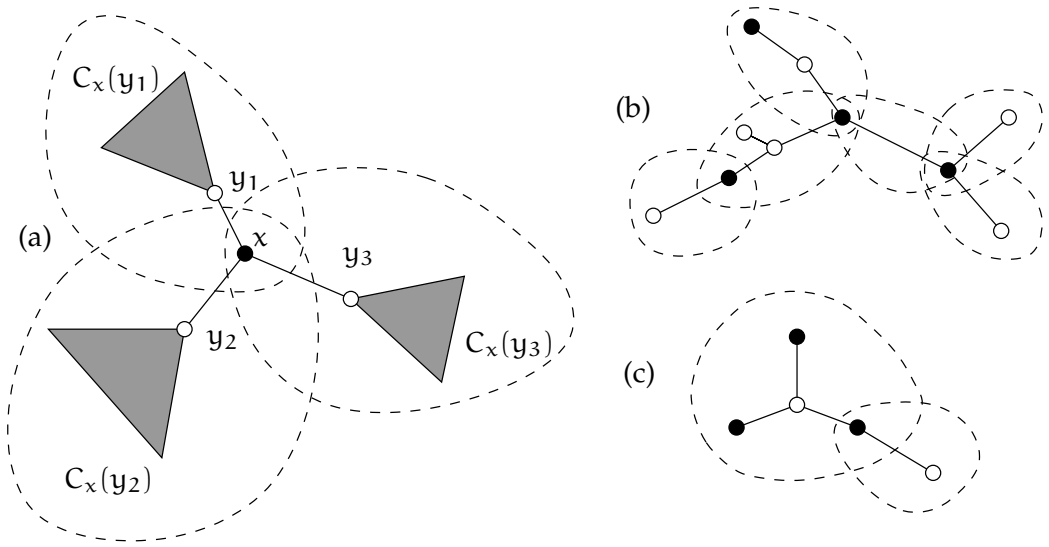

FIG. 4.7.: (a) x-split (b) valid split (c) invalid split; split nodes are marked black.

Proof. The claim that $\{m, m', u, v\}$ is a valid split set follows easily from distinguishing the cases that either m lies on path $P(u, v)$ or that it does not. Confer also Figure 4.8.

The other claim follows from the fact that each connected component of the forest $T_{uv} - m$ has at most $n'/2$ nodes due to the median property of m. The unweighted median can be computed in linear time employing Goldman's algorithm [Gol71]. □

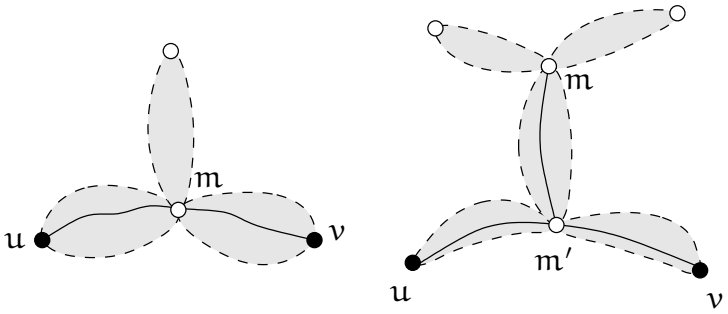

FIG. 4.8.: The two cases in the subdivision of a 2TS.

Let m and m' be as in Lemma 4.3.3. Suppose that y, y' are witnesses and α-neighbors of m, m', respectively. From the guide rule we deduce that a point x with minimum Φ-score in L (that is, $\Phi(x) = \min_{z \in L} \Phi(z)$) must lie in

the intersection of the components, that is, in the subtree

$$L' := C_m(y) \cap C_{m'}(y') \cap L\,. \tag{4.1}$$

This is exactly one of the 2TSs generated by the valid split set and can then serve as a new leader tree (confer Figure 4.10).

We summarize our considerations in the following algorithm:

---

1 **input** terminal tree T with terminals u, v and leader tree L
2 let m be an unweighted median of L
3 let m' be the projection of m on path $P(u, v)$ ($m' = m$ allowed)
4 $\{u, v, m, m'\}$ is a valid set of split nodes
and generates a partition of L into subtrees
5 determine a witness $y \in N_\alpha(m) \cup \{m\}$ of m
6 and a witness $y' \in N_\alpha(m') \cup \{m'\}$ of m'
7 **if** both witnesses are α-neighbors **then**
8 use the guide rule and Equation (4.1)
to determine the relevant subtree L'
9 and **output** this as the new terminal tree
10 **else**
11 we have $\Phi(z) = \Phi(z \prec z)$ for $z = m$ or $z = m'$
12 **output** optimum z and **stop** main algorithm

FIG. 4.9.: Algorithm HALVELEADER

Lemma 4.3.4 (Halve leader tree) *Let T be a terminal tree with leader tree L. Then we can identify in time $O(|T|)$ a new leader tree $L' \subseteq L$ of size $|L'| \leq \frac{1}{2}|L| + 1$ which satisfies $\min_{x \in L'} \Phi(x) = \min_{x \in L} \Phi(x)$.*

Proof. We claim that algorithm HALVELEADER depicted in Figure 4.9 performs the desired construction. The correctness of the algorithm is an immediate consequence of the fact that we restrict the leader set according to the guide rule. For the bound on the running time it suffices to observe that the unweighted median can be determined in linear time employing Goldman's algorithm [Gol71]. □

We point out that the size of the current tree T is not changed by an execution of algorithm HALVELEADER since it basically shifts nodes from the leader tree into the follower trees.

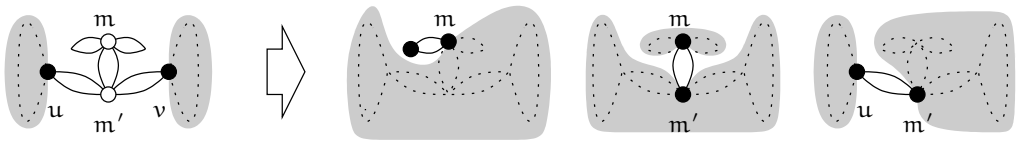

FIG. 4.10.: Example of the execution of HALVELEADER. Left: start of a phase with the leader tree separated by a valid split set $\{u, v, m, m'\}$; Right: possible states at the end of the phase. The shaded areas depict the follower trees.

The above procedure can be repeated iteratively on the current terminal tree until the leader tree consists only of a single edge. At this time we know that there is a Φ-solution on that edge. This situation is dealt with in the following section.

4.3.4. Finding an Optimum on a Single Remaining Edge

We now investigate the case where we have identified (for example, by iterating HALVELEADER) a single edge carrying the desired point of minimum Φ-score. We reduce this case to the computation of an optimal point in a modified tree.

To this end, we identify a set X_e of so-called *critical* points on the remaining edge $e = (u, v)$. These are points where the value of Φ could change. Clearly, this set of critical points has to contain points with a distance of exactly α to a node of T. Let

$$X_e := \{\, \text{point } x \in e \mid d(x, z) = \alpha \text{ for some node } z \in V \,\} \cup \{u, v\}$$

be the set of critical points on e. We sort point set X_e so that for $X_e = \{x_0, \ldots, x_k\}$ we have $x_0 = u$, $x_k = v$, and $d(u, x_i) < d(u, x_{i+1})$ for all i. Note that $|X_e| \leq n + 2$.

We claim that there are no more critical points on the edge e; in particular, the value of Φ is constant on each of the open intervals (x_i, x_{i+1}) for $i = 0, \ldots, k-1$. This follows from applying Lemma 4.2.5: For any two inner points x', x'' of the same interval, the sets of edges containing α-neighbors of x' and x'', respectively, are identical. Moreover, also the sets of corresponding front nodes coincide. Hence the user preferences and, in turn, the Φ-scores must be the same.

As a result, the function $\Phi(\cdot)$ is piece-wise constant on any edge. And even more importantly, we have reduced the problem of finding an optimal

solution within *all* points of some edge to finding an optimal solution for a discrete set of nodes.

4.3.5. A First $O(n \log n)$ Algorithm

We compute the absolute Φ-score of a tree in two phases.

Phase 1 starts with the input tree T_0 and chooses two arbitrary leaves as terminals. This defines T_0 to be the initial leader tree. Then we execute algorithm HALVELEADER repeatedly until the current leader tree consists of exactly two nodes connected by one edge. Theorem 4.3.4 guarantees that this edge always contains an optimal leader point.

At the beginning of phase 2, determine the set of critical points as outlined in the previous section. We add this set as real nodes on the edge. Now continue iterating HALVELEADER until we terminate again with a subedge, at which time an optimal point has been found.

Lemma 4.3.5 *An absolute Φ-solution can be found in time $O(n \log n)$ in a tree for all monotonic gain functions Φ.*

Proof. The correctness follows immediately from the guide rule. For the running time we observe that the first phase ends up with a single edge after $O(\log n)$ calls to HALVELEADER each of which takes $O(n)$ time. The computation and sorting of the critical points needs $O(n \log n)$. The rest of phase 2 has clearly the same complexity as the first phase. □

We are going to improve this result in the sequel by developing an algorithm with linear running time.

4.3.6. Sparsifying the Follower Tree

We now describe an operation that removes nodes from the follower tree. As motivated above we require the terminal tree to reflect the actual Φ-scores at least for those nodes in the leader tree. In contrast to the operation HALVELEADER described in the preceding section, the new operation HALVEFOLLOWER must take special care during removal of nodes, as the Φ-score of a leader node can decrease when its witness happens to reside in the follower tree and is selected for removal. The desired invariant is formalized by the notion of Φ-*equivalence*.

In this section we use subscript notation Φ_T and $\varphi_T(y, x)$ to denote the scores in a tree T, when the underlying function φ mentioned in Definition 3.5.4 is fixed. Recall that the latter is an abbreviation $\varphi_T(y, x) :=$

$\varphi(w_x(y), w_y(x))$, where the subtree weights $w_x(y)$ and $w_y(x)$ are computed with respect to T. Notation Φ and $\varphi(y, x)$ without subscripts is used when the underlying tree becomes clear from the context.

Definition 4.3.6 (Φ-equivalent terminal tree) Let T be a terminal tree such that its leader tree L is a subtree of the input tree T_0. Tree T is called *Φ-equivalent to* T_0 if

(i) $\Phi_T(x) \geq \Phi_{T_0}(x)$ for all points $x \in L$ and
(ii) $\min_{x \in L} \Phi_T(x) = \Phi^*_{T_0}$.

Let T be a terminal tree with leader tree L and assume that it is Φ-equivalent to input tree T_0. A point $x^* \in L$ with $\Phi_T(x^*) = \min_{x \in L} \Phi_T(x)$ is called an L-*optimum*. Each L-optimum in the current tree is also an optimal point in the original tree T_0 with respect to Φ_{T_0}. (Notice that the converse does not hold in general.) Moreover, the set of L-optima induces always a connected subtree, which is a consequence of the guide rule (Lemma 4.3.2). Namely, if x lies on path $P(x', x'')$ then either $\Phi(x') \geq \Phi(x)$ or $\Phi(x'') \geq \Phi(x)$ depending on where the witness of x lies. To find an optimum placement for the leader (with respect to the original tree) it thus suffices to restrict the view to the current leader tree L.

As motivated in the introductory example, the main purpose of the follower tree is to collect nodes removed from the leader tree. Such nodes may be needed to furnish certain nodes of the leader tree with a suitable witness in order to prevent that their Φ-score decreases. To this end it is not required that the follower trees be actually subtrees of T_0 and in fact their structure can be completely different from that of the corresponding part of the input tree.

From the following lemma we learn that property (i) of Φ-equivalence is tantamount to the property that the weight of the heaviest subtree with distance at least α to a given point $x \in L$ is never decreased by a sparsyfying modification made to the follower trees during our algorithm. Recall that we have made a similar observation in our introductory example.

Lemma 4.3.7 *Let $x \in L$ be a point in the leader tree, F be one of the follower trees, $F' := F - S_\alpha(x)$, and $y \in F'$ be chosen so that $w_x(y)$ is maximal. Then $\max_{y' \in F'} \Phi(y' \prec x) = \varphi(y, x)$.*

Proof. Let $y^* \in F'$ be an α-neighbor of x that attains $\max_{y' \in F'} \Phi(y' \prec x)$. Let $\tilde{y}$ be the front node of the follower party $U(y^* \prec x)$. From Lemma 4.2.1 and Lemma 4.2.5 we deduce that $\Phi(y^* \prec x) = \varphi(\tilde{y}, x)$. Let y be chosen as in

the statement of this lemma, and let y'' be the α-neighbor of x on the path $P(x,y)$. Then $\Phi(y^* \prec x) \geq \Phi(y'' \prec x) = \varphi(y'',x) \geq \varphi(y,x)$ by monotonicity. Furthermore, $\varphi(y,x) \geq \varphi(\tilde{y},x) = \Phi(y^* \prec x)$ since y was chosen maximal and the subtrees $T_y(x)$ and $T_{\tilde{y}}(x)$ are identical; here we exploit the fact that $y, \tilde{y}$ are part of the same follower tree. □

In the sequel we describe a linear time operation that halves the size of a follower tree but does not affect the Φ-equivalence of the current terminal tree T. To this end let T be the current terminal tree, L be its leader tree, u be one of its terminals, and $\delta > 0$ be an arbitrary number specifying a distance. As previously stated F_u denotes the follower tree incident with terminal u. We define subsets $F^+, F^- \subseteq F_u$ of *far* and *near* points (confer Figure 4.11) by

$$F^+ := \{\, y \in F_u \mid d(u,y) \geq \delta \,\} \qquad \text{and} \qquad F^- := \{\, y \in F_u \mid d(u,y) \leq \delta \,\}.$$

We will later choose δ as the median of the distances of all nodes in F_u its terminal u and this way divide the set of follower nodes in two almost equal sized parts, $V \cap F^+$ and $V \cap F^-$. (It is guaranteed in particular that both F^+ and F^- are nonempty.)

The main idea is to reduce the size of the follower tree by essentially discarding either the nodes in F^+ or in F^- from it. To this end we employ two subroutines DISCARDFAR (see Figure 4.13) and DISCARDNEAR (see Figure 4.14) that discard the nodes (and incident edges) in F^- or F^+, respectively, while taking further care that the Φ-scores of leader points do not decrease which is a main ingredient to maintain the Φ-equivalence property. Of course, those algorithms can not operate on the infinite point sets F^-, F^+. Instead, they actually employ similarly defined node sets (confer Step 3 in Figure 4.12):

$$\begin{aligned} \widetilde{F}^+ &:= \{\, \text{node } y \in F_u - u \mid d(u,y) \geq \delta \,\} \qquad \text{and} \\ \widetilde{F}^- &:= \{\, \text{node } y \in F_u - u \mid d(u,y) \leq \delta \,\}. \end{aligned}$$

Nevertheless, let's use the point sets F^+ and F^- in the following verbal description of the algorithms and in the proofs, which will simplify the presentation.

After determining F^+, F^- (or $\widetilde{F}^+, \widetilde{F}^-$, respectively) our algorithm determines a point $h \in F^+$ with $d(u,h) = \delta$ such that $w_u(h)$ is maximal. This defines a partition

$$\begin{aligned} L^- &:= \{\, x \in L \mid d(h,x) > \alpha \,\} \\ L^= &:= \{\, x \in L \mid d(h,x) = \alpha \,\} \\ L^+ &:= \{\, x \in L \mid d(h,x) < \alpha \,\}. \end{aligned}$$

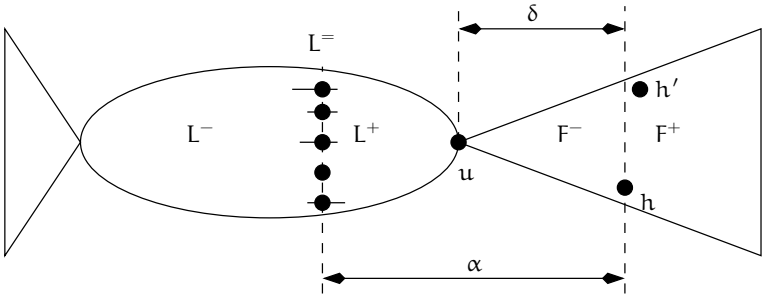

FIG. 4.11.: Illustration of the sets used in the construction.

The following two lemmas show to which extent the operations DISCARDFAR and DISCARDNEAR can change the Φ-scores of the leader tree and which parts are not affected.

Lemma 4.3.8 (DISCARDFAR) *The execution of* DISCARDFAR *does not decrease the* Φ*-score of points in* L^+ *and leaves the* Φ*-score of points in* $L^- \cup L^=$ *unchanged.*

Proof. Let T be the current terminal tree and T' be the new terminal tree after the execution of DISCARDFAR. Observe that for any point y the execution of the algorithm leaves all weights $w_x(y)$ unchanged as long as y is not removed.

Consider a point $x \in L^+$ with a witness y. If $y \notin F^+$ then $\Phi(x)$ does not change. Assume $y \in F^+$. If y lies within $S_\alpha(x)$ then $\Phi_{T'}(x) \geq \Phi_{T'}(x \prec x) = \Phi_T(x \prec x) = \Phi_T(x)$. Otherwise we may assume that y is an α-neighbor of x. Then $\Phi_T(x) = \varphi_T(y, x)$. Since in tree T we have $w_x(y) \leq w_x(h')$ we can conclude $\varphi_T(y, x) \leq \varphi_T(h', x)$ which equals $\varphi_{T'}(h', x)$ since we only collect the weight of the subtree in the node h'. Since $d_{T'}(x, h') > \alpha$ (confer Step 5 in DISCARDFAR) there must be a $y' \in T'$ such that $\Phi_{T'}(y' \prec x) \geq \varphi_{T'}(h', x)$ by Lemma 4.3.7. Clearly $\Phi_{T'}(x) \geq \Phi_{T'}(y' \prec x)$ which finally shows $\Phi_{T'}(x) \geq \Phi_T(x)$.

It remains to show that $\Phi_{T'}(x) = \Phi_T(x)$ for all $x \in L^- \cup L^=$.

At first consider $x \in L^=$ and let y' be its witness in T'. If $y' \in T - F^+$ then the claim follows from the above stated observation that the weights of subtrees are never changed. Otherwise by construction of T' the point y must lie on the new edge incident with h'. Hence $\Phi_{T'}(y' \prec x) = \varphi_{T'}(h', x) = \varphi_T(h', x)$. By Lemma 4.3.7 this equals $\Phi_T(x)$.

The arguments for $x \in L^-$ are similar with h' replaced by h, which completes the proof. □

Lemma 4.3.9 (DISCARDNEAR) *The execution of* DISCARDNEAR *does not decrease the Φ-score of points in* L^- *and leaves the Φ-score of points in* $L^= \cup L^+$ *unchanged.*

Proof. Let T be the current terminal tree and T' be the new terminal tree after the execution of DISCARDNEAR. Observe that for any $x \in L$ and $y \neq h$ the execution of the algorithm leaves all weights $w_x(y)$ and distances $d(u, y)$ unchanged as long as y is not removed.

Consider a point $x \in L^-$ with a witness y. If $y \notin F^-$ then $\Phi(x)$ does not change. Assume $y \in F^- - u$. Clearly $w_x(y) \leq w(F_u - u)$. On the other hand $w_x(h) = w(F_u - u)$ in T' and hence $\Phi_T(x) = \varphi(y, x) \leq \varphi(h, x) \leq \Phi_{T'}(x)$.

Now assume that $x \in L^= \cup L^+$ and y is a witness of x. If $y \in F^-$ then $d(x, y) \leq \alpha$ and hence also x is a witness of itself. The case $y \in F^+ - F^-$ is trivial due to the above observation. □

The preceding lemmas show that both discard operations are suited to maintain property (i) in the definition of Φ-equivalence. However we do not know so far whether the L-optimum is contained in L^+ or $L^=$ or L^- and hence which operation to apply. The following lemma states a criterion to distinguish those cases.

Lemma 4.3.10 (Φ-equivalence) *Let* T *be a terminal tree that is Φ-equivalent to the input tree* T_0*. If there is an* $x \in L^=$ *which has a witness* $y \in T_h(x)$ *then* T *remains Φ-equivalent after the execution of* DISCARDFAR*. Otherwise, if such an* x *does not exist,* DISCARDNEAR *maintains Φ-equivalence.*

Proof. Assume there are $x \in L^=$ and $y \in T_h(x)$ as stated. Due to the guide rule the subtree $T_h(x)$ contains an L-optimum. On the other hand $T_h(x)$ is contained in $L^- \cup L^=$. Then the claim follows immediately from Lemma 4.3.8.

Otherwise, if there is no such x, then $L^= \cup L^+$ must contain an L-optimum: Let x' be a point in L^- and x be the closest point in $L^=$. Then $\Phi(x') \geq \Phi(x)$ by the guide rule, since the witness of x lies outside of $T_h(x)$. The claim follows from Lemma 4.3.9. □

A straightforward implementation of a test which distinguishes both cases of the above lemma would be too expensive: $L^=$ might contain $\Theta(n)$ many points, and for each point determining a witness needs a linear time depth first search traversal. We will now describe a faster test routine with overall running time $O(n)$ and establish that this is still correct. The test traverses for each point $x \in L^=$ only the subtree $T_h(x)$ and determines a "quasi-witness" of x local to this tree. Since all those trees are node disjoint, the linear running time is obvious.

1 **input** terminal tree T with follower set F_u
2 let δ be the median of the multiset $\{\, d(u,y) \mid \text{node } y \in F_u - u \,\}$
3 compute follower sets $\widetilde{F}^- := \{\, \text{node } y \in F_u - u \mid d(u,y) \le \delta \,\}$, $\widetilde{F}^+ := \{\, \text{node } y \in F_u - u \mid d(u,y) \ge \delta \,\}$
4 determine point h in F_u with maximum weight $w_u(h)$ and distance δ to u
5 **if** h is not a node **then** replace it by a new zero weight node
6 compute $L^=$, X_q, Y_q
7 **if** X_q is not empty **then**
8 determine the set S of pairs (x,y) satisfying condition (4.2)
9 **if** $|S| = 1$ (say, $S = \{(x_c, y_c)\}$) and x_c has a witness in $T_h(x_c)$
10 **then** DISCARDFAR and **stop**
11 **else if** $\delta > \alpha$ **then**
12 DISCARDFAR and **stop**
13 DISCARDNEAR

FIG. 4.12.: Algorithm HALVEFOLLOWER

Definition 4.3.11 (Quasi-witness) Let $x \in L^=$. An α-neighbor $y \in T_h(x)$ is called *quasi-witness of* x if $\Phi(y \prec x)$ is maximum under all points in $T_h(x)$. The set X_q consists of all points in $L^=$ which have quasi-witnesses and Y_q is a set which contains for each $x \in X_q$ an arbitrary representative quasi-witness.

Observe that the set X_q and a corresponding set Y_q can be computed in $O(n)$.

Lemma 4.3.12 (Test criterion) *Assume that* $L^= \cup L^+ \neq \emptyset$ *contains no* L*-optimum. Then there is an* $x_c \in X_q$ *with quasi-witness* $y_c \in Y_q$ *such that all* L*-optima are contained in* $T_h(x_c)$*. Moreover,* (x_c, y_c) *is the only pair* (x,y) *with* $x \in X_q$ *and corresponding quasi-witness* $y \in Y_q$ *such that*

$$\varphi(y,x) = \max_{y' \in Y_q} \varphi(y',x)\,. \tag{4.2}$$

With the help of Theorem 4.3.12 we are able to formulate the algorithm HALVEFOLLOWER (confer Figure 4.12). It remains to describe how Step 8 of that algorithm is carried out. We first determine $y_1, y_2 \in Y_q$ such that the weights $w_h(y_1), w_h(y_2)$ are the two heaviest ones. Let x_1, x_2 be the corresponding points in X_q. For each $x \in X_q$, in order to examine if the pair (x,y) satisfies condition (4.2) we only need to compare $\varphi(y_1,x)$ with $\varphi(y,x)$ (with the exception of the case $x = x_1$ where we compare $\varphi(y_2,x)$ with $\varphi(y,x)$ instead.)

determine node $h' \in \widetilde{F}^+ - \widetilde{F}^-$ with maximum $w_u(h')$
let $D \leftarrow \widetilde{F}^+ - \{h, h'\}$
for all edges (s,t) with $s \in F_u - D$ and $t \in D$
    add weight $w_s(t)$ to node s and discard subtree $T_s(t)$
increase the length of the edge incident to h' to $+\infty$ (e.g. add α)

FIG. 4.13.: Algorithm DISCARDFAR to discard most nodes in F^+

set $w(h) \leftarrow w(\widetilde{F}^-)$
let $D \leftarrow \widetilde{F}^- - \{h\}$
for all edges (s,t) with $s \in D$ and $t \in F_u - D$
    connect t to node h via a new edge of length $d(u,t) - \delta$
discard all nodes of $D - \{u\}$ from the tree

FIG. 4.14.: Algorithm DISCARDNEAR to discard most nodes in F^-

Proof (of Theorem 4.3.12). If $L^= \cup L^+$ contains no L-optimum then all L-optima are contained in one single subtree $T_h(x)$ for some $x \in L^=$. Clearly x has no witness outside of $T_h(x)$ by the guide rule (Lemma 4.3.2). Since $x \in L^= \cup L^+$ and x is not an L-optimum by premise, x cannot be a witness of itself; therefore we can assume the witness to be an α-neighbor. We can conclude that $x \in X_q$ and that the corresponding quasi-witness $y \in Y_q$ is in fact a proper witness.

Consider $x' \in X_q - x$ and let $y' \in Y_q - y$ be the corresponding quasi-witness. Let y'' be an α-neighbor of x on path $P(x,y')$. Then $\Phi(y'' \prec x) < \Phi(y \prec x)$ for otherwise $T_h(x)$ could not contain points with Φ-scores strictly smaller than $\Phi(x)$ by the guide rule. Since

$$\begin{aligned} \varphi(y,x') &\geq \varphi(y,x) = \Phi(y \prec x) \\ &> \Phi(y'' \prec x) = \varphi(y'',x) \\ &\geq \varphi(y',x) \\ &\geq \varphi(y',x') \end{aligned}$$

we can conclude $\varphi(y,x) > \varphi(y',x)$ and $\varphi(y,x') > \varphi(y',x')$. This completes the proof. □

Lemma 4.3.13 (Halve follower tree) *Let T be a terminal tree Φ-equivalent to the input tree T_0, let F_u be its follower tree. Then the algorithm* HALVEFOLLOWER

constructs a follower tree F'_u *with size* $|F'_u| \leq \frac{1}{2}|F_u| + 2$ *such that replacing* F_u *by* F'_u *does not invalidate the* Φ*-equivalence of* T*. The running time is* $O(|T|)$*.*

Proof. At first consider the case where X_q is not empty and the condition in Step 10 is met. By the guide rule $T_h(x_c)$ and therefore $L^- \cup L^=$ contains an L-optimum and by Lemma 4.3.8 the invocation of DISCARDFAR maintains Φ-equivalence.

Now assume that $X_q \neq \emptyset$ but the condition in Step 10 is not satisfied, that is, S contains at least two pairs or $T_h(x_c)$ contains no witness of x_c. By Theorem 4.3.12 $L^= \cup L^+$ contains an L-optimum which allows the execution of DISCARDNEAR in Step 13.

It remains to consider the case $X_q = \emptyset$. Then either $L^= \cup L^+$ is empty and $\delta > \alpha$; in this case DISCARDNEAR is applicable. Otherwise by Theorem 4.3.12 $L^= \cup L^+$ must contain an L-optimum and DISCARDFAR maintains Φ-equivalence.

The claim on the running time of the algorithm is clear since the computation of the median in line 3 can be carried out in linear time with the well known median-of-medians algorithm [BFP+73]. □

4.3.7. The Main Algorithm

We now describe the main algorithm. During the execution the algorithm maintains a current terminal tree T with leader tree L, terminals u, v, and follower trees F_u, F_v. The terminal tree is initialized with the input tree T_0 and two arbitrary leaves playing the role of the terminals (thus initially the leader tree L is identical with T_0 and the follower trees are empty). Iteratively the algorithm determines a maximum among the sizes $|L|, |F_u|, |F_v|$ (ties are broken with a preference for a follower tree) and halves that component using one of the subroutines HALVEFOLLOWER or HALVELEADER described above. The algorithm ends at last when the size of T falls below 6 (then neither the leader tree nor the follower tree can further be reduced by the subroutines) at which time the L-optimum solution in the resulting leader tree can be determined in constant time using the algorithm from Section 4.3.5. The correctness of this algorithm follows from Theorem 4.3.13 and Theorem 4.3.4.

We now analyze the running time. To this end we subdivide the sequence of iterations into *phases*; each operation HALVEFOLLOWER terminates a current phase. Let n_i denote the size of the terminal tree after the ith phase. Observe that HALVELEADER does not change the terminal tree's total size. If HALVEFOLLOWER is executed this is because there is a follower tree of

size $f \geq \frac{1}{3}n_i$; the resulting size f' satisfies $f' \leq \frac{1}{2}f + 2$. Hence we can deduce that $n_{i+1} \leq \frac{5}{6}n_i + 2$ for all i. Moreover in each phase we can have at most two calls to HALVELEADER before an HALVEFOLLWER intervenes. Therefore the running time of phase i is linear in $O(n_{i-1})$. Since the sequence $(n_i)_i$ is bounded from above by a geometrically decreasing sequence this yields an overall linear running time:

Theorem 4.3.14 (Absolute Φ-solution of a tree) *For any monotonic gain function, an absolute Φ-solution can be found in time $O(n)$ in a tree with n nodes.*

Corollary 4.3.15 *An absolute relaxed Simpson, Condorcet, security, plurality, $(1,1)$-centroid and Stackelberg solution of a tree can be computed in linear time.* □

4.4. Discussion of Discrete Φ-Solutions

In this section we are investigating the problem of computing a discrete Φ-score, that is, the case where leader and follower are restricted to place on nodes of the tree only. It turns out that in many aspects the problem is similar to the absolute case discussed above so that it suffices to report the main differences here.

The first difference affects the definition of the α-neighborhood.

Definition 4.4.1 (α-neighborhood) Let $x \in V$ be a leader node. A node $y \in V$ is called an *α-neighbor of* x if it is the only node on the path $P(x,y)$ whose distance to x is strictly greater than α. The set of all α-neighbors of x is denoted by $N_\alpha(x)$.

It is easy to see that we still can assume without loss of generality that for a leader x a witness can always be found in the set $N_\alpha(x) \cup \{x\}$ (confer Lemma 4.2.4).

A consequence of α-neighbors being nodes only is that the distance from a leader to its witness can actually exceed α by far. In this case the leader party grows in direction to the witness and the front node is no longer necessarily in the vicinity of the leader.

Theorem 4.4.2 (Characterization of front nodes) *Let x be a leader node and y be an α-neighbor of x. Then the front node $\tilde{x}$ is the $\frac{d(x,y)+\alpha}{2}$-neighbor of y on $P(x,y)$, and the front node $\tilde{y}$ is y itself.*

Proof. As in the proof of Lemma 4.2.1 we only consider projections of nodes on $P(x,y)$. A node $z \in P(x,y)$ prefers x if and only if

$$d(z,y) - d(z,x) > \alpha \iff 2 \cdot d(z,y) - d(x,y) > \alpha \iff d(z,y) > \frac{d(x,y)+\alpha}{2}.$$

The node z on $P(x,y)$ which satisfies this condition and maximizes $d(x,z)$ is the front node $\tilde{x}$. Since it simultaneously minimizes $d(z,y)$, it is the $\frac{d(x,y)+\alpha}{2}$-neighbor of y on the path.

The remaining claim, $\tilde{y} = y$, is clear since y is chosen to be an α-neighbor of x. □

To determine all α-neighbors of x, we perform a depth first search traversal from x and maintain distances $d(x,y)$ to the root where y is the currently traversed node. If y is an α-neighbor, the corresponding front node $\tilde{x}$ can be found as follows: Let $x = v_0, v_1, \ldots, v_k = y$ be the path of nodes on the stack when the depth first search traversal arrives at the α-neighbor y. Then $\tilde{x} = v_i$ where i is the maximum index such that $d(v_i,x) < (d(x,y) - \alpha)/2$. This node can be found in $O(\log k) \subseteq O(\log n)$ by binary search in the array $(v_i)_i$.

Theorem 4.4.3 (Discrete Φ-score of a node) *For a given node x, the discrete score $\Phi(x)$ can be computed in time $O(n \log n)$ on a tree for all monotonic gain functions.* □

Observe that a guide rule (confer Lemma 4.3.2) holds similarly in the discrete case with the only difference that the operation HALVELEADER takes now $O(n \log n)$ time. This yields the following result:

Theorem 4.4.4 (Discrete Φ-solution of a tree) *A discrete Φ-solution can be computed in an n-node tree in $O(n (\log n)^2)$ time for any monotonic gain function.* □

4.5. Strong Φ-Solutions

Let's return to absolute MGFs. It is no serious restriction to assume that our input tree does not have zero-weight leaves since such leaves can be removed without affecting the weight of any subtree.

Making this assumption, it is not hard to see that $w(y \prec x) = 0$ if and only if $d(x,y) \leq \alpha$. The monotonicity property of any gain function Φ requires that $\Phi(y \prec x)$ equal the same constant Φ_0 for all pairs (x,y) with distance

at most α. The relation $\Phi(x \prec x) = \Phi_0$ thus establishes a lower bound on $\Phi(x)$ for each point x. However, the value $\Phi_0 \in \mathbb{Q}$ is not uniquely determined and can in fact be chosen arbitrarily without violating the monotonicity property. One can observe that while the constant Φ_0 is decreased the set of Φ-solutions shrinks simultaneously.

From a certain point of view the observation that the follower y can always locate at x and enforce the gain $\Phi(y \prec x) = \Phi(x \prec x) = \Phi_0$ yields somewhat uninteresting solutions: it provides no further information on the stability of x as this placement is possible for each point. This suggests to fade out the impact of location duplication and require that the follower place at a location substantially different from the leader, that is, at distance larger than α. This is enforced by setting $\Phi_0 := -\infty$. The resulting set of solutions is called *strong* solution set:

Definition 4.5.1 (Strong Φ-solution) Let Φ be a monotonic gain function induced by a function $\varphi\colon \mathbb{Q}\times\mathbb{Q} \to \mathbb{Q}$. Let Φ' be the monotonic gain function defined by

$$\Phi'(y \prec x) := \begin{cases} \varphi(w(y \prec x), w(x \prec y)) & \text{if } d(x,y) > \alpha \\ -\infty & \text{otherwise}. \end{cases}$$

Then the Φ'-solutions are called *strong* Φ*-solutions*.

Consider the security MGF Δ. Since $\Delta_0 := \Delta(x \prec x) = 0$ we obtain $\Delta^* \geq 0$. Now let m be an weighted median of the given tree. Then each of the connected components obtained by removing m from T has a weight of at most $\frac{1}{2}w(T)$. Let y be a witness of m which we can assume to be a neighbor of m. Then $\Delta(y \prec m) = w_m(y) - w_y(m) \leq \frac{1}{2}w(T) - \frac{1}{2}w(T) = 0$. Hence $\Delta^* = 0$. This relation is no longer true for *strong* security solutions as pointed out in Figure 4.1.

4.6. Competitor-Sensitive Gain Functions

An intrinsic property of the suggested model of monotonic gain functions as specified in Definition 3.5.4 is that the leader and the follower share the same estimations on the weights of users. An obvious extension of the model is hence to have two different weight functions w_L, w_F for the leader and the follower, respectively, and to change Definition 3.5.4 to

$$\Phi(y \prec x) := \varphi(w_F(y \prec x), w_L(x \prec y)) \tag{4.3}$$

which we call a *competitor sensitive* gain function. Albeit strictly speaking this is not a monotonic gain function, it can easily be seen that the approach outlined in this chapter can be adapted to also handle this extended model.

This can be applied to a generalization of the Stackelberg problem. Normally, when a user is undecided, his demand is split equally amongst the two competing providers [HTW90]. There can, however, be situations where undecided users split their demand on a per user basis among the two competitors. This can formally modeled by introducing a function $f\colon V \to [0,1]$ which specifies the individual user demand gained by the follower in the case where the user is undecided. (The original Stackelberg problem is then the special case of $f \equiv \frac{1}{2}$.) Thus we end up with a gain function

$$\Phi(y \prec x) := w(y \prec x) + (f \cdot w)(y \sim x)\,. \tag{4.4}$$

This can be modeled by a competitor sensitive gain function: to this end define $w_F := (1-f)\cdot w$ and $w_L := f \cdot w$ and $\varphi(\mu,\nu) := (f\cdot w)(T) + \mu - \nu$ for all $\mu, \nu \in \mathbb{Q}$; it is easy to verify that plugging this into (4.3) yields (4.4).

4.7. Concluding Remarks

The main contribution of this chapter is a linear time algorithm for computing a Φ-solution of a monotonic gain function Φ on a tree graph. This generalizes well-known results of several unrelaxed competitive and voting location problems to the relaxed preference model of MGFs. In comparison with Goldman's algorithm [Gol71] for computing a median of a tree (and hence also unrelaxed Simpson, Condorcet, centroid etc.), our approach for the general, relaxed model requires far more involved divide-and-conquer techniques: The main idea was to construct a geometric decreasing sequence of sparser and sparser trees carrying enough information to retrieve an optimum of the original tree. The algorithm stops when a constant number of nodes is reached. We have also discussed a similar algorithmic approach for handling the discrete model where facilities are to be located at the nodes of the graph. It is an interesting open question whether the additional logarithmic factors included in the running time of that algorithm are really necessary.

5. Computing all φ_0-Bounded Solutions of a Tree

Recall that any unrelaxed centroid of a tree is also a median and vice versa. It is a well-known fact that the set of medians of a tree is either a single node or an edge. Thus, an optimal leader position for the centroid problem is not always uniquely determined. Rather, there can be infinitely many optima. Such an outcome gives the leader more leeway and he has then the opportunity to choose one of those centroids according to a *secondary* criterion. For example, he may pick a location that minimizes the production cost among all optima. But he could also identify places being particularly attractive for the customers, for example, due to good parking facilities. In some situations it can therefore be favorable to have the complete set of centroids at hand. Of course, this can also be desirable for general monotonic gain functions.

In Chapter 4 we have presented an optimal algorithm for computing *one single* Φ-solution of a tree. Since a point x is a Φ-solution if and only if it is Φ^*-bounded, that is, if $\Phi(x) \leq \Phi^*$, the set of Φ-solutions equals $S(\alpha, \Phi^*)$. (Confer Definition 3.5.8 for the definition of φ_0-boundedness.) Hence determining all Φ-solutions is a special case of computing the set $S(\alpha, \varphi_0)$ of all φ_0-bounded solutions for a given φ_0, which is the problem we are going to discuss in this chapter.

Garcia and Pelegrin [GP03] investigate the problem of computing the complete set of Stackelberg solutions with parametric prices of a tree. They suggest a quite complicated algorithm with overall running time $O(n^3 \log n)$. The main idea of their algorithm is to move, in a first phase, a point in small steps along edges in the tree until an arbitrary optimal position has been found, and in a second phase extend the set of optimal solutions by similar small point shifts. The algorithm that we present in this chapter is also based on a two-phase approach: The first phase determines one initial Φ-solution which allows the second phase to explore the entire set of Φ-solutions. In contrast to the algorithm of Garcia and Pelegrin our approach does not employ local improvement steps but can rather be considered as a dynamic programming approach.

At first glance it seems obvious to adapt the algorithm of Chapter 4 for computing a single Φ-solution so that it is able to maintain the entire solution set. Let's remember the main ideas of that algorithm: It creates a decreasing sequence of trees being Φ-equivalent to the input tree T_0 (confer Definition 4.3.6). If we end up with a tree T_{final} such that the number of nodes is smaller than a given constant n_0 we determine a Φ-solution x. Due to the Φ-equivalence of T_{final} the solution x is guaranteed to be a Φ-solution in T_0, too. Having this in mind, it seems natural to compute the *complete set* of Φ-solutions of T_{final} and then to transform it back into the original tree T_0.

Unfortunately, a closer look reveals that we sometimes lose necessary information about the input tree T_0 during the sparsifying process: Assume that the set of centroids forms an edge of T_0. If we now create *more than* n_0 zero-weighted dummy nodes at that edge, clearly all of them being optimum, our algorithm will return at most n_0 of them.

Nevertheless, we can obtain valuable information from the algorithm of Chapter 4: The Φ-score Φ^* of the input tree. In fact, the exploration phase of our algorithm will use that knowledge to compute the solution set by means of a dynamic programming approach.

The chapter is organized as follows: We start with computation of the set of φ_0-bounded solutions for leader independent MGFs. We will see that this case is solvable in linear time. Since the Stackelberg problem with parametric prices is leader independent our algorithm brings the running time $O(n^3 \log n)$ of Garcia and Pellegrin's algorithm [GP03] down to $O(n)$, which is optimal. We will then argue that the same problem for leader *dependent* MGFs is harder. In particular we will prove a lower bound of $\Omega(n \log n)$ for the security function. This shows also that it is easier (in terms of the asymptotic running time) to compute one single Φ-solution than all of them. Finally, we present an algorithm that determines all φ_0-bounded solutions for an arbitrary MGF in $O(n \log n)$ time, which is optimal.

5.1. Leader Independent Monotonic Gain Functions

In this section we describe the computation of the set of all φ_0-bounded solutions for a leader independent MGF for a given φ_0. Throughout the remainder of this section we assume that $\varphi_0 \geq \varphi(0)$ where we use the notation $\varphi(w_y) := \varphi(w_y, 0)$. If φ_0 were smaller than $\varphi(0)$ no φ_0-bounded solution would exist since any point would be dominated by itself.

First we define for each pair (u, v) of points the value $\delta_u(v)$ as follows:

$$\delta_u(v) := \sup\{\, d(u, y) \mid \text{node } y \in T_u(v) \text{ and } \varphi(w_u(y)) > \varphi_0 \,\}. \tag{5.1}$$

This value $\delta_u(v)$ denotes the largest distance between the current candidate u and any node in the subtree $T_u(v)$ that *dominates* u. Here, dominating means only that there is a subtree whose weight is sufficiently large. If $\delta_u(v) > \alpha$ holds additionally, this implies that there is a node y with $\Phi(u) \geq \Phi(y \prec u) \geq \varphi(w_u(y)) > \varphi_0$. Thus, $\delta_u(v) > \alpha$ proves that the score $\Phi(u)$ exceeds the bound φ_0 and hence u cannot be φ_0-bounded.

We now use these values to characterize the set $S(\alpha, \varphi_0)$. Recall that the neighbors of an inner point on edge (u, v) are just the nodes u and v.

Lemma 5.1.1 *A point or node $x \in T$ belongs to $S(\alpha, \varphi_0)$ if and only if $\delta_x(v) \leq \alpha$ for all neighbors v of x.*

Proof. From Lemmas 4.2.1 and 4.2.4 we obtain that x has Φ-score at most φ_0, if and only if for each y with $d(x, y) > \alpha$ the condition $\varphi(w_x(y)) \leq \varphi_0$ holds. See also Figure 5.1 (a). □

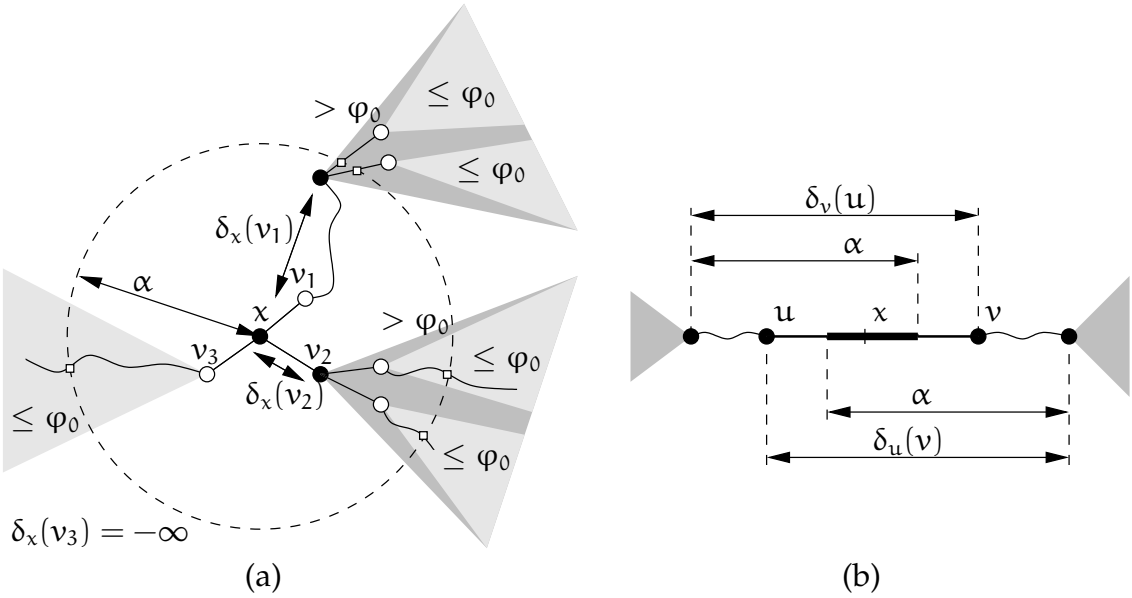

FIG. 5.1.: (a) Illustrating the proof of Lemma 5.1.1. (b) Interval test on edge (u, v).

We argue that knowledge of the values $\delta_u(v), \delta_v(u)$ for all edges (u, v) of the tree is sufficient to output the desired set $S(\alpha, \varphi_0)$ as a collection of nodes, edges, and edge segments.

According to Lemma 5.1.1, to check whether a *node* $u \in V$ is φ_0-bounded, it suffices to inspect the δ-values on the edges incident to u, which takes time $O(\deg(u))$. Therefore, we can determine all nodes that are φ_0-bounded in linear time once the δ-values have been computed.

The *inner points* that are part of the solution set can be characterized as intervals: Let $x \in (u, v)$ be an inner point on some edge (u, v). Then x is φ_0-bounded if and only if (confer also Figure 5.1 (b))

$$\delta_u(v) - \alpha \leq d(u, x) \leq d(u, v) - \delta_v(u) + \alpha\,. \tag{5.2}$$

We call this condition *interval test*. The corresponding intervals can be determined for all edges in linear total time if the δ-values are known.

Hence, in order to compute the set of φ_0-bounded solutions we only need to determine $\delta_u(v)$ and $\delta_v(u)$ for all edges (u, v). We will see that this can be carried out in linear time by a dynamic programming approach. According to the above observations this yields a linear time algorithm for computing the set of Φ-solutions for any leader independent MGF Φ.

The algorithm that accomplishes this is based on a routine for propagating the δ-values among incident edges (confer Fig. 5.2 for details). Let (u, v) be an edge. Then propagate(u, v) computes $\delta_u(v)$ if the values of $\delta_v(u')$ are known for each $u' \in N(v)$ different from u. This can be done as follows: If v does not dominate u then $\delta_u(v) = -\infty$. Otherwise the following equation holds and can be derived by elementary considerations:

$$\delta_u(v) = d(u, v) + \max\{\,\delta_v(u') \mid u' \in N(v) \text{ and } u' \neq u\,\} \cup \{0\}\,.$$

Note that propagate(u, v) takes time $O(\deg(v))$ where $\deg(v)$ denotes the node degree of v.

Let s be an arbitrary node and consider T as an s-rooted tree. The algorithm computes the values $\delta_u(v)$ for all edges (u, v) by two depth first search traversals:

The first traversal invokes propagate_up(s). This subroutine traverses the tree bottom up and calculates the value $\delta_u(v)$ for all edges (u, v) where u is the father of v. The running time is linear in the number of nodes: this follows since propagate$(\cdot, v)$ takes time $O(\deg(v))$ and is called exactly once for each node v.

The second traversal invokes propagate_down(s) and is a top-down traversal that computes the missing values $\delta_v(u)$. A naive implementation as in Fig. 5.2 results in a quadratic runnning time since propagate$(\cdot, u)$ would be called at least $\deg(u) - 1$ times for each node u. This can be avoided as follows: When node u is reached, first traverse the set

```
1  procedure propagate(u, v)
2      if φ(w_u(v)) ≤ φ_0 then return −∞
3      δ ← 0
4      for each u′ ∈ N(v), u′ ≠ u do
5          δ ← max{δ, δ_v(u′)}
6      return δ + d(u, v)
7
8  procedure propagate_up(u)
9      for each son v of u do
10         propagate_up(v)
11         δ_u(v) ← propagate(u, v)
12
13 procedure propagate_down(u)
14     for each son v of u do
15         δ_v(u) ← propagate(v, u)
16         propagate_down(v)
```

FIG. 5.2.: Subroutines used for computing the set of Φ-solutions on trees.

of all neighbors of u (that is, the sons and the father) and determine $v_1 := \arg\max_{v \in N(u)} \delta_u(v)$ and $v_2 := \arg\max_{v \in N(u)-v_1} \delta_u(v)$. Then, instead of invoking propagate(v, u), we compare each son v of u with v_1. Assume that u dominates v, that is, $\varphi(w_v(u)) > \varphi_0$ holds. If $v \neq v_1$ then $\delta_v(u) = \max\{\delta_u(v_1), 0\} + d(v, u)$ otherwise, if $v = v_1$, we have that $\delta_v(u) = \max\{\delta_u(v_2), 0\} + d(v, u)$. With this modification we can evaluate the values $\delta_v(u)$ for all sons v of u in total running time $O(\deg(u))$, which guarantees an overall linear running time.

This yields the following result:

Theorem 5.1.2 *Let T be a tree with n nodes, let Φ be a leader independent MGF, and let $\varphi_0 > 0$. Then the set of all φ_0-bounded solutions can be computed in $O(n)$ time. In particular, this holds for the set of Φ-solutions.*

Proof. The first claim has been proved above. In order to compute the set of Φ-solutions we determine previously the score Φ^* by means of the algorithm of Chapter 4. □

Corollary 5.1.3 *In a tree with n nodes, the set of all Stackelberg solutions with parametric prices can be computed in $O(n)$.*

Proof. Recall the definition of the Stackelberg problem with parametric prices (confer Section 3.4) where leader and follower sell their good at different prices p and q, respectively. In the case $p \geq q$, the leader captures nothing [GP03]. The nontrivial case $p < q$ is similar to the α-Simpson problem as pointed out in Section 3.2. We first compute the Stackelberg score Σ^* with the help of the algorithm described in Chapter 3: The essential difference between the underlying user preference models is that users satisfying the equality $d(u, y) = d(u, x) - \alpha$ decide for the y-party in the Stackelberg case while they are undecided in the monotonic gain function model. This suggests to redefine the α-neighborhood (for the original definition confer Def. 4.2.3) of a point x to be the set $N_\alpha(x) := \{y \mid d(y, x) = \alpha\}$, and it is easy to observe that with this alternative definition the algorithm indeed computes the Stackelberg score.

The linear-time computation of the δ-values has been outlined above and is not changed. In the subsequent edge tests (confer Equation (5.2) on page 85) we basically replace $\leq$ with $<$. □

5.2. Computational Lower Bound for the Absolute Security Set

In this section we contrast the linear-time results for the set of Φ-solutions of leader independent MGFs established above with a lower bound of $\Omega(n \log n)$ for computing the whole security set. This shows that in general computing the set of all Φ-solutions (and hence also of φ_0-bounded solutions) becomes harder when the monotonic gain function no longer depends solely on the weight of the follower party. Furthermore it demonstrates that the computation of all Φ-solutions is more difficult than finding an arbitrary representative of that set since the latter problem can be solved in linear time. More precisely, we prove a lower bound of $\Omega(n \log n)$ for the exploration of the absolute security set in a unit cost RAM model as defined by Ben-Amram and Galil [BG01]. In Section 5.3 we will complement this result by developing an algorithm with a matching running time $O(n \log n)$.

To derive the lower bound we make use of general results for several computation models from [Yao91, BG01] which can essentially be summarized as follows: Let $(W_n)_{n\in\mathbb{N}}$ be a sequence of point sets where each set $W_n \subseteq \mathbb{R}^n$ is *scale invariant* (this means that $z \in W$ implies $\lambda z \in W$ for all $\lambda > 0$) and *rationally dispersed* (which means that for every $x \in \mathbb{R}^n$ and every $\varepsilon > 0$ there is a rational z in the ε-neighborhood of x such that $x \in W \Leftrightarrow z \in W$). Then

the problem of deciding whether a given $x \in \mathbb{Z}^n$ is contained in W_n needs $\Omega(\log \beta((\overline{W_n})^\circ))$ time. Here, $\beta(X)$ denotes the number of connected components of a set X, $\overline{X}$ the closure of X, X° the interior of X, and $\partial X = \overline{X} - X^\circ$ the boundary of X.

In the sequel we are going to define a set W with $n!$ connected components which, according to the above mentioned results, needs time $\Omega(n \log n)$ to be decided. Then we show that the set W can be decided by computing the security set, which yields the desired lower bound. To this end we define the set $W \subseteq \mathbb{R}^{2n}$ as follows:

$$W := \{ (d_1, \ldots, d_n, w_1, \ldots, w_n) \mid d_j \leq d_i \Rightarrow w_j \leq w_i \text{ for all } i, j = 1, \ldots, n \}.$$

Observe that W is scale invariant and rationally dispersed.

Lemma 5.2.1 *The set $(\overline{W})^\circ$ has at least $n!$ many connected components.*

Proof. Let $\pi \in S_n$ be a permutation in the symmetric group S_n. Consider the set

$$W_\pi := \left\{ (d_1, \ldots, d_n, w_1, \ldots, w_n) \in W \;\middle|\; \begin{array}{l} d_{\pi(1)} < \ldots < d_{\pi(n)} \text{ and} \\ w_{\pi(1)} < \ldots < w_{\pi(n)} \end{array} \right\}.$$

For different permutations π, the sets W_π are maximally connected subsets of $(\overline{W})^\circ$ and pairwise disjoint. This follows from the following observation: If $z \in \partial W$ then there must be indices $j \neq i$ such that $d_j = d_i$ or $w_j = w_i$. Since $d_j = d_i$ implies $w_j = w_i$ we can assume that $d_j \leq d_i$ and $w_j = w_i$. For each $\varepsilon > 0$ consider the point z' with is derived from z by replacing the coordinates $d_i' := d_i + \varepsilon$, $d_j' := d_j - \varepsilon$, $w_i' := w_i - \varepsilon$, $w_j' := w_j + \varepsilon$. Then $z' \notin \overline{W}$ because in the $\frac{\varepsilon}{2}$-neighborhood of z' there is no point of W and hence $z \notin (\overline{W})^\circ$. Thus $(\overline{W})^\circ \subseteq W^\circ$ and in fact $(\overline{W})^\circ = W^\circ$. It is not hard to see that the sets W_π are the connected components of W°. Hence $(\overline{W})^\circ$ has at least $n!$ many connected components. □

Together with the results of Ben-Amram and Galil [BG01] it follows that in the unit cost RAM model with integer inputs the complexity of deciding membership for W is $\Omega(n \log n)$.

Now we are going to show that deciding whether the security set of a given tree contains a particular subset of the nodes is as least as hard as deciding membership in W. Let $z := (d_1, \ldots, d_n, w_1, \ldots, w_n) \in \mathbb{Z}^{2n}$ be given. We construct a tree T as a star consisting of a center node v and $2n$ nodes x_i, y_i for $i = 1, \ldots, n$. Confer Figure 5.3. The node weights are set to $w(x_i) :=$

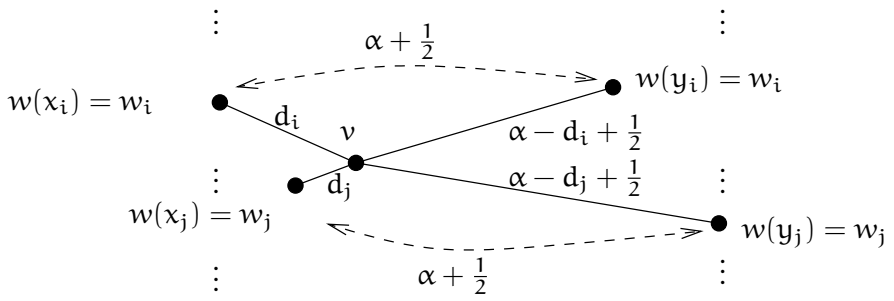

FIG. 5.3.: Star constructed from a tuple $(d_1, \ldots, d_n, w_1, \ldots, w_n)$ for which membership in W has to be decided. Observe that $\Delta(x_i) > 0$ if and only if $w_j > w_i$ for some j with $d_j \leq d_i$.

$w(y_i) := w_i$. With $\alpha := 2 \cdot \max_i d_i$ the edge lengths are set to $d(v, x_i) := d_i$ and $d(v, y_i) := \alpha - d_i + \frac{1}{2}$ for all i.

Assume that the leader locates at a leaf x_i with $\Delta(x_i) > 0$. Then the follower won't locate at an edge incident with some x_j since $d(x_i, x_j) = d_i + d_j \leq \alpha$ for all i, j. Instead the follower will pick a position on an edge incident at some y_j with $d_j \leq d_i$ for otherwise we had again $d(x_i, y_j) = d_i + \alpha - d_j + \frac{1}{2} < \alpha$. Hence the security score of x_i is determined by $\Delta(x_i) = \max\{ w_j - w_i \mid d_j \leq d_i \} \cup \{0\}$. Then it is clear that $\Delta(x_i) = 0$ for all i if and only if $z \in W$. On the other hand $\Delta^* = 0$ since $\Delta(v) = 0$. We can conclude: the node set $\{x_1, \ldots, x_n\}$ is contained in the security set if and only if $z \in W$ which completes the reduction.

Theorem 5.2.2 (Lower bound) *The complexity of the computation of the absolute security set of a tree is $\Omega(n \log n)$ in the unit cost RAM model.* □

As a consequence of this result the algorithm in the following section for general monotonic gain functions is optimal.

5.3. Computing the Set of All φ_0-Bounded Solutions

In this section we develop an algorithm that outputs the set of all absolute φ_0-bounded Φ-solutions in a tree where Φ is an arbitrary monotonic gain function and $\varphi_0 \geq \varphi(0, 0)$ is a given bound. To this end we introduce for each point u and neighbor $v \in N(u)$ a value $\delta_u(v)$ which denotes the maximum distance of u to a node $y \in T_u(v)$ that dominates v, that is,

$$\delta_u(v) := \sup\{ d(u, y) \mid \text{node } y \in T_u(v) \text{ and } \varphi(y, u) > \varphi_0 \}. \qquad (5.3)$$

This is a straightforward generalization of (5.1) since in the case of a leader independent MGF we have $\varphi(y,u) = \varphi(w_u(y))$. Recall that we set $\varphi(y,u) := \varphi(w_u(y), w_y(u))$. Observe that the set can be empty; in that case $\delta_u(v) = \sup\emptyset = -\infty$.

Lemma 5.3.1 *A point or node $x \in T$ is φ_0-bounded if and only if $\delta_x(v) \leq \alpha$ for each neighbors v of x.*

Proof. Assume that $\delta_x(v) > \alpha$ for some neighbor v of x. Then there is a node $y \in T_x(v)$ with $d(x,y) > \alpha$ and $\varphi(y,x) > \varphi_0$. Let $y' \in P(x,y) \cap N_\alpha(x)$. Obviously, $w_x(y') \geq w_x(y)$ and on the other hand the leader party is now exactly $T_y(x)$ (confer Lemma 4.2.5). Hence $\Phi(x) \geq \Phi(y' \prec x) = \varphi(y',x) \geq \varphi(y,x) > \varphi_0$.

Conversely, if $\Phi(x) > \varphi_0$ then there is an α-neighbor y of x that is also a witness. Let $\tilde{y}$ be the front node of y (which possibly coincides with y). Then $\varphi(\tilde{y},x) = \Phi(y \prec x) = \Phi(x) > \varphi_0$ and $d(x,\tilde{y}) > \alpha$. Hence $\delta_x(v) > \alpha$ for the neighbor v of x on the path $P(x,\tilde{y})$. □

Lemma 5.3.1 allows us to explore the set of φ_0-bounded solutions by means of the interval test (5.2), exactly as we did for leader independent MGFs in the preceding Section. Again the running time is linear when all δ-values are known. Thus it remains to discuss how these δ-values can be computed when a leader *dependent* monotonic gain function is involved.

In the case of a leader independent MGF we were able to compute the δ-values by a fast propagation of these values among incident edges. More precisely, if (u,v) is an edge and $\varphi(w_u(v)) > \varphi_0$, then $\delta_u(v) = d(u,v) + \max\{\delta_v(u') \mid u' \in N(v) \text{ and } u' \neq u\} \cup \{0\}$. This reflects the fact that the Φ-score depends only on the weight of the follower party. Specifically, for a given $u \neq y$, the value $\varphi(y,x) = \varphi(w_u(y))$ is constant for all $x \in T_y(u)$. Unfortunately, an analogous relation does not hold for general MGFs anymore: Now the Φ-score depends on the weights of both the follower and the leader party.

5.3.1. Degree Bounded Trees

We are now going to describe how to compute the desired δ-values. To simplify the presentation we first restrict ourselves to the case where the input tree is degree-3-bounded. Later we will argue (see Section 5.3.2) how the algorithm can be modified to work with general trees.

The main algorithm divides the input tree recursively into two-terminal subtrees (2TSs). Given two terminal nodes u and v recall that the 2TS T_{uv} is

the unique maximal subtree T_{uv} of T that contains u and v as leaves. Confer also Section 4.3.3.

Since we are dealing with a degree-3-bounded tree the *valid subdivision rule* of Lemma 4.3.3 yields now a bounded number of 2TSs.

Lemma 5.3.2 *Let T_{uv} be a 2TS, let m be the unweighted median of T_{uv}, and let m' be the projection of m onto path $P(u,v)$. Then the valid split set $\{m, m', u, v\}$ induces at most five new 2TSs to which we refer as* child 2TSs of T_{uv}.

Proof. This property follows easily from Lemma 4.3.3 and the degree-3-boundedness. Confer also Figure 4.8. □

Our algorithm computes, for each 2TS S that occurs during the recursive division and for each edge (u,v) of S, a *restricted δ-value* that is defined as

$$\delta_u^S(v) := \sup\{\, d(u,y) \mid \text{node } y \in T_u(v) \cap S \text{ and } \varphi(y,u) > \varphi_0 \,\}. \tag{5.4}$$

The restricted value $\delta_u^S(v)$ denotes the maximum distance to any node dominating u, where only nodes from the 2TS S are taken into account as valid opposition nodes. During the execution of the algorithm we propagate the restricted δ-values from single edges (at the bottom of the recursion) towards larger 2TSs ending up with the input tree, where the unrestricted δ-values according to (5.3) will have been computed.

Let S be a 2TS with child 2TSs S_i and assume that the restricted δ^{S_i}-values have already been computed. In order to determine the δ^S-values for the parent 2TS we essentially have to update the δ^{S_i}-values with a loosened restriction that an opposition may also place within a child $S_j \neq S_i$: For any edge $(u,v) \in S_i$ we have the relation

$$\delta_u^S(v) = \max\{\delta_u^{S_i}(v)\} \cup \left\{ d(u,y) \;\middle|\; \begin{array}{l} \text{node } y \in S_j \text{ for some } S_j \subseteq T_u(v) \\ \text{and } \varphi(y,u) > \varphi_0 \end{array} \right\}.$$

In the remainder of this section we will describe how all these values can be computed at once efficiently in total linear time $O(|S|)$.

To this end we consider each pair (S_i, S_j), $S_i \neq S_j$, and handle the case where the leader locates in S_i and the follower in S_j (the opposite direction is symmetric). The above operation is supported by lists stored in the child 2TSs, and the update of the restricted δ-values can be carried out for each edge in a single traversal of these lists.

For each 2TS $S := T_{st}$ with terminals s, t we maintain two lists $L^S(s), L^S(t)$. We describe the construction of $L^S(t)$ only, as the other list is defined symmetrically. The list $L^S(t)$ contains for each edge $e = (u,v) \in T_{st}$ (directed

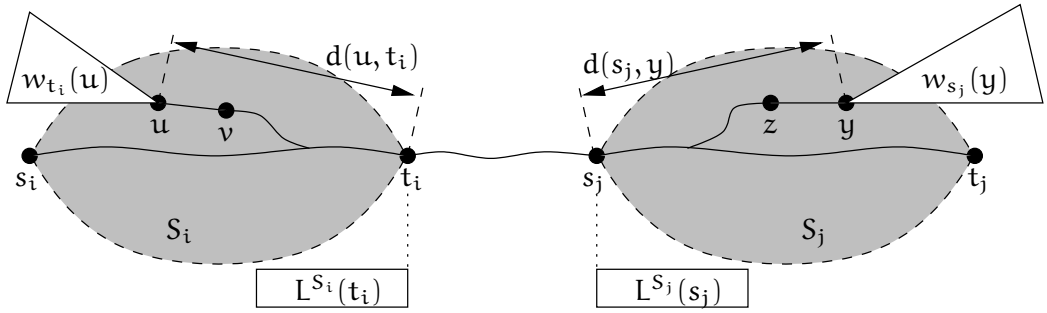

FIG. 5.4.: Situation in algorithm COMPUTEDELTAVALUES.

such that v is the endpoint of e closest to t) the entry $\big(e, d(u,t), w_t(u)\big)$. The lists are kept sorted by the weights.

If the 2TS S contains only one edge, $e = (s,t)$, the list is initialized with one element: $L^S(t) := \langle (e, d(s,t), w_t(s)) \rangle$. The restricted δ-values are initialized as follows:

$$\delta^S_s(t) := \begin{cases} d(s,t) & \text{if } \varphi(t,s) > \varphi_0, \\ -\infty & \text{otherwise}. \end{cases}$$

If the 2TS S consists of more than one edge and is subdivided into child 2TSs S_i (with terminals s_i, t_i), we consider each pair (S_i, S_j) of distinct children and traverse their lists $L^{S_i}(t_i)$ and $L^{S_j}(s_j)$ in parallel and in descending order with respect to the weights. (The situation is illustrated in Figure 5.4 and the details of this algorithm are drawn out in Figure 5.5.) We maintain two pointers to an edge (u,v) in the list $L^{S_i}(t_i)$ and to an edge (z,y) in list $L^{S_j}(s_j)$, respectively, with the invariant that y is the least-weighted node still dominating u, that is, $\varphi(y,u) > \varphi_0$. Moreover we store in d_{max} the largest distance encountered in the list $L^{S_j}(s_j)$ so far. In other words, the value d_{max} describes the maximum distance from s_j to a node in S_j that dominates u. Whenever the pointer in $L^{S_j}(s_j)$ reaches the least weighted node dominating u, we update the value $\delta^S_u(v) := \max\{\, \delta^S_u(v), d(u,t_i) + d(t_i,s_j) + d_{max} \,\}$ and advance the pointer in $L^{S_i}(t_i)$. This way, (that is, by a single traversal of both child lists) all values in the parent list are updated, and the running time for handling a pair (S_i, S_j) is linear in the size of S_i and S_j. Since the number of children is constant the execution of Algorithm COMPUTEDELTAVALUES in Figure 5.5 is in fact $O(|S|)$.

The list $L^S(t)$ of the parent S consists of a merge of the lists $L^{S_i}(t_i)$ of all children S_i, where in each entry $(\cdot, d(u,t_i), \cdot)$ the distance is updated to $d(u,t) := d(u,t_i) + d(t_i,t)$. (Here we assume that the terminal nodes s_i, t_i of S_i are ordered such that t_i is the terminal node closest to t.) This compu-

input: a 2TS S
for all child 2TSs S_i of S
  initialize $\delta^S_u(v) \leftarrow \delta^{S_i}_u(v)$ and $\delta^S_v(u) \leftarrow \delta^{S_i}_v(u)$ for all $(u,v) \in S_i$
for each pair S_i, S_j of different child 2TSs
  let $t_i \in S_i$, $s_j \in S_j$ be the terminals closest to each other
  assume $w_{t_i}(u_1) \geq \ldots \geq w_{t_i}(u_l)$ for edges $(u_k, v_k) \in L^{S_i}(t_i)$
  assume $w_{s_j}(y_1) \geq \ldots \geq w_{s_j}(y_{l'})$ for edges $(z_k, y_k) \in L^{S_j}(s_j)$
  initialize $k \leftarrow 1$, $k' \leftarrow 1$, $d_{max} \leftarrow -\infty$
  while $k \leq l$
    while $k' \leq l'$ **and** $\varphi(y_{k'}, u_k) > \varphi_0$
      $d_{max} \leftarrow \max\{d_{max}, d(s_j, y_{k'})\}$
      $k' \leftarrow k' + 1$
    $\delta^S_{u_k}(v_k) \leftarrow \max\{\delta^S_{u_k}(v_k), d(u_k, t_i) + d(t_i, s_j) + d_{max})\}$
    $k \leftarrow k + 1$
output: restricted values $\delta^S_u(v)$ for all $(u,v) \in S$

FIG. 5.5.: Algorithm COMPUTEDELTAVALUES.

tation can be implemented by linear traversals of the lists of type $L^{S_i}(t_i)$ in a merge-sort like manner, as they have already been presorted in the preceding recursion level.

Theorem 5.3.3 *For any monotonic gain function Φ, any degree-3-bounded n-node tree, and any bound φ_0, the set of all φ_0-bounded solutions can be computed in time $O(n \log n)$.*

Proof. For proving the correctness it remains to show that the values computed by the algorithm comply with the definition given in (5.4). This can be easily shown by structural induction over the depth of the decomposition into 2TSs.

We claim that the running time of the algorithm is $O(n \log n)$. This follows from the fact that the running time $T(n)$ on an n-node tree can be bounded by

$$T(n) = c \cdot n + \sum_{i=1}^{r} T(n_i) \leq c \cdot n \cdot \log_2 n$$

where c is a constant (describing the effort for dividing the problem into and merging it from subproblems of size $n_i \leq n/2$ each) and $r \leq 5$ is the number of subproblems. The size bound $n_i \leq n/2$ is guaranteed since we use a median node for splitting the current subtree. □

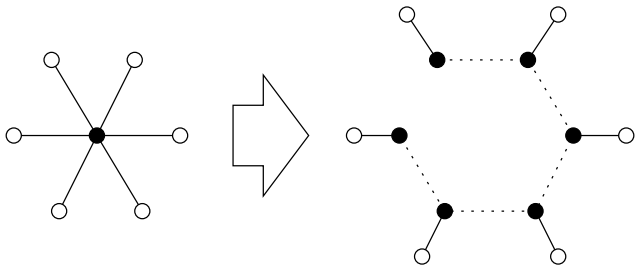

FIG. 5.6.: Reduction of a general tree to a degree-3-bounded tree by node splitting. Dotted lines represent zero-length edges.

5.3.2. General Trees

We are now going to loosen the temporary restrictions imposed in the previous section. If an input tree is not degree-3-bounded we can enforce this property by splitting nodes of larger degree and inserting *zero-length* edges. This is can be accomplished successively in the following way: If u is a node with neighbors $v_1, \ldots, v_k$ and $k \geq 3$, we replace u by a path $u_1, u_2, \ldots, u_k$ of length zero and connect u_i with v_i by an edge of length $d(u, v_i)$. We set $w(u_1) := w(u)$ and $w(u_i) := 0$ for all $i \geq 2$. Confer Figure 5.6. This construction increases the size of T by a factor of at most two. Now we can employ the algorithm of the preceding section to compute the δ-values in the modified tree, which takes time $O(n \log n)$. We actually want, however, to determine the δ-values for the *original* tree. To this end, we observe that any edge (u, v) in the original tree has a uniquely determined representative (u', v') in the modified tree. We simply set $\delta_u(v) := \delta_{u'}(v')$ to retrieve the desired δ-values.

There seems, however, to be a flaw in this approach at first glance: Until now we have assumed that our input tree contains no zero-length edges (confer Section 1.2). If the tree does contain zero-length edges then the conclusions drawn from Lemma 5.3.1 are no longer true. This is due to the fact that for a given leader-follower pair (x, y) with $y \in N_\alpha(x)$ Lemma 4.2.5 does not need to hold, that is, the leader party $U(x \prec y)$ could contain nodes not lying in $T_y(x)$ (namely those with distance zero to x). Thus, there can be "wrong" δ-values in the modified tree. Nevertheless, this does not invalidate the correctness: It is not hard to see that only the zero-length edges themselves are affected by this problem. The edges corresponding to edges in the original tree have always positive length and their δ-values can be taken from the corresponding edges in the expanded tree, whereas the values of the added zero-length edges are simply ignored.

Corollary 5.3.4 *For any monotonic gain function Φ, any n-node tree and any bound φ_0 the set of all φ_0-bounded solutions can be computed in time $O(n \log n)$. In particular, the same holds for the set of Φ-solutions.* □

In what follows we discuss some simplifications for specific monotonic gain functions and shed some light on how far our approach can be applied to the discrete case.

5.3.3. Strong Φ-Solutions

Let's make the assumption that the input tree does not have zero-weighted nodes of degree less than three. As argued in Section 4.5 we do not loose any relevant information if we drop zero-weighted leaves. As regards zero-weighted nodes of degree two, one can observe that any two edges incident with such a node can be glued together to one edge without distorting the distance relations between the remaining nodes.

Consider the security MGF Δ. As shown in Section 5.2 there is a lower bound of $\Omega(n \log n)$ for the complete exploration of the set of all Δ-solutions. However, the first phase of the algorithm, namely the computation of the optimal security score Δ^*, becomes trivial since $\Delta^* = 0$ holds for each tree.

If we proceed to the *strong* security score Δ' the case $\Delta'(x) < 0$ can indeed occur. Moreover, the solution set does not necessarily contain a weighted median as demonstrated in Figure 4.1. While this complicates the computation of the optimal Δ'-score, that is, the first phase of our algorithm, the exploration phase is now much simpler since the solution set cannot contain inner points of more than one edge: For, if x_1 and x_2 are inner points of different edges, each node u on path $P(x_1, x_2)$ must have a strong security score $\Delta'(u) < \max\{\Delta'(x_1), \Delta'(x_2)\}$ (confer proof of the next theorem).

These considerations carry over to all functions φ that are *strictly monotonic in the second parameter*.

Theorem 5.3.5 *Let $\varphi\colon \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$ be a function increasing in the first and strictly decreasing in the second parameter. Let Φ be the induced monotonic gain function. Then the set of all* strong *Φ-solutions is a closed segment of one single edge and can be determined in linear time.*

Proof. Let x_1, x_2 be inner points of distinct edges and u be a node on path $P(x_1, x_2)$. Let y be a witness of u, that is, $\Phi(y \prec u) = \Phi(u)$. One of the paths $P(u, x_1)$ and $P(u, x_2)$ intersects $P(u, y)$ only at u. Assume that this applies to

x_1. Then $w_y(x_1) < w_y(u)$ and $w_{x_1}(y) = w_u(y)$ due to our assumption that all zero-weighted nodes have degree at least three. Hence

$$\Phi(x_1) \geq \Phi(y \prec x_1) \geq \varphi(y, x_1) > \varphi(y, u) = \Phi(u)\,,$$

which shows the first claim.

To prove the second claim consider an optimal point x. We are going to show that it is possible to identify the edge (u, v) containing the solution set in linear time. This establishes the claim since once we know edge (u, v) we are able to compute $\delta_u(v)$ as well as $\delta_v(u)$ in linear time and thus can determine the solution set by the interval test (5.2).

If x is an inner point of an edge (u, v) we are done. So let x be a node. Compute an optimal follower location y and a neighbor $v \in N(x)$ such that y is not contained in the subtree $T_v(x)$. This is possible in $O(n)$. Owing to the strict monotonicity of φ all points $x' \neq x$ in $T_v(x)$ have a score $\Phi(x') \geq \Phi(y \prec x') > \Phi(y \prec x) = \Phi(x)$. This shows that the solution set is a subset of the edge (x, v) and completes the proof. □

5.3.4. Comparing Absolute and Discrete Model

Let's take a look at the discrete case where leader and follower place their facilities only at nodes of the tree. There seems to be no obvious way to adapt our approach to this model. This is surprising at first glance since one could conjecture that a solution arising from discrete model is merely a subset of the absolute solution set constructed by intersecting with the node set. This conjecture, however, is not true: The example in Figure 5.7 shows that in fact it is possible that the absolute strong security set and the discrete strong security set may be arbitrary many nodes apart from each other. For a given $k \in \mathbb{N}$, the graph consists of $2k + 3$ nodes. All but one node (with weight $2k-1$) have unit weight. In the absolute case, the leader places at the node x_{abs} with a leader party weight of $2k + 1$ and a follower party weight of 1, resulting in a strong security score of $-2k$. In the discrete case, the leader places at x_{discr} and gains $4k$ while the follower gains 1, resulting in a strong security score of $1 - 4k$.

What prevents us from using the recursive dividing algorithm described in the previous sections for the discrete model is that the leader party $U(x \prec y)$ may contain nodes outside the subtree $T_y(x)$ hanging from leader node x; in fact it can be a subtree with a root node on the path $P(x, y)$. Determining these subtrees is treated in Section 4.4.

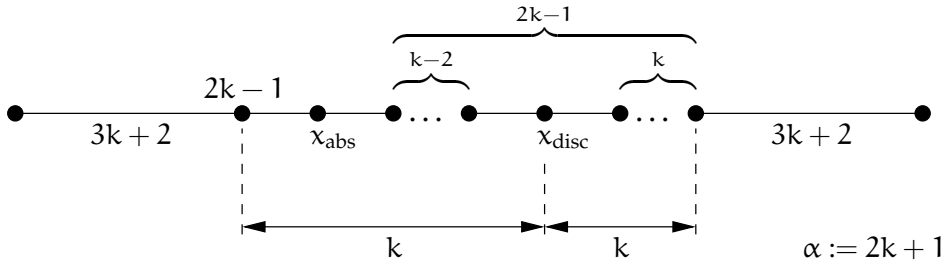

FIG. 5.7.: Example where the discrete and the absolute strong security solution set are separated by arbitrarily many nodes. Node weights and edge lengths are 1 unless otherwise indicated.

If we, however, restrict our attention to leader independent gain functions, the structure of the leader party is clearly no longer important. This enables us to formulate Theorem 5.1.2 also for the discrete model.

Theorem 5.3.6 *Let Φ be a leader independent monotonic gain function and φ_0 be a bound. Then the set of discrete φ_0-bounded Φ-solutions can be computed in linear time. In particular, this holds for the set of all Φ-solutions.*

Proof. We claim that the approach of Section 5.1 works also for determining the set of discrete φ_0-bounded solutions. It is easy to observe that in the case of leader independence the discrete and the absolute Φ-scores of a node are identical which shows the first claim.

As a consequence, if the absolute optimum found by the algorithm suggested in Chapter 4 falls on a node then it is also a discrete optimum. If it falls on an inner point of an edge then by the monotonicity property one of the endpoints of that edge must be a discrete optimum. The Φ-scores of these two endpoints can be determined in linear time. This completes the proof. □

5.4. Computing All φ_0-Tolerant Solutions on Trees

Recall that a point x is called φ_0-tolerant for an MGF Φ and a fixed bound φ_0 if x is in the set $S(\alpha^*(\varphi_0), \varphi_0)$ (confer Definition 3.5.12). In other words, we are looking for a solution where α has the smallest possible value such that a φ_0-bounded solutions exists. This has to be compared with the problem of finding Φ-solutions where we fix α and try to minimize φ_0.

As we have demonstrated in Section 3.5, the family $(S(\alpha, \varphi_0))_{\alpha,\varphi_0}$ is *not* always monotonic with respect to α on general graphs. In contrast, monotonicity does hold for arbitrary MGFs on *trees* as a consequence of Lemma 5.3.1: Observe that the definition of the δ-values is *independent* of α. Hence, if we increase α the set of points satisfying the condition in this Lemma, that is, $S(\alpha, \varphi_0)$, increases inclusion-wise.

Lemma 5.4.1 *On a tree, the family of sets $(S(\alpha, \varphi_0))_{\alpha,\varphi_0}$ is monotonically increasing with respect to both parameters α and φ_0 for an arbitrary MGF Φ.* □

This result suggest that minimizing α on trees might lead to "well-structured" problems that admit efficient algorithms, similar to the minimization of φ_0 (that is, finding Φ-solutions).

In this section we investigate the computation of all φ_0-tolerant solutions of a tree for a given $\varphi_0 \geq \varphi(0,0)$. According to Lemma 5.3.1 a point $x \in T$ is φ_0-bounded if and only if $\delta_x(v) \leq \alpha$ for all $v \in N(x)$. For the definition of $\delta_x(v)$ confer Equation (5.3) on page 89. Let us now fix a point x and define the value

$$\alpha^*(x) := \max\big(\{\, \delta_x(v) \mid v \in N(x) \,\} \cup \{0\}\big)\,.$$

It is clear that x is in $S(\alpha, \varphi_0)$ if and only if $\alpha \geq \alpha^*(x)$. Observe that in the above definition of $\alpha^*(x)$ values $\delta_x(v) = -\infty$ might occur. This is why we include the additional zero into the right-hand expression. We obtain immediately:

$$\alpha^*(\varphi_0) = \min_{x \in T} \alpha^*(x)\,.$$

Assume that we have determined the δ-values with the help of the algorithm of Section 5.3, which takes $O(n \log n)$ time. Then we can compute $\alpha^*(x)$ for any *node* $v \in V$ in $O(\deg(v))$ time. Hence, to determine $\alpha^*(\cdot)$ for *all* nodes $v \in V$ we need time $O(n)$ once the δ-values are known.

Now we are going to determine for every edge (u, v) a segment containing exactly those inner points with minimum $\alpha^*(\cdot)$ among all inner points of (u, v). To this end let x be an inner point of (u, v). Then $\delta_x(v) = \delta_u(v) - d(u, x)$ and $\delta_x(u) = \delta_v(u) - d(u, v) + d(u, x)$. Observe that $\alpha^*(x) = \max\{\delta_x(u), \delta_x(v), 0\}$. Therefore, our desired segment consists exactly of all inner points x such that the distance $d(u, x)$ minimizes the piecewise linear and convex function

$$f(z) := \max\{\delta_u(v) - z, \delta_v(u) - d(u, v) + z, 0\}\,.$$

Determining those segments for all edges takes time $O(n)$. Hence the set of all points or segments minimizing $\alpha^*(\cdot)$ can be determined in total linear time when all δ-values have been computed. Since the δ-values can be determined in $O(n \log n)$ for general and in $O(n)$ for leader independent MGFs this yields:

Theorem 5.4.2 *The set of (absolute) φ_0-tolerant solutions of an n-node tree can be determined in $O(n \log n)$. If the underlying MGF is leader independent this takes linear time.* □

Note that Lemma 5.3.1 also holds for the discrete case as long as the underlying MGF is leader independent. Hence we can apply the above approach also to this case:

Corollary 5.4.3 *The set of discrete φ_0-tolerant solutions of a tree for a leader independent MGF can be computed in linear time.*

5.5. Characterization of φ_0-Tolerant Solutions for Leader Independent MGFs

In this section we will take a closer look at φ_0-tolerant solutions of *leader independent* MGFs (again, we assume $\varphi_0 \geq \varphi(0)$). As pointed out in the previous section the set of those solutions can be computed in linear time. Nevertheless it is worth investigating the structural properties of this set. In what follows we present a more general view on the problem and a relation to centers on trees. This allows us to derive an algorithm for computing *efficient* Φ-solutions (confer Section 3.5). Our considerations also lead to a simpler algorithm for computing φ_0-tolerant solutions and we conjecture it would also be faster in practical implementations.

We define a digraph G_T for the input tree T as follows: For each (undirected) edge (u, v) in T we add the (directed) arc (u, v) to G_T if $\varphi(w_u(v)) > \varphi_0$. In other words, the arc (u, v) indicates that the subtree $T_u(v)$ is heavy enough to dominate all points lying outside. We call such a subtree $T_u(v)$ *qualified*. Note that the construction of G_T allows also adding pairs of antiparallel arcs. We call a strongly connected component of G_T *nontrivial* if it contains more than one node.

Lemma 5.5.1 *The digraph G_T contains at most one nontrivial strongly connected component.*

Proof. Assume we have two such strongly connected components. Let (u, v), (v, u) be a pair of antiparallel arcs in the first component, x be a node in the second component. Without loss of generality, $x \in T_u(v)$. Since by definition, $T_v(u)$ is qualified, this holds also for each super-tree, hence u can be reached from x in graph G_T. By interchanging the role of the two components we show also that x is reachable from u which completes the proof. □

Lemma 5.5.2 *Let G_T contain a nontrivial strongly connected component C_T. Then a point x is in $S(\alpha, \varphi_0)$ if and only if its eccentricity $ecc(x)$ in C_T is at most α.*

Proof. If $ecc(x) > \alpha$ then there is an arc (u, v) in C_T such that $T_u(v)$ is qualified and $d(x, v) > \alpha$. Then v dominates x. Let $ecc(x) \leq \alpha$. Since all arcs outside the component C_T are directed towards C_T itself, tree $T_x(v)$ is not qualified for any node $v \notin C_T$. Thus no node with distance $> \alpha$ can dominate x. □

Lemma 5.5.3 *If G_T contains a nontrivial strongly connected component C_T, then $\alpha^*(\varphi_0)$ equals the radius of C_T and the set of φ_0-tolerant solutions coincides with the center of C_T. Otherwise, $\alpha^*(\varphi_0) = 0$ and the median of T is a φ_0-tolerant.*

Proof. The first statement has been shown in the previous considerations. Assume now that G_T has no nontrivial components. A node u is φ_0-tolerant if and only if the outdegree $\deg^+(u) = 0$, for, otherwise it would be dominated by its neighbor. Since G_T does not contain a nontrivial strongly connected component, there are at most $n-1$ arcs and hence there exists at least one node with this property. □

Theorem 5.5.4 *The set of φ_0-tolerant solutions for a leader-independent MGF on a tree can be computed in $O(n)$.*

Proof. The algorithm starts by computing G_T and C_T in linear time. If there is a nontrivial component C_T, it is actually a cactus and the center of the underlying tree can be computed in linear time as described in [Han73]. Otherwise, we refer to Theorem 5.1.2. □

5.6. Computing All Φ-Solutions on Trees

In this section we develop an algorithm for determining the set of all efficient Φ-solutions in trees. Recall that a pair is efficient if there is a pair

(α, φ_0) such that $x \in S(\alpha, \varphi_0)$ but $S(\alpha - \varepsilon, \varphi_0)$ and $S(\alpha, \varphi_0 - \varepsilon)$ are empty for any $\varepsilon > 0$.

We use a similar construction as before. To this end we now replace the cactus C_T with the underlying subtree. Let's start with the smallest possible value $\varphi_0 = \varphi(0)$. With continuously increasing φ_0, C_T shrinks in general. Let α be the radius of C_T, that is, $\alpha^*(\varphi_0)$. Obviously, α is a step-wise monotonically decreasing function of φ_0. At each point where α decreases, (α, φ_0) must be an efficient pair.

Observe that the changes in C_T appear only at those values of φ_0 where φ_0 attains the value

$$\tilde{\varphi}(e) := \min\{\varphi(w_u(v)), \varphi(w_v(u))\}$$

for some edge $e := (u, v)$. In other words, $\tilde{\varphi}(e)$ is the smallest value of φ_0 for which edge e is not contained in C_T.

Hence we do not need to compute C_T each time from scratch: Instead, we can update C_T by removing those edges with smallest value of $\tilde{\varphi}$. To this end, we need to compute $\tilde{\varphi}$ in a preprocessing step (which can be performed in linear time), and then sort the edges according to $\tilde{\varphi}$.

In order to update the set of φ_0-tolerant (or equivalently the center) of the tree C_T efficiently, we make use of results by Alstrup et al. [AHT00]. They provide a data structure, called *top trees*, that allows us to maintain the center of a dynamic tree in logarithmic time. In particular, they show that the center and the radius of an n-node tree can be updated in $O(\log n)$ time per edge removal. The preprocessing time for building the top tree is $O(n)$.

This yields the algorithm depicted in Figure 5.8 and the following result.

Theorem 5.6.1 *Given a leader independent MGF Φ and an n-node tree, the set of efficient (absolute) Φ-solutions can be computed in $O(n \log n)$ time.*

Proof. The correctness for the case of a non-trivial component C_T has been argued above. If φ_0 induces only trivial components then $\alpha^*(\varphi_0) = 0$ according to Theorem 5.5.3. Hence the Φ-solutions for $\alpha = 0$ are the corresponding efficient Φ-solutions.

For the running time observe that any edge is deleted at most once. Hence there are $O(n)$ edge removals each of which consumes time $O(\log n)$ for updating center and radius of C_T. □

As in the previous cases, there is no significant difference between the absolute and the discrete model for leader independent MGFs. We only have to maintain discrete instead of absolute centers of component C_T. Thus

for leader independent MGFs the set of efficient discrete Φ-solutions can also be determined in $O(n \log n)$.

```
1   φ_0 ← φ(0)
2   let α ← ∞
3   compute C_T for φ_0
4   compute φ̃-values for all edges of C_T
5   sort edges increasingly according to their φ̃-value
6   while C_T contains at least one edge
7       if the radius of C_T is smaller than α then
8           output the center of C_T
9           α ← radius of C_T
10      delete from C_T all edges with minimum φ̃-value
11      in doing so, maintain center and radius of C_T
12  end while
13  output all Φ-solutions for α = 0
```

FIG. 5.8.: Algorithm for computing all efficient Φ-solutions of a tree.

6. Summary and Further Remarks

6.1. Summary

In the first part of this thesis we have considered single location problems from the area of competitive and voting location. First, in Chapter 3, we introduced a new concept called *monotonic gain functions* that unifies all particular optimization problems under investigation. In Chapter 4 we have studied the efficient computation of one Φ-solution and related problems on trees. In particular we have given in Section 4.3 a linear time algorithm for computing the Φ-score of a tree. In Chapter 5 we have also analyzed the complexity of exploring the complete set of all φ_0-bounded solutions. For the case of leader independent MGFs we have presented a linear time algorithm. As a byproduct this brings down the running time of $O(n^3 \log n)$ for computing all Stackelberg solutions on trees with parametric prices [GP03] to $O(n)$. For general monotonic gain functions Φ we have provided an algorithm to compute the set of all φ_0-bounded solutions with running time $O(n \log n)$ and a matching lower bound proving that this running time is optimal. We have also shown that on trees, the set of φ_0-tolerant Φ-solutions can be obtained in $O(n \log n)$ and $O(n)$ for arbitrary and leader independent MGFs, respectively. For leader independent MGFs Φ we have developed an $O(n \log n)$ algorithm for computing the set of efficient Φ-solutions. Figure 6.1 gives an overview of the most important complexity results known for monotonic gain functions.

6.2. Remarks on General Graphs

In this section we discuss an observation concerning the computation of Φ-solutions on general graphs. This observation justifies that the situation on general graphs might be harder than on trees.

The fastest published algorithm [HL88] for computing an (unrelaxed)

problem	property of MGF	placement of leader and follower	
		absolute (at points)	discrete (at nodes)
Φ-score		$O(n)$ (optimal) [Thm. 4.3.14]	$O(n \log^2 n)$ [Thm. 4.4.4]
set of Φ-solutions	general	$O(n \log n)$ (optimal) [Cor. 5.3.4] [Thm. 5.2.2]	$O(n^2 \log n)$ [enumeration]
	strong	$O(n)$ (optimal) [Thm. 5.3.5]	
	leader independent	$O(n)$ (optimal) [Thm. 5.1.2]	$O(n)$ (optimal) [Thm. 5.3.6]
φ_0-tolerant Φ-solutions	general	$O(n \log n)$ [Thm. 5.4.2]	$O(n^2 \log n)$ [enumeration]
	leader independent	$O(n)$ (optimal) [Thm. 5.4.2]	$O(n)$ (optimal) [Cor. 5.4.3]
efficient Φ-solutions	leader independent	$O(n \log n)$ (optimal) [Thm. 5.6.1]	$O(n \log n)$ (optimal) [Thm. 5.6.1]

TABLE 6.1.: Overview of the complexity bounds obtained in first part of this thesis for MGF-related problems on trees.

absolute Simpson solution of a general graph has a running time of $O(|V|^4|E|^2 \log(|V||E|) \log w(G))$ which, albeit polynomial, might be too time-consuming for practical purposes. To the best of our knowledge no further improvements in asymptotic running time have been made since then.

The discrete case has been considered by Campos and Moreno [CM03]. Notice that Campos and Moreno use a slightly different model where users and locations belong to possibly different, finite sets U and L, respectively. Furthermore user weights and a distance function $d\colon U \times L \to \mathbb{Q}_0^+$ are specified. They give a polynomial algorithm for computing the set $C(\alpha, \gamma)$ of (α, γ)-Condorcet solutions. The running time of their algorithm is $O(|U||L|^2)$. Applying this to our scenario where $n = |V| = |U| = |L|$, we obtain a performance of $O(n^3)$. Although their algorithm is quite simple and enumerative in nature we will give some evidence that developing a significantly faster algorithm is at least no easy task. In particular, we can show that computing $C(\alpha, \gamma)$ on a graph is as least as hard as solving the *vector maximization problem*. We remark that vector maximization should not be confused with *vector optimization* in the usual mathematical programming context.

The vector maximization problem determines for a given set S of n vectors in $\mathbb{R}^k$ the set of maximal vectors. A vector $u = (u_1, \ldots, u_k)$ is called *maximal* if there is no $v = (v_1, \ldots, v_k) \in S$ such that $u \neq v$ and $u_i \leq v_i$ for all $1 \leq i \leq k$. This problem was introduced in [KLP75] and is well investigated in the literature (see [GSG04] for a comprehensive survey) due to its fundamental nature and a large body of applications. From our result it follows that an algorithm for computing $C(\alpha, \gamma)$ with running time significantly faster than $O(n^3)$ as stated in [CM03] would imply an algorithm for the vector maximization problem that is also faster than the currently known algorithms.

Theorem 6.2.1 *If $t(\cdot, \cdot)$ is a function such that there is an algorithm computing the set $C(\alpha, \gamma)$ of discrete (α, γ)-Condorcet solutions in time $O(t(|U|, |L|))$ for the model of [CM03], then this implies an algorithm for the k-dimensional vector maximization problem with running time $O(k|S| \log |S| + t(k, |S|))$. In particular, this holds for the unrelaxed case.*

Proof. We describe an algorithm for the vector maximization problem that calls a subroutine for computing $C(\alpha, \gamma)$. To this end, let $S = \{x_1, \ldots, x_n\}$ be a set of n k-dimensional vectors. For each $1 \leq i \leq k$ let $\lhd_i$ denote a lexicographic order of S such that the i-th coordinates of the vectors have the highest priority. For $1 \leq i \leq k$ and $1 \leq j \leq n$ let $\nu(i, j) := |\{ x_r \in S : x_j \lhd_i x_r \}|$.

The values of ν can be computed in $O(kn \log n)$ time. Now we introduce a set $U = \{u_1, \ldots, u_k\}$ of users and set $L := S$. We connect each pair u_i, x_j by an edge of length $d(u_i, x_j) := \nu(i, j)$. We set $w \equiv 1$, $\gamma := \frac{n-1}{n}$ and $\alpha := 0$.

Observe that a node $x \in L$ is dominated by some node $y \neq x$ if and only if $d(u_i, y) < d(u_i, x)$ for all $u_i \in U$. To put it in the language of vector maximization, x is dominated if and only if $y \lhd_i x$ for all i, or equivalently, if and only if $y_i \leq x_i$ for all i where $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$.

From the above considerations it follows that the set $C(\alpha, \gamma)$ is exactly the set of maximal vectors of S. By introducing an additional user u_{k+1} with an appropriate weight, we can even assume that $0 < \gamma < 1$ is fixed to some value. □

The naive approach for solving the vector maximization problem is to compare all pairs of vectors, which leads to a running time of $O(k|S|^2)$. To the best of our knowledge, there are no algorithms (see [GSG04] for a comprehensive survey) with a better asympotic running time if k is part of the input. For fixed k, algorithms with running time $O(|S| \log^{k-2} |S|)$ have been developed [KLP75]. But clearly, this does not supersede the naive approach if k is part of the input.

In terms of computing the set of Condorcet solutions this means that an algorithm being faster than the straightforward, enumerative approach of Campos and Moreno [CM03] would imply also a faster algorithm for the well-studied vector maximization problem. This is one of the motivations why we have not concentrated on sequential location problems for general graphs but on the more promising special case of tree networks.

6.3. Threshold Functions

In this section we introduce *threshold functions* as a concept that covers both the additive indifference suggested by Campos and Moreno (see Section 3.4) as well as obvious generalizations, for example, multiplicative indifference. We can show that most of the basic properties mentioned by Campos and Moreno [CM03] and in Section 3.5 continue to hold in the generalized model.

A function $\delta \colon \mathbb{Q}_0^+ \times \mathbb{Q}_0^+ \to \mathbb{Q}_0^+$ is called a *threshold function* if $\delta(0, y) = 0$ for all $y \in \mathbb{Q}_0^+$ and δ is (weakly) monotonically increasing in both parameters. Given a threshold function δ and a parameter $\beta \geq 0$, we replace the definition of relaxed user preference stated in (3.1) on page 46 to the general for

$$U(y \prec x) := \{\, u \in V \mid d(u,y) < d(u,x) - \delta(\beta, d(u,x)) \,\} \tag{6.1}$$

Examples: The additive α-indifference introduced in [Slo78] and [CM03] is modeled by the threshold function $\delta\colon (\beta, d) \mapsto \beta$ with parameter $\beta := \alpha$. We can also model a multiplicative indifference where a user is undecided if the *ratio* of the distances to two facilities does not exceed a factor α by choosing the threshold function $\delta\colon (\beta, d) \mapsto \beta \cdot d$ and $\beta := \alpha - 1$.

Replacing the additive indifference (3.1) by (6.1) we can define monotonic gain functions, Φ-score, Φ-solutions and φ_0-bounded solutions also for this extended preference model. The set of φ_0-bounded solutions for some fixed threshold function δ and a given indifference parameter β is denoted by $S(\beta, \varphi_0)$.

6.4. Basic Properties of Threshold Functions

In Sections 3.4 and 3.5 we have considered basic properties that hold for (α, γ)-Condorcet and MGFs, respectively. It is not hard to see that monotonicity and Φ-minimality also hold for the generalized indifference model:

Lemma 6.4.1 (Monotonicity) *The family $(S(\beta, \varphi_0))_{\beta,\varphi_0}$ of φ_0-bounded Φ-solutions is monotonically increasing with respect to φ_0. If Φ is leader independent this family is also monotonically increasing with respect to β.*

Lemma 6.4.2 (Φ-Minimality) *For any $\beta \in \mathbb{Q}_0^+$, there is a smallest φ_0, denoted by $\Phi^*(\alpha)$ such the set $S(\beta, \varphi_0)$ is not empty.* □

While minimizing the second parameter, φ_0, is possible for all threshold functions, this does not hold in general for the first parameter, β. To see this consider, for example, the multiplicative threshold function $\delta\colon (\beta, d) \mapsto \beta \cdot d$. Since the function value vanishes for zero-distances, we can easily construct instances where β-minimality does not hold. Consider for example a graph consisting of one single edge with unit-weighted nodes. Then for any $\beta \in \mathbb{Q}_0^+$ there is no $(\beta, \frac{1}{3})$-Condorcet solution since any point x is dominated by some node y different from x.

In a certain sense we need to require threshold functions to attain a threshold value large enough for creating indifferent solutions in every situation. To this end, we call a threshold function δ *invertible* if the value

$$\min\{\, \beta \geq 0 \mid \delta(\beta, d) \geq c \,\}$$

is well defined for all $c, d \in \mathbb{Q}_0^+$. Then we have the following result:

Lemma 6.4.3 (β-Minimality) *For all invertible threshold functions and each $\varphi_0 \geq \varphi(0,0)$, the value*

$$\beta^*(\varphi_0) := \min\{\, \beta \in \mathbb{Q}_0^+ \mid S(\beta, \varphi_0) \neq \emptyset \,\} \qquad \square$$

is well defined.

We remark that while the multiplicative threshold function is not invertible, for each $\varepsilon > 0$ the function

$$\delta(\beta, d) := \beta \cdot \max\{d, \varepsilon\}$$

is invertible and comes closest to the multiplicative threshold function.

6.5. Future research

The results of the first part of this thesis suggest two future research directions. On the one hand, one could try to obtain non-trivial results for graph classes beyond trees. On the other hand, it would be interesting to extend the existing results to more powerful and more realistic location models.

Some specific questions are the following. Are there more efficient algorithms for handling the *discrete* location case on trees? This concerns the computation of the Φ-score as well as the computation of all Φ-solutions. It would also be interesting to investigate the computation of efficient Φ-solutions for arbitrary MGFs on trees. More complex graph classes that could be examined are tree-width bounded, planar and Euclidean graphs.

Our algorithms for the additive indifference model on trees do not immediately carry over to general threshold functions since then the parties of the competitors do not need to be subtrees hanging from some node which is a basic ingredient of our algorithms. Thus it would be worth investigating this more general scenario.

Another promising research direction would be to study competitive location models based on *attraction functions* that depend not only on the distance but also on the specific location or other factors such as workload, purchase prices or transportation cost. Finally, one could also investigate scenarios where the prices are not fixed but part of the decision making of the competitors. It is also reasonable to assume that the competitors can *design* their facilities by varying certain parameters that have an impact on the attraction of the facility. For an overview of competitive location models we refer to Eiselt et al. [ELT93].

Part II.

Multiple Location

7. Multiple Competitive Location on General Graphs

7.1. Introduction and Problem Definition

In the second part of this thesis we will examine *multiple* competitive and voting location problems. These problems are natural generalizations of their pendants in single location considered in the first part. In multiple *competitive*, location we allow leader and follower to place more than one server each, that is, the placements of the competitors are (finite) point sets. In multiple *voting* location, the candidate placements are point sets of equal size.

In the case of single location problems our research was motivated by the existing linear-time algorithms for trees under the unrelaxed model. Our goal was to find similar results also for the more general and also more realistic, relaxed model (confer monotonic gain functions in Section 3.5). Although the relaxed case has turned out to be substantially more involved, we were able to develop fast and for the central problems even optimal algorithms.

In contrast, we will see that the computation of optimal solutions in the multiple location scenario is inherently difficult even under the unrelaxed model. Therefore our goal is no longer to add more complexity to the model (with the intention to obtain more realistic models) but rather to examine the boundary of efficient solvability and approximability by considering simpler and more special problem instances. For this reason, we will restrict ourselves to the investigation of the unrelaxed model and the most basic sequential location problems, namely medianoid and centroid (or in the voting parlance Simpson and Condorcet, respectively).

There are several possibilities to extend our single location models to the multiple case. For example, we might require the competitors to place their servers alternately. Such a problem has been investigated by Chawla et al. [CRRS06] with respect to the existence of solutions in the n-dimensional Euclidean space. From a computational point of view, problems with alter-

nately moving players and variable number of moves are often PSPACE-hard [GJ79] and thus extremely difficult.

We consider a *two-stage* scenario: The leader selects a specified number p of locations for his servers. Afterwards the follower places r servers. The number r is known by the leader in advance, and the follower knows the exact positions of the leader's servers. As in the single case, both competitors aim at maximizing their own benefit corresponding to the weight of users they serve.

As in the single case we will always assume that we are given an undirected graph $G = (V, E)$ with positive edge lengths $c\colon E \to \mathbb{Q}^+$ and non-negative node weights $w\colon V \to \mathbb{Q}_0^+$. Recall that the edge lengths induce a metric $d\colon G \times G \to \mathbb{Q}_0^+$ of shortest path distances on the set of points of the graph.

In order to define medianoid and centroid in the multiple location scenario, we first need to adapt our notions of distances and preferences (confer Definition 3.1.1 on page 39) appropriately for *sets* of locations: Let $X, Y \subset G$ be finite sets of points, specifying a server placement of the leader and the follower player, respectively. The distance of a user $u \in V$ to a finite point set M is given by $d(u, M) := \min_{m\in M} d(u, m)$. The user u prefers the follower if $d(u, Y) < d(u, X)$. By $U(Y \prec X)$ we denote the set of users preferring Y over X. We call this user set Y-*party*. The weight of the Y-party is denoted by $w(Y \prec X) := w(U(Y \prec X))$. X-party and $U(X \prec Y)$ are defined analogously.

We are now ready to define problem of the follower:

Definition 7.1.1 ((r, X_p)-Medianoid) Let $r, p \in \mathbb{N}$ and let $X_p \subset G$ be a set of p points. Then

$$w_r(X_p) := \max_{\substack{Y_r \subset G \\ |Y_r| = r}} w(Y_r \prec X_p)$$

is the maximum influence any r-element follower placement can gain over the fixed leader placement X_p. An *absolute* (r, X_p)-*medianoid* of the graph is any set $Y_r \subset G$ of $|Y_r| = r$ points where $w(Y_r \prec X_p) = w_r(X_p)$ is attained.

The problem of the leader is formulated in terms of the gain achievable by the follower:

Definition 7.1.2 ((r, p)-Centroid) Let $r, p \in \mathbb{N}$. Then

$$w_{r,p} := \min_{\substack{X_p \subset G \\ |X_p| = p}} w_r(X_p)$$

is the minimum gain any leader placement must leave to the follower. An *absolute* (r,p)*-centroid* of the graph is any set $X_p \subset G$ of p points where $w_r(X_p) = w_{r,p}$ is attained.

The above definitions of absolute (r,X_p)-medianoid and absolute (r,p)-centroid refer to point sets on the input graph. The computational problems of *determining* an absolute (r,X_p)-medianoid and an absolute (r,p)-centroid are *optimization problems*. We will use the terms absolute (r,X_p)-medianoid and absolute (r,p)-centroid also for these optimization problems and as well for the corresponding *decision problems* as the meaning will become clear from the context.

The notions *discrete* (r,X_p)*-medianoid* and *discrete* (r,p)*-centroid* are defined similarly, with the server sets restricted to nodes, that is, $X_p, Y_r \subseteq V$, rather than points.

As in the single-location case, competitive location problems have their counterparts in voting location. *Multiple* voting location has first been considered by Campos and Moreno [CM08]. Besides the introduction of this problem field, the authors give exact exponential time algorithms based on integer programming techniques. In comparison to single voting location, in the multiple case candidates of the virtual election are always p-element location sets. A p-*Simpson solution* can be defined as a (p,p)-centroid. A p-*Condorcet solution* is a p-element point set X such that $w_p(X) \leq \frac{1}{2}w(G)$. The set X is a p-γ-*Condorcet solution* if $w_p(X) \leq \gamma \cdot w(G)$. Again the intention of voting location is to formalize some notion of social stability: For a p-Condorcet solution there is no opposing candidate placement being preferred by a majority of the users. The p-Simpson problem tries to minimize the influence of opposing candidates.

Besides stating some basic discretization results on absolute (r,p)-centroid and (r,X_p)-medianoid, this chapter is primarily devoted to the analysis of the complexity and the approximability of these problems on *general* graphs.

The NP-hardness of (r,p)-centroid and (r,X_p)-medianoid has been shown by Hakimi [Hak90]. Additionally, Hakimi proves that there is no constant-factor approximation algorithm for (r,p)-centroid unless $P = NP$. However, the results of Hakimi leave several questions open: First, it is easy to see that discrete (r,X_p)-medianoid is in NP: Given any bound W, simply guess an r-element node set Y_r and verify that $w(Y_r \prec X_p) \geq W$. In the case of (discrete) (r,p)-centroid there is no straightforward, efficiently verifiable certificate proving (or disproving) that $w_{r,p}$ is smaller than a specified bound. This is mainly because checking whether $w_r(X_p) \leq W$ holds for a

given point set X_p and a given bound W is already NP-hard (as follows from the hardness of the follower problem). Indeed, Hakimi [Hak90] states that (r,p)-centroid seems to be "exceedingly difficult". We will justify Hakimis conjecture by showing that the decision problem of (r,p)-centroid is in fact Σ_2^p-complete. Recall that Σ_2^p is the class of problems decidable in polynomial time by a non-deterministic turing machine with access to an oracle for NP. It is widely believed that Σ_2^p is a proper superset of NP. A definition of the class Σ_2^p can also be found in Section 1.2.

But also the approximability of multiple competitive location problems is not completely resolved by Hakimi's results. To the best of our knowledge no upper or lower bound regarding the approximability of (r,X_p)-medianoid has been published. We will show that there is an approximation-preserving reduction from (r,X_p)-medianoid to the well known maximum coverage problem which yields a tight bound of approximability.

The constant-factor lower bound of Hakimi [Hak90] for (r,p)-centroid does not rule out the existence of good input-dependent approximation ratios such as $O(\log n)$. We will, however, sharpen Hakimi's result by showing that for no $\varepsilon > 0$ there is an $(n^{1-\varepsilon})$-approximation algorithm for (r,p)-centroid unless $P = NP$. Roughly speaking, there is probably no efficient algorithm with a "reasonable" worst-case approximation factor.

7.2. Relations Between Absolute and Discrete Model

From an algorithmic point of view we face the problem of an infinite solution space in the absolute case. In the single case on trees we resolved this for the follower problem by means of the α-neighborhood (confer Definition 4.2.3). The leader problem was tackled with the help of critical points (confer Section 4.3.4) which could be determined efficiently.

Megiddo et al. [MZH83] introduce so called *boundary points* as a concept allowing a discretization of the absolute (r,X_p)-medianoid problem. We remark that the authors consider a more general scenario, called *maximum coverage location* problem (confer Definition 8.1.1 on page 132), of which (r,X_p)-medianoid is a special case. Given a point set X_p a point z is a *boundary point* if there is a node u such that $d(u,z) = d(u,X_p)$. Santos-Peñate et al. [SSD07] call z a (u,X_p)-*isodistant point*. In the sequel we will adopt the latter terminology. Moreover, we will call a point X_p-*isodistant* if it is (v,X_p)-isodistant

for some $v \in V$.

Such (v, X_p)-isodistant points are of particular importance: They constitute exactly the boundary of the connected point set of all positions where the follower claims the node v. Hence the gain of the follower is constant within each interval limited by X_p-isodistant points. Indeed, Megiddo et al. observe that for any point y within any open edge segment containing no X_p-isodistant points the party $U(y \prec X_p)$ is the same. Moreover, they observe that any edge has $O(n)$ many X_p-isodistant points and thus that the set B of all X_p-isodistant points has polynomial size $O(n|E|)$. Now consider the connected components of $G-(B \cup V)$ each of which is an open edge segment. From each of these segments we choose one arbitrary representative. Let C be the set, called *candidate set*, which consists of all such representatives, all boundary points and all vertices. According to the above considerations we can conclude that there is an (r, X_p)-medianoid Y_r with $Y_r \subseteq C$. Moreover, the set C has polynomial size and can be computed in polynomial time.

Theorem 7.2.1 (Discretization (r, X_p)-medianoid [MZH83, SSD07]) *Given an instance of absolute (r, X_p)-medianoid we can compute in polynomial time a finite point set containing an optimal follower placement.* □

Santos-Peñate et al. [SSD07] note that similar discretization results are not known for the more sophisticated (r, p)-centroid problem on general graphs. We will justify this in Sections 9.1 and 9.2 by showing that it is not possible at all to identify such a finite set in polynomial time unless $P = NP$. This holds even for the very simple case of a path. It will, however, turn out convenient to have at least some *finite* (but possibly exponentially large) candidate set at hand. To this end, we will assume that all edge lengths are integral which can be achieved by scaling all rational edge lengths with their lowest common denominator. Note that this can be carried out with polynomial effort.

Theorem 7.2.2 (Discretization of (r, p)-centroid) *Let G be a graph whose edge lengths are positive integers. Then there is an (r, p)-centroid X_p for G such that $d(x, v) \in \frac{1}{2}\mathbb{N}$ for each $x \in X_p$ and each vertex v.*

Proof. We assume without loss of generality that all edges have unit length, which can be achieved by creating zero weighted nodes at an integer grid. Now let X_p be an (r, p)-centroid.

We transform X_p into a new set X'_p by moving each point to the nearest node, unless the point is the midpoint of an edge. Notice that each point

moves by less than $\frac{1}{2}$ by this transformation. Therefore, all isodistant points move by less than $\frac{1}{2}$.

We show that $w_r(X'_p) \leq w_r(X_p)$. Assume the contrary. Then there must be an interval between two X'_p-isodistant points where the follower gains a set of nodes that was not present in the original instance. This means that there must be a pair (i_1, i_2) of two isodistant points on an edge which has interchanged its relative position during the transformation. More exactly, if i_1, i_2 are the distances of the points to one fixed endpoint of the edge before the transformation, and i'_1, i'_2 are the positions after the transformation, then we must have $i_1 \geq i_2$ and $i'_1 < i'_2$. Obviously i'_1, i'_2 are either endpoints or midpoints, that is, $i'_1, i'_2 \in \{0, \frac{1}{2}, 1\}$ (where for the sake of an easier presentation we identify points with their respective distances).

If one of those points, say i'_1, is a midpoint then the point has not moved at all, that is, $i_1 = i'_1$. This implies that point i_2 has moved by at least $\frac{1}{2}$ which is impossible. On the other hand, if both i'_1, i'_2 are endpoints, the total sum of the movement is at least 1 which is again a contradiction. This shows the claim. □

We have seen how to *discretize* the absolute follower and the absolute leader problem: We are able to identify finite candidate sets that are guaranteed to contain an optimal solution. In a certain sense, these results can be considered as a *reduction from the absolute to the discrete case*. From an algorithmic point of view it will turn out that the discrete variant is easier to attack than the absolute one. Therefore, the foregoing discretization results will help derive *upper bounds* on the complexity also in the absolute case.

In the following sections we will pursue the contrary goal: We will try to develop *lower bounds* on the complexity and the approximability of (r, p)-centroid and (r, X_p)-medianoid. Again, it will be easier to formulate our arguments for the discrete case first. Thus, in order to extend these results also to the absolute case, we need a reduction in the reverse direction, that is, *from the discrete to the absolute case*.

The following result shows that we can construct for any discrete instance of the leader problem an absolute instance such that the respective values of $w_{r,p}$ differ only insignificantly. Since we are going to deal with different problem instances of (r, p)-centroid we will use superscript notation $w^I_{r,p}$, $w^I_r(X_p)$ and $w^I(Y_r \prec X_p)$ to emphasize the dependence of these values from the problem instance I.

Theorem 7.2.3 *Let* k *be some fixed positive integer. Then we can efficiently construct for any instance* I *of discrete* (r, p)*-centroid an instance* I' *of absolute*

(r,p)-*centroid such that* $|w^{I}_{r,p} - w^{I'}_{r,p}| \leq \varepsilon$ *and* $w(G) = w(G')$. *Here,* $G = (V,E)$ *is the graph of instance* I, G' *is the graph of* I' *and* $\varepsilon = \max_{v\in V} w(v)/n^k$.

Proof. The graph G' we are going to construct consists of two layers: The first layer contains exactly the node set V. Each node of the first layer has weight zero. The second layer contains for any node $v \in V$ a collection of $l := \max\{p,r\} \cdot n^k$ nodes $v_1, \ldots, v_l$ each of weight $w(v_i) := w(v)/l$. Let $u, v \in V$ be (possibly equal) nodes of graph G. Then we connect u in the first layer with each v_i in the second layer by an edge of length $d(u,v_i) = \operatorname{diam}(G) + d_G(u,v) + 1$. This completes the construction of the graph G'. It is clear that $w(G) = w(G')$. Note that only nodes of the second layer have positive weights. Thus only these nodes are relevant for the respective user parties. Moreover we remark that any placement for the discrete instance I may also be viewed as a placement in the absolute instance I' within the first layer. By observing that the graph G' is in fact metric, it follows easily that $w^I(Y_r \prec X_p) = w^{I'}(Y_r \prec X_p)$ for all sets $X_p, Y_r \subseteq V$.

Let X_p and Y_r be leader and follower placements in I', respectively. Assume that $y \in Y_r - V$ is a point that is not part of the first layer. Point y lies on some edge (u,v_i). Now assume that $s_j \neq v_i$ is some node of the second layer preferring y over X_p. Since $d_{G'}(v_i,s_j) \geq 2 \cdot \operatorname{diam}(G) + 2 > d_G(u,s) + \operatorname{diam}(G) + 1 = d_{G'}(u,s_j)$ any shortest path between s_j and y meets u. Thus, if we replace the follower server y with u, the gain of the follower decreases by at most $\frac{\varepsilon}{r}$, namely the weight of the node v_i. Performing this transformation for any point in $Y_r - V$ we arrive at a set $\tau(Y_r)$ of first-layer nodes such that $w^{I'}(\tau(Y_r) \prec X_p) \geq w^{I'}(Y_r \prec X_p) - \varepsilon$. Symmetrically, we obtain that $w^{I'}(Y_r \prec \tau(X_p)) \leq w^{I'}(Y_r \prec X_p) + \varepsilon$. Since the nodes of the first layer are exactly the nodes of our original graph G the transformation $\tau(\cdot)$ may be seen as mapping placements of the absolute instance I' to placements of the discrete instance I. The above inequalities show that the absolute error of this transformation is small.

We are now going to show that $|w^I_{r,p} - w^{I'}_{r,p}| \leq \varepsilon$. To this end let X^I_p and $X^{I'}_p$ be (r,p)-centroids of I and I', respectively. Moreover let $Y^{I'}_r$ be an $(r, \tau(X^{I'}_p))$-medianoid. Then

$$\begin{aligned} w^{I}_{r,p} &= w^{I}(X^{I}_{p}) \\ &\leq w^{I}_{r}(\tau(X^{I'}_{p})) \\ &\leq w^{I'}_{r}(\tau(X^{I'}_{p})) = w(Y^{I'}_{r} \prec \tau(X^{I'}_{p})) \\ &\leq w^{I'}_{r}(Y^{I'}_{r} \prec X^{I'}_{p}) + \varepsilon \\ &\leq w^{I'}_{r,p} + \varepsilon \end{aligned}$$

To show that $w^{I}_{r,p}$ is not smaller than $w^{I'}_{r,p} - \varepsilon$, consider an (r, X^{I}_{p})-medianoid $Z^{I'}_{r}$ in the absolute instance I'. We obtain:

$$\begin{aligned} w^{I'}_{r,p} &\leq w^{I'}_{r}(X^{I}_{p}) = w^{I'}(Z^{I'}_{r} \prec X^{I}_{p}) \\ &\leq w^{I'}(\tau(Z^{I'}_{r}) \prec X^{I}_{p}) + \varepsilon \\ &\leq w^{I}_{r}(X^{I}_{p}) + \varepsilon \\ &= w^{I}_{r,p} + \varepsilon \end{aligned}$$

□

7.3. Complexity of the Leader Problem

Hakimi [Hak83] proved the NP-hardness of (absolute and discrete) $(1, p)$-centroid, that is, of the leader problem for $r = 1$. The corresponding decision problem, which asks whether there is a p-element point set X_p such that $w_1(X_p) \leq W$ for a given bound W, is in NP and hence also NP-complete. This follows from the fact that an $(1, X_p)$-medianoid of a p-element point set X_p can be determined in polynomial time (confer Section 8.2). Indeed, a non-deterministic turing machine may guess a p-element point set and then compute a $(1, X_p)$-medianoid Y_1. Finally, it can verify in polynomial time whether or not $w_1(X_p) = w(Y_1 \prec X_p)$ exceeds W. Note that guessing X_p in the absolute case includes the application of the discretization result stated in Theorem 7.2.2. More precisely, each point in X_p is determined by an edge and a half-integral distance to one of its end nodes. These considerations can be straightforwardly generalized to arbitrary $r \geq 1$ as long as r is fixed, that is, if it is not part of the input. Again, this is due to the polynomial-time solvability of (r, X_p)-medianoid for fixed r [Hak90].

The situation becomes more involved if r is part of the input. In this case, the computation of an (r, X_p)-medianoid itself is already NP-hard [Hak83]

and there is no obvious alternative way to construct witnesses proving that $w_{r,p} \leq W$ for a given bound W. In other words it is not clear whether (r,p)-centroid is in NP at all. Hakimi [Hak90] conjectures that the (r,p)-centroid problem seems to be "exceedingly difficult".

One may object that it is quite a theoretical question whether or not (r,p)-centroid is in NP since its hardness is already known. But it can be argued that this question very well has practical implications. In fact, the knowledge that a given problem is an NP-optimization problem (that is, its decision problem is in NP) allows us to apply various practical, heuristic approaches. We remark that many heuristics are based on the fact that the goal function of the underlying problem is polynomially computable. This concerns for example the greedy strategy as well as many meta-heuristics such as hill climbing, tabu search, simulated annealing, genetic algorithms etc. Thus, if we were able to formulate (r,p)-centroid as an NP-optimization problem we could immediately apply all these general tools for attacking NP-hard optimization problems. Unfortunately, it will turn out that such a formulation probably does not exist. We will show that (r,p)-centroid is Σ_2^p-complete, which rules out that it is in NP unless the polynomial-time hierarchy collapses to NP, which is widely believed to be false.

The class Σ_2^p is part of the polynomial time hierarchy (confer Section 1.2) and contains all decision problems that can be described using a logical formula of the form $\exists x\, \forall y\ \varphi$ where φ is a quantifier-free formula [SU02].

We will prove the Σ_2^p-completeness of (p,p)-centroid, which implies the Σ_2^p-completeness of the decision problems for p-Simpson and p-γ-Condorcet. The hardness proof uses a reduction from the decision problem $\exists\forall$3SAT. We briefly recall some preliminaries. Let X denotes a set of logical (that is, binary) variables. If $x \in X$ is a variable, then $\neg x$ is its negation. The set $\{x, \neg x \mid x \in X\}$ is the set of *literals*. A *term* is a conjunction of literals. A formula is in *disjunctive normal form* (DNF) if it is a disjunction of terms. A *formula in 3-DNF* is a disjunction of terms where each term consists of exactly 3 literals. An *assignment* X_0 for a set X of variables is a mapping $X_0 \colon X \to \{0,1\}$. If all variables of a formula φ are defined this way, the truth value of the whole formula is determined and denoted by $X_0(\varphi)$. If X_0 is an arbitrary assignment, then $\overline{X_0} \colon x \mapsto 1 - X_0(x)$ is the assignment generated from X_0 by flipping the truth value of each variable.

The decision problem $\exists\forall$3SAT is defined as follows: Let $X = \{x_1, \ldots, x_{|X|}\}$ and $Y = \{y_1, \ldots, y_{|Y|}\}$ be disjoint sets of logical variables and let φ be a 3-DNF over $X \cup Y$, decide whether the formula $\exists x_1 \ldots \exists x_{|X|}\ \forall y_1 \ldots \forall y_{|Y|}\ \varphi$ is satisfied, where $x_i \in X$ and $y_j \in Y$.

Theorem 7.3.1 ([SU02]) *The problem* $\exists\forall 3\text{SAT}$ *is* Σ_2^p*-complete.* □

Let $|\varphi|$ denote the number of terms in φ. A variable x_i is called *trivial* in φ if φ contains either x_i or $\neg x_i$ but not both.

Corollary 7.3.2 $\exists\forall 3\text{SAT}$ *is* Σ_2^p*-complete, even if* $|X| = 2|\varphi| + |Y|$ *and* φ *contains no trivial variables.*

Proof. We show that the known hardness result from above continues to hold even with the additional restrictions. If φ contains a trivial variable, say x_i or $\neg x_i$, then add a new term $x_i \wedge \neg x_i \wedge z$ where z is any literal distinct from x_i and $\neg x_i$. To enforce $|X| = 2|\varphi| + |Y|$, add yet unused variables to X or to Y respectively, which are nontrivial by definition. □

Theorem 7.3.3 *Discrete* (r, p)*-centroid is* Σ_2^p*-complete. This holds even when the underlying graph is bipartite and* $r = p$.

Proof. First, we convince ourselves that (r, p)-centroid is in Σ_2^p. To this end, we consider an instance of (r, p)-centroid which consists of a node and edge-weighted graph along with numbers r, p and a bound W. The task is to decide whether $w_{r,p} \leq W$ by means of a non-deterministic polynomial time turing machine having access to an oracle for NP. First, we guess an (r, p)-centroid X_p non-deterministically. Then we consult our oracle for NP to check whether $w_{r,p} = w_r(X_p) \leq W$. Here, we exploit that the decision problem for (r, X_p)-medianoid is in NP. The time needed for this computation is clearly polynomially bounded.

Now, we show that (r, p)-centroid is also complete for Σ_2^p. To this end we use a reduction from $\exists\forall 3\text{SAT}$ where the additional requirements of Corollary 7.3.2 are fulfilled. Let φ be a respective formula and let X, Y be defined as above.

For a literal l of φ let $\nu(l)$ denote the number of appearances of l in φ. We set $M := \max\{\nu(x_i), \nu(\neg x_i) \mid x_i \in X\}$, $N := \sum(\nu(y_i) + \nu(\neg y_i))$ and $W := m'M + 3|\varphi| + |Y| + N - 1$, where m' is the number of used variables from X.

The graph constructed in the sequel consists of several subgraphs that we call *components*. These components will then be combined to the graph of our instance for (r, p)-centroid. In doing so, we will guarantee that the mentioned subgraphs only intersect at dedicated nodes called *terminal nodes*. In contrast, *non-terminal nodes* are part of only one component.

We use two main types of components: An s*-diamond* has two terminal nodes t_1, t_2 and s further nodes $v_1, \ldots, v_s$, where each v_i is connected both

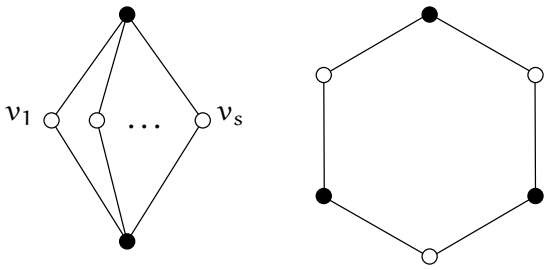

FIG. 7.1.: An s-diamond to the left and a term component to the right. Terminals are marked black and non-terminals are marked white.

to t_1 and t_2. A *term component* is a cycle C_6 where three pairwise nonadjacent nodes are terminals. Confer Figure 7.1. The total graph is composed by connecting diamonds and term components at terminal nodes. All edges in diamonds and term components as well as all non-terminal nodes are unit-weighted, whereas the terminal nodes are zero-weight.

For each variable $x_i \in X$, let $C(x_i)$ be a $(W+2)$-diamond, called the *variable component*, with terminal nodes x_i and $\neg x_i$. Similarly, add a 1-diamond $C(y_i)$ for each $y_i \in Y$. For each term $c = l_1 \wedge l_2 \wedge l_3$ in φ, add a *term component* with terminal set $\{l_1, l_2, l_3\}$. Now connect each of these terminal nodes l_i by a new diamond $L_c(i)$ to the corresponding terminal node in the variable component. We adjust the size of these connecting diamonds so that the degree of each used literal l over X in the variable components is $M+W+2$; obviously this can be achieved, for example, by choosing exactly one $(M - \nu(l) + 2)$-diamond and all remaining diamonds as 1-diamonds.

Now, connect the variable nodes x_1 and $\neg x_1$ by edges of length 2 to all non-adjacent non-terminal nodes in the graph. This completes the construction of the graph. Note that every edge connects a terminal and a non-terminal node. Hence the graph is bipartite. Finally, we set $r := p := |X| = 2|\varphi| + |Y|$. Confer Figure 7.2.

The idea of the proof is to establish a relationship between assignments of the variables of the $\exists\forall 3$SAT instance on the one hand and p-element node subsets of the (r, p)-centroid instance on the other hand. Throughout the following proof we identify assignments and node subsets in the natural way, that is, that a terminal node of a variable component is part of the subset if and only if the corresponding literal is set to 1.

A p-element node subset X_p is called a *valid X-placement* if it contains for each variable $x_i \in X$ exactly one of the terminal nodes $x_i, \neg x_i$ of the corresponding variable component; a valid Y-placement Y_r is defined in a similar way. Note that a valid X-placement can be treated as an X-assignment in the

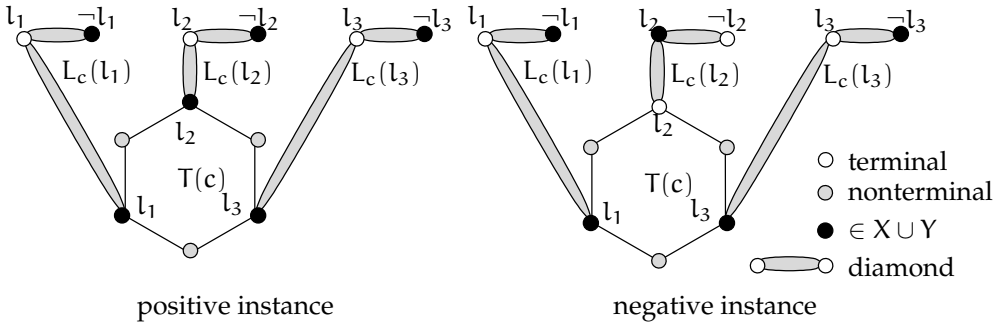

FIG. 7.2.: Illustration of the reduction. Note that a literal l_i may denoted a negated variable in which case $\neg l_i$ denotes the non-negated variable.

aforementioned sense. The converse also holds.

We claim that any X_p with $w_r(X_p) \leq W$ must be a valid X-placement. Indeed, if there is a variable component $C(x_i)$ such that none of its terminals is occupied by X_p then each of these terminals is preferred over X_p at least $W + 1$ non-terminals of $C(x_i)$.

We call all non-terminal nodes v with $d(X_p, v) > 1$ *follower nodes*.

We state the following observation: If X_p is a valid X-placement, then there is a Y_r where all non-terminal nodes preferring Y_r over X_p are actually follower nodes. This can be shown as follows: First observe that since only non-terminal nodes have positive weight, terminal nodes do not have any impact on the value $w(Y_r \prec X_p)$. Second, let Y_r be an opposition and let v be a non-terminal node in $U(Y_r \prec X_p)$ which is *not* a follower node. Then $v \in Y_r$, and v is preferred by no node other than v itself. Thus we can replace v by any yet unused follower node without decreasing the value $w(Y_r \prec X_p)$.

Let X_p be a leader placement with $w_r(X_p) > W$. As before there is a Y_r such that all non-terminals in $U(Y_r \prec X_p)$ are follower nodes. From $w(Y_r \prec X_p) \geq W + 1$ it follows that Y_r must be preferred over X_p by *all* follower nodes. Thus each term component must contain at least two nodes from Y_r. On the other hand, each variable component $C(y_i)$ must contain at least one node from Y_r. Since term and variable components do not have common nodes we can deduce the following observation: For a valid X-placement X_p we have $w_r(X_p) > W$ if and only if there is a valid Y-placement Y_r such that:

(i) each variable component $C(y_i)$ contains exactly one node from Y_r,
(ii) each term component contains exactly two nodes from Y_r, and
(iii) each literal component contains at least one node from $X_p \cup Y_r$.

This means that (as follows by simply counting the available facilities) each

node from Y_r is either part of a variable component or a term component, in particular it cannot be a non-terminal node in a literal component.

We complete the proof by showing that the instance of $\exists\forall 3\text{SAT}$ is satisfied if and only $w_{r,p} \leq W$ in the (r,p)-centroid instance constructed.

To this end, let X_p be an assignment for the variables in X such that the formula $\forall y\ \varphi$ is satisfied. Then we claim that $\overline{X_p}$ satisfies $w_r(X_p) \leq W$ (confer Fig. 7.2 left): For the sake of a contradiction, consider a leader placement Y_r satisfying conditions (i)–(iii) stated above. Then $(X_p \cup \overline{Y_r})(\varphi) = 1$. Consequently there exists a term $c = l_1 \wedge l_2 \wedge l_3$ such that $(X_p \cup \overline{Y_r})(l_i) = 1$ for all i. Hence, the complementary assignment $\overline{X_p} \cup Y_r$ does not contain any of l_1, l_2, l_3 in the variable components. To satisfy condition (iii), Y_r must contain all three terminal nodes of $T(c)$, which contradicts condition (ii).

Conversely, assume that $\exists x\,\forall y\ \varphi$ is not true. Let X_p be an arbitrary valid X-placement (confer Figure 7.2 right). Then there exists an Y_r such that $(\overline{X_p} \cup Y_r)(\varphi) = 0$, hence for all terms $c = l_1 \wedge l_2 \wedge l_3$ there is at least one literal l_i such that $(\overline{X_p} \cup Y_r)(l_i) = 0$. Then the set $Y_r' := \overline{Y_r}$ can be completed to form an follower placement that dominates X_p at the end: Since $(X_p \cup \overline{Y_r})(l_i) = 1$, we can add the two terminals l_j, $j \neq i$, of the term component $T(c)$ to Y_r'. After this has been performed for all terms, conditions (i)–(iii) from above are satisfied for the set Y_r'; hence $w(Y_r' \prec X_p) > W$. □

We observe that the reduction used in the above proof also works for the absolute case:

Corollary 7.3.4 *Absolute (r,p)-centroid is Σ_2^p-complete even when the underlying graph is bipartite and $r = p$.*

Proof. First, we claim that absolute (r,p)-centroid is in Σ_2^p. To this end, we employ the discretization result stated in Theorem 7.2.2: We guess an (r,p)-centroid X_p non-deterministically. In doing so, each point of X_p is either a node or can be described by an edge and the half-integral distance to one of its end nodes. Hence a polynomial number of non-deterministic bits is sufficient to guess X_p. Then we consult our oracle for NP to check whether $w_{r,p} = w_r(X_p)$ exceeds the bound specified in the instance of the decision problem. Here, we exploit that the decision problem for absolute (r, X_p)-medianoid is in NP as pointed out above.

In order to show that absolute (r,p)-centroid is complete for Σ_2^p we employ Theorem 7.2.3. Let I be the instance constructed in the proof of Theorem 7.3.3. Let us now construct some instance I' of absolute (r,p)-centroid by choosing $k = 1$ in Theorem 7.2.3. The theorem guarantees that the absolute error of this reduction is strictly smaller than $1/n$ which is without

loss of generality smaller than $\frac{1}{3}$. Therefore, if $w^{I}_{r,p} \leq W$ then $w^{I'}_{r,p} \leq W + \frac{1}{3}$ and if $w^{I}_{r,p} \geq W + 1$ then $w^{I'}_{r,p} \geq W + \frac{2}{3}$. Thus $w^{I}_{r,p} \leq W$ if and only if $w^{I'}_{r,p} \leq W + \frac{1}{3}$. □

Since the decision problems of (p, p)-centroid and of p-Simpson are both equivalent to p-γ-Condorcet, we obtain immediately.

Corollary 7.3.5 *Absolute and discrete* p*-Simpson and* p*-*γ*-Condorcet are* Σ^{p}_{2}*-complete.* □

In the statement of the corollary the parameter γ is part of the input. One may ask for the complexity of the problem if γ is fixed. Note that if γ comes close to 0 (resp. 1) the restriction imposed on any p-γ-solution X becomes very strong (resp. weak). In fact, if $\gamma \in \{0, 1\}$ the problem is (trivially) decidable in polynomial time. One might even suspect, that for large enough $\gamma < 1$ a p-γ-Condorcet solution always exists.

The following theorem shows that this intuition is wrong: The above mentioned trivial cases $\gamma = 0$ and $\gamma = 1$ are the only ones where p-γ-Condorcet can be decided in polynomial time.

Theorem 7.3.6 *If* $P \neq NP$*,* p*-*γ*-Condorcet is efficiently solvable if and only if* $\gamma \in \{0, 1\}$.

Before proving this theorem, it should be pointed out that its statement is a significant sharpening of the so-called *Condorcet paradox*. Remember the introductory example depicted in Figure 1.2 at the very beginning of this thesis. The example shows an election with three candidates A, B, C none of them being a Condorcet winner. Specifically, the situation is as follows: Candidate A dominates candidate B, that is, an absolute majority of two thirds of the voters prefers A over B, which suggests that A is significantly better than B. But there seems to be an even better candidate since C is preferred over A by two thirds of the voters, too. So C should be better than A and all the better than B. The latter, however, is *not* true: B dominates C and not the other way round.

This paradox can easily be generalized to elections with arbitrarily large numbers of candidates and voters where the candidates constitute a large cycle along which they dominate each other. In doing so, the fraction of users preferring a candidate over the next one can be brought arbitrarily close to 1 by increasing the number of voters and candidates.

The proof of the theorem is based exactly on this idea. We are going to construct a graph in which the preference structure resembles “cyclic”

elections like those mentioned above. However, our proof shows not only that there are arbitrarily bad voting location instances but establishes also the NP-hardness of the corresponding decision problem. To this end, we will "embed" an instance of the well-known NP-complete 3-DIMENSIONAL MATCHING problem [GJ79] into a cycle-shaped graph and boost the number of nodes until we achieve the desired fraction γ.

Proof (of Theorem 7.3.6). Let's start with the discrete case. We reduce from the NP-complete 3-DIMENSIONAL MATCHING problem (3DM for short), which is defined as follows. Let M be a subset of $R \times S \times T$ where R, S, T are pairwise disjoint sets of cardinality q. Decide whether M has a *perfect matching*, i. e. a set $M' \subseteq M$ such that $|M'| = q$ and the elements of M' pairwise do not have common coordinates.

Given an instance I of 3DM and positive integers m, k such that $m \geq |M| + 3q$ we construct an instance $I(m, k)$ of p-γ-Condorcet in the following way (confer Figure 7.3): The underlying graph $G = (V, E)$ of $I(m, k)$ consists of three sets V_{in}, V_{mid} and V_{out} of nodes. The cardinalities of these node sets are $|V_{out}| = 3qk$ and $|V_{in}| = |V_{mid}| = m$, respectively. By construction we guarantee that $M \cup R \cup S \cup T \subseteq V_{mid}$ holds. The elements of V_{mid}, V_{in} are indexed somehow. Note that any element $u \in M \cup R \cup S \cup T$ can be identified with some $x_{i_u} \in V_{mid}$, where i_u is a suitable index in $\{0, \ldots, m-1\}$. The set V_{out} contains for each $u \in R \cup S \cup T$ exactly k elements $u_0, \ldots, u_{k-1}$. Now let E be the union of $V_{out} \times V_{mid}$ and $V_{mid} \times V_{in}$. Define the lengths of the edges in E as follows: Let $u \in R \cup S \cup T$. Then u corresponds to a node $x_{i_u} \in V_{mid}$ and to some node set $\{u_0, \ldots, u_{k-1}\} \subset V_{out}$. Let x_j be an arbitrary element of V_{mid}. If $x_j \in M$ and u is a coordinate of x_j then $c(u_l, x_j) := m$ else $c(u_l, x_j) := m + ((j - i_u) \mod m)$ for all $0 \leq l \leq k-1$. For $x_j \in V_{mid}$ and $v_i \in V_{in}$ we set $c(v_i, x_j) := m + ((j - i) \mod m)$. While the nodes in $V_{out} \cup V_{in}$ have unit weight, all nodes in V_{mid} have weight zero. Finally, let $p := q$ which completes the construction. Confer Figure 7.3 for an illustration.

Let X, Y be p-element placements. Observe that only the nodes in $V_{out} \cup V_{in}$ have an impact on the value $w(Y \prec X)$ since all nodes in V_{mid} are zero weighted. Hence we call the nodes in $V_{out} \cup V_{in}$ *essential*. Now let $0 \leq i \leq m-1$ and $x_i \in V_{mid}$. We define $i' := (i-1) \mod m$. Verify that $x_{i'}$ is preferred over x_i by all essential nodes whose distance to x_i exceeds m.

Now let M' be a perfect matching of the 3DM instance I and consider $X := M'$ as a candidate placement in $I(m, k)$. We claim that $w(Y \prec X) \leq m$ for any p-element node set Y. To this end let Y_{io} be the set of essential nodes in Y and let $y \in Y_{io}$. Observe that y is the only essential node in $U(y \prec X)$. It follows that $w(Y_{io} \prec X) \leq p$. Now let $Y_{mid} := Y \cap V_{mid}$. Since

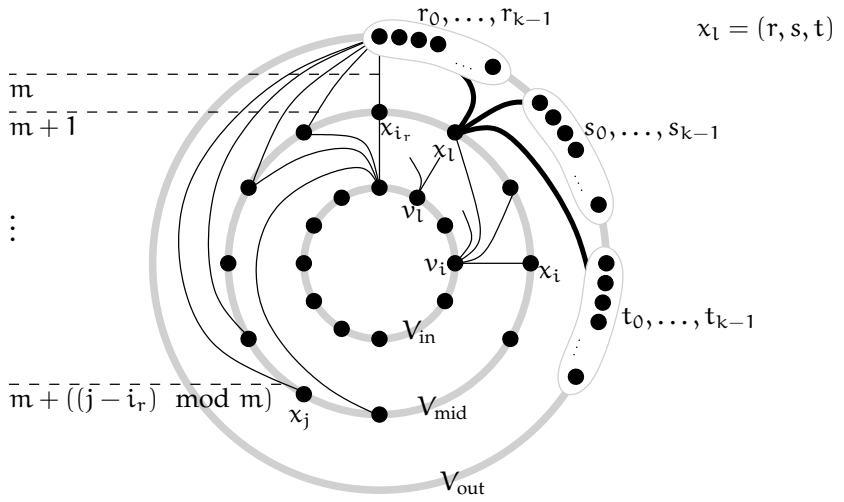

FIG. 7.3.: Illustration of an instance $I(m, k)$. Node x_l represents the triple (r, s, t) from the perfect matching. Node x_{i_r} corresponds to $r \in R$. Thick edges are of weight m.

$d(u, V_{mid}) = m$ for all essential nodes $u \in V$, Y_{mid} can only be preferred by essential users u' for which $d(u', X) > m$ holds. Since $M' = X$ is a perfect matching, $d(u_j, X) = m$ holds for all $u_j \in V_{out}$. It follows that $U(Y_{mid} \prec X)$ and V_{out} are disjoint. Furthermore, there are exactly p nodes $u \in Y_{in}$ such that $d(u, X) = m$. Hence $w(Y_{mid} \prec X) \leq m - p$. We conclude that indeed $w(Y \prec X) \leq w(Y_{io} \prec X) + w(Y_{mid} \prec X) \leq m - p + p = m$, hence our claim holds.

On the other hand, if X is a p-element node set that is not a perfect matching for I then $w_p(X) \geq m - p + k$. To see this, let $X_{mid} := X \cap V_{mid}$ and define an opposing candidate Y such that $Y \supseteq \{ x_{i'} \mid x_i \in X_{mid} \}$. The case where X and V_{mid} are disjoint is trivial. We verify that Y is preferred by all essential users u such that $d(u, X) > m$. First we count the nodes $u_j \in V_{out}$ such that $d(u_j, X) \leq m$. To this end, let $u \in R \cup S \cup T$. Consider the corresponding $x_{i_u} \in V_{mid}$ and assume that $d(u_j, X) \leq m$ for some corresponding $u_j \in V_{out}$. This can only be the case if either $u_j \in X$ or $x_{i_u} \in X$ or u is covered by X. Since X is no perfect matching there must be an u not covered by M and such that $x_{i_u} \notin X$. Hence there are at least $k - |X_{out}|$ corresponding nodes $u_j \in V_{out}$ for which $d(u_j, X) > m$ is satisfied. Here we set $X_{out} := X \cap V_{out}$. A similar argument shows that V_{in} contains at least $m - |X_{mi}|$ nodes v_j such that $d(v_j, X) > m$. Here, $X_{mi} := X \cap (V_{mid} \cup V_{in})$. We conclude $w(Y \prec X) \geq k - |X_{out}| + m - |X_{mi}| = m - p + k$.

To summarize, we have shown that if I has a perfect matching then $w_{p,p} \leq$

m and otherwise $w_{p,p} \geq m - p + k$. Our aim is to choose m, k such that $m \leq \gamma \cdot w(V)$ and $m - p + k > \gamma \cdot w(V)$ is satisfied. If we could achieve this, the 3DM instance I would have a perfect matching if and only if the constructed voting instance $I(m, k)$ admits an p-γ-Condorcet solution. The difficulty is that in the above inequalities the total number $w(V)$ of users is not constant but itself depends on m and k by the relation $w(V) = m + 3pk$. This, however, is not a serious problem: If we ensure that k is large enough in comparison to p, and m is large in comparison to $3pk$, those inequalities can be made true by means of elementary calculations. Note that we additionally need to require that m be larger than $|M| + 3q$ since we have to embed $M \cup R \cup S \cup T$ into V_{mid}. As this only adds an additional lower bound on m the solvability of the above linear inequality system is not affected. In fact, it is possible to calculate m and k in polynomial time so that they fulfill the inequalities and their values are polynomially bounded in the input size of I. This allows us to construct $I(m, k)$ with polynomial effort.

As in the proof of Corollary 7.3.4 we extend this result also to the absolute case with the help of Theorem 7.2.3. Note that it is essential here that the reduction from the discrete to the absolute case do not alter the total weight of the graph. □

7.4. Approximability

In this section we study the approximability of multiple competitive location problems, that is, (r, X_p)-medianoid and (r, p)-centroid, on general graphs. For the first problem we give an optimal approximation while for the latter we sharpen Hakimi's result and show that (r, p)-centroid is essentially not approximable at all (unless $P = NP$).

7.4.1. Follower Problem

The (r, X_p)-medianoid problem turns out to be strongly related to the MAXIMUM COVERAGE problem [Hoc97], for which a tight bound of approximability is known [KMN99]. The latter problem is defined as follows. Given a finite set M with weights $w_C \colon M \to \mathbb{Q}_0^+$, a collection $S \subseteq 2^M$ of subsets of M, and a positive integer $k \leq |S|$ find a subset $S' \subseteq S$ such that $|S'| = k$ and $w_C(\bigcup S')$ is maximal. Khuller et al. [KMN99] showed that MAXIMUM COVERAGE can be approximated within a factor of $e/(e-1)$ and that for any $\varepsilon > 0$, it is not possible to approximate the problem within a factor of $e/(e-1) - \varepsilon$ unless $P = NP$.

Theorem 7.4.1 *The absolute and the discrete* (r, X_p)*-medianoid problem are both equivalent to* MAXIMUM COVERAGE *under approximation preserving reduction and hence approximable within* $e/(e-1)$ *but not within* $e/(e-1) - \varepsilon$ *for any* $\varepsilon > 0$ *unless* P = NP.

Proof. Again we start with the discrete case. Consider an instance of MAXIMUM COVERAGE. We construct a graph $G = (V, E)$ in the following way: For each $m \in M$ and each $M' \in S$ we introduce nodes in G with weights $w(m) := w_C(m)$ and $w(M') := 0$, respectively. Furthermore we connect $m \in M$ and $M' \in S$ by an edge if and only if $m \in M'$ holds. The leader placement X_p consists of $p := k$ additional, zero-weighted nodes such that X_p and $M \cup S$ are disjoint. Now we connect each node v in X_p with each $M' \in S$ by an edge. Finally, we set the weights of all edges to 1.

First observe that $d(m, X_p) = 2$ for each node m corresponding to an element of M. Moreover, these nodes are exactly the ones with positive weight. Now consider any follower node y. If y corresponds to an element of M then y is the only node of positive weight in $U(y \prec X_p)$. Now let M' be some neighbor of y being simultaneously an element of S. Then y is still contained in $U(M' \prec X)$. Thus, if we replace y with M', the follower's gain can not decrease. Hence, we may assume that the follower placement Y_r contains only elements of S. Now consider an arbitrary node m of positive weight and $M' \in S$. It is clear by construction that $d(m, M') = 1$ if $m \in M'$ and otherwise $d(m, M') \geq 2$. Therefore, the elements of positive weight in $U(M' \prec X_p)$ are exactly the elements of M covered by M'. Thus the gain $w(Y_r \prec X_p)$ of the follower placement equals exactly the weight $w_C(\bigcup Y_r)$ of elements covered by Y_r where Y_r is considered a feasible solution of MAXIMUM COVERAGE. Hence the weight of an optimal solution S' for MAXIMUM COVERAGE equals the value $w_r(X_p)$ for $r = k$. We conclude that (r, X_p)-medianoid is as least as hard to approximate as MAXIMUM COVERAGE. In the absolute model there is obviously no benefit to place a server at an inner point of some edge. Therefore we do not need to consider this case separately.

Conversely, given an instance of (r, X_p)-medianoid we construct an equivalent instance of MAXIMUM COVERAGE as follows: First set $M := V$, $w_C := w$ and $k := r$. Then, let the collection S consist exactly of the sets $U(y \prec X)$ for all $y \in V$.

Again it follows easily that $w_r(X_p)$ and the optimal weight of the MAXIMUM COVERAGE instance are equal which completes the proof of the equivalence of the two problems.

The absolute case is tackled analogously by letting M be the set of candi-

dates (confer the discussion of discretizing (r, X_p)-medianoid in Section 7.2) instead of merely the set V of nodes.

The stated bounds now follow from the approximability properties of MAXIMUM COVERAGE [KMN99]. □

7.4.2. Leader Problem

We now study the approximability of the leader problem (r, p)-centroid. It will turn out that this problem is not satisfactorily approximable on general graphs. The following result is a strengthening of the lower bound of Hakimi [Hak90] who showed that (r, p)-centroid admits no constant-factor approximation algorithm.

Theorem 7.4.2 *The absolute and the discrete (r, p)-centroid problem cannot be approximated within $n^{1-\varepsilon}$ unless $P = NP$. In particular, this holds for any fixed $r \geq 1$.*

Proof. Again we start with the discrete case. Let $G_C = (V_C, E_C)$ be an undirected graph. A subset $V' \subseteq V_C$ is called a *vertex cover* if each $e \in E_C$ is incident to a node $v \in V'$. The NP-complete decision problem VERTEX COVER (confer [GJ79]) asks whether, given a graph G_C and a natural number $k \leq |V_C|$, the graph G_C has a vertex cover containing at most k nodes.

Given an instance of VERTEX COVER and an arbitrary positive integer s, we construct an instance $I(s)$ of (r, p)-centroid: The underlying undirected graph $G = (V, E)$ of $I(s)$ is obtained by replacing each edge $(u, v) \in E_C$ by an s-diamond $D(u, v)$ with terminal nodes u, v. The notion *s-diamond* is defined exactly as in the proof of Theorem 7.3.3; see also Figure 7.1. Note that the reduction ensures V_C to be a subset of terminal nodes of V. Finally we set $p := k$.

We are now going to show that $w_{r,p} \leq r$ if G_C has a vertex cover and otherwise $w_{r,p} \geq s$. This gap producing reduction will then enable us to derive the desired inapproximability bound by choosing s large enough.

First assume that G_C has a vertex cover V' containing at most $p = k$ nodes. Now consider $X_p := V'$ as a leader placement in $I(s)$. Since each non-terminal node in G is adjacent to a node in X_p, it follows easily that each node $y \in V$ can be preferred over X by at most one user. Hence $w(Y_r \prec X_p)$ can not exceed r for any r-element node set Y_r.

On the other hand, given a leader placement X_p we construct a set of nodes in V_C by replacing each non-terminal node in X_p by an adjacent terminal node. We denote this set by $\tau(X_p)$.

Let X_p be a leader placement such that $w_r(X_p) < s$. We claim that in this case $\tau(X_p)$ is a vertex cover of G_C. To this end assume that G_C has no vertex cover with at most k nodes. Then, for any placement X_p there is an edge (u, v) that is not covered by $\tau(X)$. It is clear that each follower placement Y_r containing u is preferred by all s non-terminal nodes of $D(u, v)$ and hence $w(Y \prec X) \geq s$.

Note that n depends linearly on s. Thus we can determine a suitable s in polynomial time such that the equation $rn^{1-\varepsilon} < s$ is satisfied. Furthermore s is itself polynomially bounded by the input size of the VERTEX COVER instance, which ensures that $I(s)$ can be constructed in polynomial time. Let A be an algorithm approximating (r, p)-centroid within $n^{1-\varepsilon}$, and let X_A be the result obtained by applying A to $I(s)$. Further assume that G_C has a vertex cover with at most $p = k$ nodes. Then $\tau(X_A)$ is a vertex cover of G_C. For, the performance guarantee of A yields $w_r(X_A) \leq rn^{1-\varepsilon} < s$.

Hence a polynomial running time for A would imply VERTEX COVER $\in$ P as follows: First compute s and $I(s)$. Then apply A to $I(s)$. Finally, check whether $\tau(X_A)$ is a vertex cover for G_C.

As in the proof of Corollary 7.3.4 we extend this result also to the absolute case by means of Theorem 7.2.3. □

7.5. Conclusion and Further Remarks

In this chapter we have investigated the complexity and the approximability of multiple competitive location problems on general graphs. We have shown that (r, p)-centroid is Σ_2^p-complete which suggests that it is substantially more complex than typical NP-hard optimization problems whose objective function is efficiently computable. In terms of approximability, we have shown that for any $\varepsilon > 0$ there is no $n^{1-\varepsilon}$-approximation algorithm for (r, p)-centroid unless P = NP. Roughly speaking, there is no approximation algorithm for this problem with a reasonable worst case behavior.

The complexity status of (r, X_p)-medianoid has already been settled by Hakimi [Hak83]. He has shown that its decision problem is NP-complete. To the best of our knowledge its approximability has not been examined yet. We have established a simple equivalence between (r, X_p)-medianoid and the well-studied MAXIMUM COVERAGE problem. This yields a tight bound $e/(e-1)$ for the approximability of (r, X_p)-medianoid.

For general graphs, we have resolved the complexity and the approximability of the problems satisfactorily. Hence, we will concentrate in subsequent chapters on special types of instances. As in the single location case

we will examine the problems on trees. In the case of the sophisticated (r, p)-centroid problem, we shall see that it is even worth investigating the simple case of a path.

8. The Follower Problem on Trees

This chapter is devoted to analyzing the complexity of the follower problem, namely (r, X_p)-medianoid, on trees. As in the single server case, the problem turns out to be significantly easier on trees than on general graphs. We have seen in the preceding chapter that (r, X_p)-medianoid is NP-hard on general graphs and have even stated a lower bound of approximability. In contrast, the problem becomes efficiently solvable on trees. Specifically, we shall see in Section 8.1 that it is a special case of the so called *maximum coverage location* problem, which can be solved with an algorithm of Tamir [Tam96], in $O(rn^2)$ time.

In the remainder of this chapter we concentrate on the case where the follower places only *one* server (aka *single* maximum coverage location) in the presence of a multipoint leader placement. We start with the observation that in this specific case Tamir's algorithm leads in this specific case to a quadratic running time which, asymptotically, is not faster than the naive approach of performing a depth first search traversal from every node.

Our first non-trivial approach to attack single maximum coverage location is based on a so called *recursive coarsening* technique which we present in Section 8.2.1. The resulting algorithm has a subquadratic running time of $O(n \log^2 n / \log\log n)$.

In Section 8.2.2 we describe a general model of Kim et al. [KLTW96] for locating a *tree-shaped facility* on a tree. In this context we consider the *indirect covering subtree* problem which has single maximum coverage location as a special case. In Section 8.2.3 we then briefly reproduce an algorithm of Kim et al. with running time $O(n \log^2 n)$ for indirect covering subtree which is, albeit subquadratic, slightly slower than our tailor-made recursive coarsening algorithm for single maximum coverage location.

Finally, in Section 8.2.4 we present an improvement of the algorithm of Kim et al. which leads to an surprisingly simple algorithm for indirect covering subtree (and thus also single maximum coverage location) with running time $O(n \log n)$. To the best of our knowledge this is the fastest known algorithm for this class of problems.

8.1. Tamir's Algorithm

The (r, X_p)-medianoid problem on trees has first been studied by Megiddo et al. [MZH83]. In that work the authors investigate the more general *maximum coverage location problem* which is defined as follows:

Definition 8.1.1 (Maximum Coverage Location on a Tree [MZH83])
The input instance consists of a tree $T = (V, E)$ with edge lengths $c\colon E \to \mathbb{Q}_0^+$ inducing a distance function $d\colon T \times T \to \mathbb{Q}_0^+$. Additionally, there are functions $\rho, w\colon V \to \mathbb{Q}_0^+$ specifying the *sensitivity radius* $\rho(v)$ and the *demand value* $w(v)$ for each node $v \in V$. Moreover, we are given a positive integer $1 \leq k \leq n$.

When a subset $F \subseteq G$ of facilities (points) has been chosen, a node $v \in V$ is said to be *covered by* F if and only if $d(v, F) \leq \rho(v)$. The gain $W(F)$ of F is the sum $\sum\{w(v) \mid d(v, F) \leq \rho(v)\}$ of the demand of the nodes covered by F. The goal is to find a point set $F \subseteq G$ of size $|F| = k$ whose gain $W(F)$ is maximum.

It is easy to see that an instance of the (r, X_p)-medianoid problem can be formulated as a maximum coverage location problem: Leave the input tree unchanged, and define the sensitivity radius of each node to be $\rho(v) := d(v, X_p) - \varepsilon$ where ε is a suitably small constant. The number of facilities is $k := r$. Then the total demand $W(Y_r)$ of nodes covered by any r-element point set Y_r is exactly $w(Y_r \prec X_p)$.

Using the concept of boundary points (confer Section 7.2) Megiddo et al. [MZH83] reduce the (absolute) maximum coverage location problem to its discrete counterpart where only node sets $F \subseteq V$ are taken into account. In particular, they show that for any instance of maximum coverage location it is possible to identify efficiently a candidate set $C \subseteq G$ that contains at least one optimal solution and has size $O(n)$. For this reason, we will restrict ourselves to the *discrete* maximum coverage location problem in what follows.

The central result of Megiddo et al. [MZH83] is a polynomial-time algorithm for discrete maximum coverage location with running time of $O(k^2n^2)$. In their original work, Megiddo et al. claimed even a running time $O(kn^2)$. However, Broin and Lowe [BL86] pointed out an mistake in the argumentation of Megiddo et al. and proved the slightly lower performance stated above.

Later, Tamir [Tam96] actually improved this running time to $O(kn^2)$. His result applies to an even more general model than maximum coverage location. In this model, each node v is associated with a non-negative *setup cost*

$c(v)$ and a non-decreasing function f_v. The problem consists in determining a k-element node set F minimizing the *total cost*

$$c(F) + \sum_{u \in V} f_u(d(u, F)) .$$

By setting $c \equiv 0$ and

$$f_u(x) = \begin{cases} -w(u) & \text{if } x \leq r(u) \\ 0 & \text{otherwise,} \end{cases}$$

the cost of any k-element node set F in Tamir's model equals the negative gain $-W(F)$ of F in maximum coverage location. We obtain

Theorem 8.1.2 *The absolute and the discrete (r, X_p)-medianoid problem on an n-node tree can be solved in $O(rn^2)$ time.* □

8.2. Single Follower on Trees

In this section we examine $(1, X_p)$-medianoid on trees, that is, the follower places one single server in the presence of a multipoint leader placement. Note that for $p = 1$, this amounts to the (single) x-medianoid problem investigated in the first part of this thesis. Indeed, in Theorem 4.2.6 we have observed that this simple case can be solved in linear time.

For $p \geq 2$ a *quadratic* running time can be obtained with the following trivial approach. For any node v we determine the distances $d(u, v)$ to each node $u \in V$ by means of one depth-first search. In doing so, we sum up the weights of all nodes u where $d(u, v) \leq \rho(u)$. Hence we can compute the gain $W(v)$ of v in linear time. Performing this for all nodes $v \in V$ we can compute all optimal nodes in quadratic time.

Can we improve this trivial upper bound? Unfortunately, Tamir's algorithm [Tam96] does not help here, since it yields quadratic running time for $r = 1$, too. An algorithm faster than the naive approach has first been suggested by Kim et al. [KLTW96]. The authors provide an algorithm for solving the so called *indirect covering subtree* problem on a tree in $O(n \log^2 n)$ which contains maximum coverage location as a special case.

In the following subsection we describe a new, slightly faster algorithm for single maximum coverage location. We then reproduce the algorithm of Kim et al. briefly in Section 8.2.3. Finally, we improve their algorithm and achieve a running time of $O(n \log n)$, see Section 8.2.4. This is the best bound currently known.

8.2.1. A Recursive Coarsening Algorithm for Single Maximum Coverage Location

In this subsection we present an algorithm with running time $O(n \log^2 n / \log\log n)$ for solving the maximum coverage location problem. The approach is based on the so-called *recursive coarsening* technique. The algorithm is slightly faster than the previously best algorithm of Kim et al. [KLTW96] which has a running time of $O(n \log^2 n)$ and will be described later in Section 8.2.3.

In the process of my research, the idea of coarsening has been an important intermediate step in devising the two-terminal subtree technique, with which we have already become acquainted in Chapters 4 and 5, in order to determine Φ-solutions and the set of φ_0-bounded solutions. Indeed, we will use this technique once again in Section 8.2.4 for developing an $O(n \log n)$ algorithm for single maximum coverage location, which is faster than the coarsening approach described in this section and also than the algorithm of Kim et al. The reader might argue that this stronger result makes the recursive coarsening technique obsolete. However, I feel that its description might nevertheless be valuable. First of all, it documents the progress being made during our research. Moreover, it underlines the effectiveness of the 2TS technique. Specifically, we will learn that a small modification of the approach causes a significant speedup and simplification. And finally, it may well be that the coarsening idea turns out to be fruitful for other optimization problems on trees.

In order to avoid confusions, we emphasize that the order in which the results are presented in this thesis is guided by thematic and systematical viewpoints and does not match the chronological order in which they have been developed. As mentioned above, the idea of coarsening is in fact a predecessor of the two-terminal subtree technique and has been devised in the context of single maximum coverage location. Thus this chapter is the best place to go also into the details of this preliminary approach and to explain also some motivations and ideas behind the 2TS technique.

The naive approach of solving the single maximum coverage location problem is to determine for each possible facility position $u \in V$ the weight

$$W(u) := \sum \{ w(v) \mid d(u,v) \leq \rho(v) \}$$

of nodes covered by facility u, and then choosing a facility with maximum value as the solution. As observed above, the number $W(u)$ can be determined in linear time by a depth-first search traversal of the tree. This yields

an algorithm with running time $O(n^2)$.

It is not hard to see that this approach performs some redundant computations. The property whether a node is covered by a facility or not depends only on their distance and not on the exact position in the tree. Thus it is possible to gather the distance information for one facility and reuse part of that information for subsequent traversals. To be (a little) more precise, we store sorted lists at certain nodes which contain the distance information of all nodes in the subtree hanging from this node. Thus we can avoid repeated traversals of the same subtree since whenever a depth first search traversal reaches such a list it can replace traversing the subtree by reading the stored information.

The drawback of this approach is that maintaining these lists does not come for free. In particular it is not possible to store the lists at each of the nodes, as this would take more time than the naive approach. Thus we introduce the notion of a *coarsened tree*. This tree is constructed by identifying a suitable subset of the nodes, called the *guard nodes*, then splitting the tree at the guard nodes, and contracting each of the resulting trees into a single edge. The distance lists described above are then stored only at guard nodes. Then we solve the problem recursively locally on each of the contracted subtrees and globally with the help of the lists stored at the guard nodes.

Obviously the number of guard nodes and the size of the resulting subtrees are inversely proportional. Our analysis carefully balances those two parameters so that we can provide an algorithm which is substantially faster than the naive approach described above.

Coarsening the Tree

For the definition of a k-coarsening we make use of the notions terminal, two-terminal subtree (2TS) and valid split as they have been introduced in Section 4.3. (Originally, the definition of a k-coarsening was not based on 2TSs. Here, we are using a reformulation for the sake of a uniform terminology.)

Definition 8.2.1 (k-coarsening) Let $T = (V, E)$ be a tree, $k \in \mathbb{N}$, $n \geq 2k$. A tree $\tilde{T} = (\tilde{V}, \tilde{E})$ is a k-*coarsening* of T if

1. $\tilde{V} \subseteq V$ and $|\tilde{V}| \leq 2\Delta k$ where Δ is the maximum node degree in T. We call the nodes in $\tilde{V}$ *guard nodes*.
2. The set $\tilde{V}$ of guard nodes forms a valid split. The induced 2TSs are called *super edges* and constitute the edge set $\tilde{E}$.

3. The end nodes of a super edge are its terminals.
4. The size $|\tilde{e}|$ of each super edge $\tilde{e}$ in E is at most n/k.

A coarsening of a tree may be regarded as a generalization of the subdivision of T into 2TSs by splitting at the median (confer Lemma 4.3.3. There, we subdivided a tree into $O(\Delta)$ subtrees of which each had size at most $n/2$. By comparison, a k-coarsening splits into $O(k\Delta)$ subtrees of size at most $fracnk$. We use the more vivid notion of a super edge instead of 2TS since we will deal with $\tilde{T}$ as a tree in its own right and perform basic operations such as depth first search traversals on it.

Lemma 8.2.2 *A k-coarsening of a tree always exists and can be computed in linear time.*

Proof. Let $T = (V, E)$ be given. Choose an arbitrary root and perform a depth first search traversal. We divide the tree into super edges starting near the leaves. To this end, maintain at each node u the current size $s(u)$ of the super edge u belongs to, and a number $f(u)$ which counts the guard nodes in that super edge. If u is a leaf, then $s(u) = 1$ and $f(u) = 0$.

Otherwise, let u be an inner node with sons $v_1, \ldots, v_r$. Assume as induction hypothesis that $s(v_i) < n/k$ and $f(v_i) \leq 1$ for $i = 1, \ldots, r$. Let $s := 1 + \sum_i s(v_i)$ and $f := \sum_i f(v_i)$. If both $s < n/k$ and $f \leq 1$, then set $s(u) := s$, $f(u) := f$ and continue the traversal. Otherwise, mark u as a guard node and set $s(u) = 1$ and $f(u) = 1$. If u is marked as a guard node but $f = 0$, this defines a set of super edges near the periphery of the tree without a second endpoint; in this case select for each son of u an arbitrary leaf in that subtree below the son and mark it also as a guard node.

Properties 2, 3, and 4 are clear by construction. It remains to show that the number of guard nodes constructed by the algorithm is not too large as stated by property 1. This follows from a potential function argument. We call a guard node to be *active* if it is adjacent to a regular node not yet assigned to a completed super edge. The potential function Φ counts the number of active guard nodes. Obviously $\Phi \geq 0$ and initially $\Phi = 0$.

We investigate two cases where a new guard node is created: When it is created due to exceeding the size bound n/k, this increases Φ by at most 1. Moreover, this increase can happen no more than k times during the whole algorithm since none of the n nodes is counted more than once. On the other hand, when a new guard node is created due to the guard node count overflowing, we replace at least two active guard nodes by the new active node. Thus the value of Φ decreases by at least 1. From these observations it follows that the total number of guard nodes that are active at some time

during the algorithm is at most 2k. The remaining guard nodes are the ones which are inserted at the periphery of the tree as described above. Each of the already counted 2k guard nodes can carry at most $\Delta - 1$ of them; hence the total number of guard nodes is at most $2\Delta k$. □

In the following we assume that the input tree T is 3-regular and hence $\Delta = 3$. This is accomplished by the same construction we used in Section 5.3.1 to compute all φ_0-bounded solutions on a degree-3-bounded tree. Recall that this operation increases the size of the node set size by a factor of less than 2. This will enable us to construct super trees of size $O(k)$ each of whose super edges contain at most n/k nodes.

Super Edge Lists

For each super edge $\tilde{e} = (u, v)$, we define the *super edge list* $L_u(v)$ to be the set

$$L_u(v) := \{ (v', \rho') \mid v' \in T_u(v) \text{ and } \rho' = \rho(v') - d(v, v') \}.$$

The set $L_u(v)$ is organized as a sorted list where the numbers ρ' play the role of the keys. This list is stored at the guard node v. The purpose of list $L_u(v)$ is to maintain the coverage information of all nodes in the subtree $T_u(v)$ when a facility is placed outside this subtree: A node $v' \in T_u(v)$ is covered by facility $f \in V - T_u(v)$ if and only if $d(f, v) \leq \rho'$ where ρ' is the key of v' stored in the list.

We first describe the algorithm ComputeSuperEdgeList depicted in Figure 8.2. This algorithm uses a depth first search traversal of the coarsened tree to compute the list $L_u(v)$ for a given super edge (u, v). Let z_1 and z_2 denote the sons of v in $\tilde{T}$. Assume that $L_v(z_1)$ and $L_v(z_2)$ have already been computed. This scenario is depicted in Figure 8.3.

At this point we are able to compute $L_u(v)$. Observe that each node v' in $L_u(v)$ is either an element of one of the super edges $(v, z_1), (v, z_2)$ or it appears already in one of the lists $L_v(z_1), L_v(z_2)$. In the latter case, the key for v' in the list $L_u(v)$ is given by its key in $L_v(z_i)$ minus the distance $d(v, z_i)$. This is implemented efficiently as follows. First create copies $L'_v(z_1), L'_v(z_2)$ of lists $L_v(z_1), L_v(z_2)$ and keep the original lists at their super edges for later reference. This takes overall time $O(|V|)$ for both lists. Second, in each copied list $L_v(z_i)$ decrease all keys uniformly by the value $d(v, z_i)$. Notice that this does not affect the order in the lists and thus takes overall time $O(|V|)$ for both lists. Third, we obtain $L_u(v)$ by merging the lists $L_v(z_i)$ and two special lists L_1, L_2 of the nodes in $(v, z_1), (v, z_2)$, namely $L_i := \{ (v', \rho') \mid$

```
1  input: Tree T = (V, E)
2  W(v) ← 0 for each v ∈ V
3  construct coarsened tree T̃ = (Ṽ, Ẽ)
4  for each super edge ẽ = (u, v) ∈ Ẽ
5      if |ẽ| > 2
6          ComputeWeights(T[ẽ − {u, v}])
                   on tree induced by ẽ − {u, v}
7      compute L_u(v) ← ComputeSuperEdgeList(u, v)
8      for each v′ ∈ ẽ
9          set W(v′) ← W(v′)+
                   ∑{ w(z′) | (z′, ρ′) ∈ L_u(v) and ρ′ ≥ d(v, v′) }
10     handle list L_v(u) analogously
11 output: the weight W(v) for each v ∈ V
```

FIG. 8.1.: Algorithm COMPUTEWEIGHTS.

$v' \in (v, z_i)$ and $\rho' = \rho(v') - d(v, v')\}$ for $i = 1, 2$. The computation of all distances $d(v, v')$ and the *sorted* lists L_i needs time $O(|V|/k \cdot \log(|V|/k))$ per super edge. The time for the merge operations is clearly in $O(|V|)$.

The Covered Demand

Let v' be a node in super edge $\tilde{e} = (u, v)$. Recall that v' covers all nodes z' with $\rho(z') \geq d(v', z')$. If $z' \notin \tilde{e}$, then it appears w.l.o.g. as a pair (z', ρ') in the list $L_u(v)$, and the above condition is equivalent to $d(v', v) \leq \rho'$. Hence the total weight of all nodes in $T - \tilde{e}$ covered by v can be computed by traversing both super edge lists $L_u(v)$ and $L_v(u)$ and summing up the weights of all nodes satisfying this condition. The weight of the nodes covered by z' in the super edge $\tilde{e}$ itself is computed by applying the algorithm recursively to the tree $T[\tilde{e} - \{u, v\}]$. At this point, the guard nodes u, v are removed for technical reasons to avoid double counting their weight.

Our implementation is as follows. Let $\tilde{e} = (u, v)$ be a super node and let $v_1, \ldots, v_t$ be its nodes. We assume in particular $v_t = v$. Sort the nodes such that $d(v_1, v) \geq \ldots \geq d(v_{t-1}, v)$. Let $L := L_u(v)$ be the super edge list of v w.r.t. u sorted with descending key values. We maintain a current node v_i and a corresponding pointer p pointing to the first element in L with radius strictly smaller than $d(v_i, v_t)$, together with a variable W summing the weight of all nodes before p. It is easy to observe that when we move through the list $v_1, \ldots, v_t$, then the pointer p advances linearly through L and the weight W

```
1  input: super edge (u, v)
2  if L_u(v) has already been computed, then return L_u(v)
3  initialize L_u(v) ← ∅
4  for each super edge ẽ_i = (v, z_i) incident with (u, v)
5      L'_v(z_i) ← ComputeSuperEdgeList(v, z_i)
6      decrease all keys in L'_v(z_i) by d(v, z_i)
7      merge sorted lists L_u(v) ← L_u(v) ∪ L'_v(z_i)
8      compute special list
           L_i ← { (v', ρ') | v' ∈ ẽ_i, ρ' := ρ(v') − d(v, v') }
9      merge sorted lists L_u(v) ← L_u(v) ∪ L_i
10 output: List L_u(v)
```

FIG. 8.2.: Algorithm COMPUTESUPEREDGELIST.

can be maintained by adding the weights of the visited nodes. Thus the total time for handling a set $L_u(v)$ is bounded by $O(n/k \cdot \log(n/k) + n)$.

The Total Running Time

We now estimate the total running time of algorithm COMPUTEWEIGHTS (see Figure 8.1) within one single level of recursion. To this end, consider first the call to COMPUTESUPEREDGELIST in line 7. As can been seen in line 2 of COMPUTESUPEREDGELIST, this subroutine is called at most twice per super edge, namely once in each direction. This leads to an overall running time of $O(k\,(n/k \cdot \log(n/k) + n)) \subseteq O(n \log n + kn)$ for computing all super edge lists.

As pointed out before the handling of a list $L_u(v)$ in Line 8 of algorithm COMPUTEWEIGHTS can be implemented with running time $O(n + n/k \cdot \log(n/k))$. As this happens at most twice per super edge we obtain the same overall time $O(n \log n + kn)$ for handling all super edge lists.

Finally the algorithm solves the problem recursively for each super edge $\tilde{e}_i$ or more precisely for each induced subtree $T_i := T[\tilde{e}_i - \{u_i, v_i\}]$ where u_i, v_i are the endpoints of $\tilde{e}_i$. Let $n_i := |T_i|$ denote the size of this subtree. Clearly the sum $\sum_i n_i$ of these sizes is at most n since different super edges are disjoint except for the endpoints which are removed from the subtrees T_i. Moreover $n_i \leq n/k$ as this condition holds already for the guard edges $\tilde{e}_i$.

This allows us to estimate the total running time by the following recur-

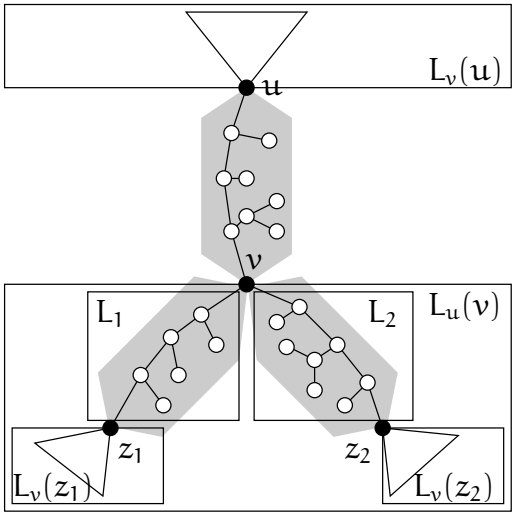

FIG. 8.3.: Computation of the Super Edge Lists. Guard nodes u, v, z_i are painted black, super edges $\tilde{e} = (u, v)$ and $\tilde{e_i} = (v, z_i)$ are painted gray. Rectangular areas mark the nodes contained in the respective lists.

rence equation, for some constant c:

$$h(n,k) \leq c \cdot (n \log n + k \cdot n) + \sum_i h(n_i, k) \tag{8.1}$$

We prove by induction that (8.1) is solved by the function

$$h(n,k) \leq c \cdot n \cdot (\log n + k) \cdot \frac{\log n}{\log k}. \tag{8.2}$$

The induction basis is clear. As induction hypothesis we assume that (8.2) holds for $h(n_i, k)$. Plugging this into (8.1) yields

$$\begin{aligned} h(n,k) &\leq c \cdot n \cdot (\log n + k) + \sum_i c \cdot n_i \cdot (\log n_i + k) \cdot \frac{\log n_i}{\log k} \\ &\leq c \cdot n \cdot (\log n + k) + \sum_i c \cdot n_i \cdot (\log n + k) \cdot \frac{\log(n/k)}{\log k} \\ &\leq c \cdot n \cdot (\log n + k) + c \cdot n \cdot (\log n + k)\left(\frac{\log n}{\log k} - 1\right) \\ &= c \cdot n(\log n + k)\frac{\log n}{\log k} \end{aligned}$$

as claimed in (8.2). Let N be the size of the original input instance and choose $k := \log N$. This yields the following main result.

Theorem 8.2.3 *The single maximum coverage location problem on trees can be solved in time* $O(n\,(\log n)^2/\log\log n)$. □

8.2.2. Tree-Shaped Facilities

In this section we examine the *discrete subtree location model* of Kim et al. [KLTW96], which is a generalization of the single maximum coverage location. Based on this model we will develop in the next section an efficient algorithm which solves single maximum coverage location as a byproduct.

The discrete subtree location model is as follows: We are given a tree T along with edge weights $c\colon E \to \mathbb{Q}^+$ inducing a distance function $d\colon T \times T \to \mathbb{Q}_0^+$. Besides, a mapping $t\colon V \times V \to \mathbb{Q}$ specifies the *transportation cost* $t(u,v)$ between any node pair $u, v \in V$. Here, $t(u,v)$ may be interpreted as the cost imposed on the provider when customer u utilizes the service provided by a facility placed at v (for example when transporting a good). The transportation cost $t(u,v)$ is zero if $u = v$. We remark that in most practical applications $t(u,\cdot)$ is *increasing* in the distances $d(u,\cdot)$, that is, $d(u,v) \geq d(u,v')$ implies $t(u,v) \geq t(u,v')$. The following considerations and results, however, are also valid in the more general scenario in which $t(u,\cdot)$ is allowed to be uncorrelated to the actual distances in the tree.

If Y is a subtree of T then $t(u,Y)$ denotes the transportation cost $t(u,y)$ between u and the unique node $y \in V(Y)$ closest to u according to the *distance* $d(u,\cdot)$, that is, y is the only node $y \in Y$ such that the path $P(u,y)$ contains no edge from Y. Note that, since the transportation cost may be independent from the distances in the tree, y does *not* need to be the node minimizing the transportation cost $t(u,\cdot)$. Such a scenario may occur in reality when the users decide which facility they use but the provider has to bear cost.

If $U \subseteq V$ is a node set, we define $t(U,Y) := \sum_{u\in U} t(u,Y)$. In other words $t(U,Y)$ is the total transportation cost for the set U of customers induced by facility Y. We write $t(Y) := t(V,Y)$. Now, the goal of the discrete subtree location problem is to identify a subtree Y of T such that $c(Y) + t(Y)$ is minimum. Here, $c(Y)$ represents the *setup cost* of the tree-shaped facility Y and $t(Y)$ is the *total transportation cost* induced by Y.

The subtree location model is quite general and contains several well known location problems as a special case.

The *direct covering subtree* problem has been investigated by Church and Current [CC93]. In their model the transportation cost $t(u,v)$ is zero if $u = v$ and $w(u)$ otherwise. Here, $w(u)$ is some weight depending only on the user node u. In other words, the transportation cost equals the total weight of

nodes *not* contained in the subtree Y. Another problem, called *median subtree location* problem uses transportation cost proportional to the distance. Specifically, we set $t(u,v) = w(u) \cdot d(u,v)$. This problem can be considered as a generalization of the well known 1-median problem.

The *indirect covering subtree* problem is a generalization of the direct subtree location model. Similar to the maximum coverage location problem we assign to each node u a sensitivity radius $\rho(u)$. Then, the transportation cost $t(u,v)$ is zero if $d(u,v) \leq \rho(u)$ and $w(u)$ otherwise. The total transportation cost incurred by some subtree Y thus equals the sum of weights $w(u)$ of nodes u where $d(u,Y) > \rho(u)$. Clearly, this is equivalent to *maximize* the sum of weights of nodes u where $d(u,Y) \leq \rho(u)$. Therefore, if we were able to enforce Y being a node rather than a subtree we would arrive at the single maximum coverage location problem. In fact, this can easily be achieved by scaling all edge weights and radii by a suitably large constant which makes non-trivial subtrees too expensive in terms of their setup cost. Thus, single maximum coverage location is a special case of the indirect covering subtree problem.

8.2.3. The Algorithm of Kim et al.

In the sequel we will briefly describe the approach of Kim et al. [KLTW96] for solving the discrete subtree location problem.

To this end we assume that the input tree T is rooted at some distinguished node s. For technical reasons we shall adopt the convention that s is the father of itself. The authors reduce the solution of the problem to the computation of the values $t(v)$, $t(T_v, v)$ and $t(T_v, f(v))$ for each node v of T. Here, $t(v)$ is the total transportation cost induced by service node v, $t(T_v, v)$ is the transportation cost for the users in the subtree T_v hanging from v when a service node is located at v, and $t(T_v, f(v))$ is the transportation cost within T_v induced by the father $f(v)$ of v. The authors show that one can determine in linear time an optimum to the subtree location problem once these values have been precomputed for all nodes v. This can be seen as follows.

Let v be a node of T. Assume that we have computed the values $t(v)$, $t(T_v, v)$ and $t(T_v, f(v))$. Then let us define

$$C(v) := \min\{\, c(Y) + t(T_v, Y) \mid Y \text{ is a subtree of } T_v \text{ containing } v \,\},$$

and

$$C^+(v) := \min\{\, c(Y) + t(T_v, Y) \mid Y \text{ is a subtree of } T_v^+ \text{ containing } f(v) \,\},$$

where T_v^+ equals the union of T_v with the edge $(v, f(v))$.

It is not hard to see that the optimum can now be expressed by

$$\min_{v \in V}(C(v) + t(v) - t(T_v, v)\,).$$

Moreover, Kim et al. demonstrate that the $C(\cdot)$- and $C^+(\cdot)$-values can be computed in linear time by means of a simple bottom-up dynamic programming approach. To this end assume that v is a leaf of T. Then

$$C(v) = 0 \quad \text{and} \quad C^+(v) = \min\{t(v, f(v)), c(v, f(v))\}\,.$$

Otherwise, we have

$$C(v) = \sum_{u \text{ is son of } v} C^+(u)\,,$$

and

$$C^+(v) = \min\{C(v) + c(v, f(v)), t(T_v, f(v))\}\,.$$

From these equations it follows that indeed an optimal solution can be determined in linear time in a bottom-up fashion once the values $t(v)$, $t(T_v, v)$ and $t(T_v, f(v))$ have been computed for each $v \in V$. In general, we need time $\Omega(n^2)$ to calculate these t-values. However, Kim et al. provide faster algorithms for several interesting special cases. Specifically, they develop linear time algorithms for solving the above introduced direct covering subtree and the median subtree location problem. They also suggest a subquadratic algorithm for the indirect covering subtree problem which contains our single maximum coverage problem as a special case. In the sequel we give an outline of their algorithm.

Indirect Covering Subtree in Subquadratic Time

The algorithm of Kim et al. for solving the indirect covering subtree problem is based on the so called *bitree model*. In this model each (undirected) edge (u, v) of the input tree is replaced by two anti-parallel, directed arcs (u, v), (v, u). We call the resulting tree T' *bitree* of T. With each arc (u, v) of the bitree we associate a cost $c_{T'}(u, v)$ representing the length of this arc. But in contrast to the edge lengths of the input tree T, we allow these lengths to be negative. This induces a distance function $d_{T'}\colon V \times V \to \mathbb{Q}$ where $d_{T'}(u, v)$ is the length of the unique u–v path in T'. Now we define our transportation cost $t'(u, v)$ between two arbitrary nodes u, v to be zero if $d_{T'}(u, v) \leq \rho(u)$ and $w(u)$ otherwise. We set $t'(v) = \sum_{u \in V} t'(u, v)$.

The algorithm of Kim et al. is based on a subroutine for efficiently computing $t'(\cdot)$ on a given bitree T'. By means of this subroutine it is then quite easy to retrieve the values $t(v)$, $t(T_v, v)$ and $t(T_v, f(v))$ for each node v of the input tree T. It follows from the above discussion that the knowledge of these values enables us to identify an optimal tree-shaped facility.

The algorithm uses a divide-and-conquer technique to calculate $t'(\cdot)$. Specifically, it partitions the node set into two sets V_1, V_2 with the following properties

(i) $V_1 \cup V_2 = V$,

(ii) $V_1 \cap V_2 = \{z\}$ for some node z,

(iii) $|V_i| \leq \frac{2}{3}n + \frac{1}{3}$ for $i = 1, 2$, and

(iv) V_i induces a subtree T_i of T for $i = 1, 2$.

Such a decomposition along with the *split node* z can be determined in $O(n)$ [KA75].

Let v be an arbitrary node. Then

$$t'(v) = t'(V_1, v) + t'(V_2 - z, v)\,. \tag{8.3}$$

First, we will show that the second term $t'(V_2 - z, v)$ can be computed in $O(n \log n)$ time for all nodes $v \in V_1$. To this end consider the list L_1 containing all nodes v in V_1 sorted in increasing order with respect to their distance $d(z, v)$ from the split node z. The list L_2 contains all nodes u of $V_2 - z$ sorted in increasing order with respect to the value $\rho(u) - d(u, z)$. It is clear that both lists can be constructed in $O(n \log n)$. Now we merge both lists into one sorted list L. (Ties are broken in favor of nodes from V_1.) We observe that the term $t'(V_2 - z, v)$ equals the sum of all weights $w(u)$ of nodes $u \in V_2 - z$ such that $\rho(u) - d(u, z) < d(z, v)$. Note that the respective hands of this inequality are exactly the sorting keys of the lists L_2 and L_1, respectively. Thus, the values $t'(V_2 - z, v)$ can be computed all at once by a single traversal through the combined list L where we keep track of the sum of weights of all nodes from $V_2 - z$ encountered so far. Symmetrically, we can now compute $t'(V_1, v)$ for all nodes $v \in V_2 - z$.

Finally, we apply this procedure recursively to the sub-bitrees induced by V_1 and $V_2 - z$, thereby computing the values $t'(V_1, v)$ for all $v \in V_1$ and $t'(V_2 - z, v)$ for all $v \in V_2 - z$, respectively. The values $t'(\cdot)$ can now be obtained by Equation (8.3).

The running time $h(n)$ of the resulting algorithm can be determined by using the recurrence equation

$$h(n) = O(n \log n) + h(|V_1|) + h(|V_2|)\,,$$

where $|V_i| \leq \frac{2}{3}n + \frac{1}{3}$ and $|V_1| + |V_2| = n + 1$. Using standard inductive arguments [AHU74] one can easily show that h is $O(n \log^2 n)$.

To sum up, we are able to compute the values $t'(\cdot)$ in time $O(n \log^2 n)$ on the bitree T' for the input tree T. It remains to explain how we can use this information to determine $t(v)$, $t(T_v, v)$ and $t(T_v, f(v))$ for all nodes v of the input tree T which, in turn, is sufficient to build an optimal tree-shaped facility.

First we describe how we can determine $t(\cdot)$. For this purpose we simply set $c_{T'}(u, v) := c_{T'}(v, u) := c_T(u, v)$ for all edges (u, v) of the input tree T. It is then immediately clear that $t(v) = t'(v)$ for all $v \in V$.

In order to compute $t(T_v, v)$ for all $v \in V$ we set $c_{T'}(u, f(u)) := c_T(u, f(u))$ and $c_{T'}(f(u), u) := -\infty$ for all $u \neq s$. This constructions ensures that the transportation cost $t'(u, v)$ is always zero if u is *not* a descendant of v. Thus $t(T_v, v) = t'(v)$ holds for this construction.

Finally, we wish to determine $t(T_v, f(v))$. To this end we introduce on each edge $(v, f(v))$ of T a new node $f'(v)$ such that edge $(v, f'(v))$ has length $c_T(v, f(v))$ and edge $(f'(v), f(v))$ has length zero. This increases the number of nodes to $2n - 1$. We set $w(f'(v))$ and $\rho(f'(v))$ to zero. It is easy to see that $t(T_v, f(v))$ in the original tree equals $t(T_u, u)$ in the newly constructed undirected tree where $u := f'(v)$. At this point, $t(T_u, u)$ can be computed for all nodes u in total time $O(n \log^2 n)$ as described above. This completes the description of the algorithm of Kim et al.

Theorem 8.2.4 ([KLTW96]) *The indirect covering subtree problem on an n-node tree can be solved in $O(n \log^2 n)$ time.* □

8.2.4. An $O(n \log n)$-Time Algorithm

In this section we describe an algorithm for the indirect covering subtree problem with running time $O(n \log n)$.

Our algorithm uses the algorithmic framework of Kim et al. described in the preceding section. Specifically, we will provide an improved routine for computing the values $t'(\cdot)$ on a given bitree in $O(n \log n)$ time which can then be extended to an algorithm with the same asymptotic running time for the solving indirect covering subtree problem.

Recall the basic steps of the algorithm of Kim et al. Their algorithm partitions the node set V into two sets V_1, V_2 of bounded size such that both induce subtrees and have exactly one node in common. Then it sorts the sets V_i and computes, by means of a clever merge procedure, for all $v \in V_i$ the transportation cost $t'(v, V_j)$ of the users in V_j where $j \neq i$. Applying this recursively to the subtrees induced by V_1, V_2 the algorithm determines the t'-values. Our coarsening algorithm presented in Section 8.2.1 may be regarded as a refinement of this algorithm. Instead of partitioning into two node sets it decomposes the tree into $O(\log n)$ node sets (called super edges).

My observation when developing the following $O(n \log n)$-time algorithm has been that the running time of these approaches would be improved if one could avoid the *explicit sorting*, which causes the additional log-factor. In fact, we will modify the routine such that sorting is no longer performed explicitly but rather by recursion. To this end we will devise a routine that does not only compute $t'(\cdot)$ for a given sub-bitree but also suitably sorted lists of nodes. The difficulty lies in the fact that a direct application of the decomposition technique by Kim et al. would face a possibly superconstant number of such lists to be computed at a single recursion step which compromises the desired asymptotic running time.

Fortunately, we can overcome this problem with the help our two-terminal-subtree technique devised in Section 4.3.3 for computing the Φ-score of a tree and later used in Section 5.3 for computing all Φ-solutions of a tree. This refined decomposition method ensures that we have to determine sorted lists only for the two terminals of a given sub-bitree at any recursion step. Of course this two-terminal property is also satisfied in the coarsening decomposition. However, this approach employed extensive sorting which caused its slow-down.

Let $T = (V, E)$ be the input tree. We assume that T has maximum degree 3. Otherwise, we may split nodes of larger degree by introducing suitable zero-length edges and zero-weighted nodes (confer Section 5.3.2). Let T' be the bitree corresponding to T.

If s and t are distinct nodes then T'_{st} denotes the maximal sub-bitree of T' having s and t as leaves. Let V_{st} be the node set of T'_{st}. We call s and t *terminals* and T'_{st} *two terminal sub-bitree* (TTSB). These definitions are analogous to the ones in Section 4.3.3 for computing Φ-solutions of an (undirected) tree.

Consider an TTSB T'_{st}. We define the lists $L_{d,s}(T'_{st})$ and $L_{\rho,s}(T'_{st})$. Each of them contains all nodes v of T'_{st}. Both lists are sorted in increasing order with respect to the values $d(s, v)$ and $\rho(v) - d(v, s)$, respectively. Lists $L_{d,t}(T'_{st})$ and $L_{\rho,t}(T'_{st})$ are defined symmetrically.

Our algorithm divides the input bitree recursively into TTSBs (confer Section 5.3.1). Since we are dealing with a degree-bounded bitree we can subdivide any TTSB S into at most five child TTSBs by means of Lemma 5.3.2. Each of these child TTSBs has at most $\frac{1}{2}|S| + 1$ nodes.

The algorithm computes $t'(v, S)$ for all $v \in S$ as well as the four lists $L_{d,s}(S)$, $L_{d,t}(S)$, $L_{\rho,s}(S)$ and $L_{\rho,t}(S)$ for any TTSB S occurring during the recursion. We shall see that this information can be propagated inductively from child towards parent TTSBs such that we will have computed $t'(\cdot, T') = t'(\cdot)$ at the top of the recursion.

To this end consider an arbitrary TTSB $S = T'_{st}$ being subdivided into at most five child TTSBs S_i with terminals s_i, t_i. Moreover, assume that we have already computed $t'(\cdot)$ and the four L-lists for each of the child TTSBs.

We start with computing $L_{d,s}(S)$. To this end we maintain a list L which is initialized with an empty list. Now we perform the following operations for all child TTSBs S_i: Assume that s_i is the terminal of S_i closest to s. Then the list $L_{d,s_i}(S_i)$ contains all nodes $v \in S_i$ with associated sorting keys $d_{T'}(s_i, v)$. Now we create a copy L' of this list and add the value $d_{T'}(s, s_i)$ to all sorting keys which does not affect its order. As a result L' contains all nodes v of S_i sorted with respect to their distance $d_{T'}(s, v)$ from terminal s. Finally we merge L with L'. After having carried out this for all child TTSBs S_i the list L equals obviously the list $L_{d,s}(S)$. The list $L_{\rho,s}(S)$ is computed very similarly with the difference that we *subtract* the value $d_{T'}(s_i, s)$ from the sorting keys when handling the list $L_{\rho,s}(S_i)$. The respective lists for terminal t are computed symmetrically. The total running time for computing the four lists associated with S is clearly $O(|S|)$.

We are now going to explain how $t'(v, S)$ can be determined for all $v \in S$. To this end assume that v is contained in S_i. Since we already know $t'(v, S_i)$ by the inductive hypothesis it suffices to determine $t'(v, S_j)$ for all $S_j \neq S_i$ and to add these values to $t'(v, S_i)$. Consider an arbitrary $S_j \neq S_i$. and assume that s_i, s_j are the terminals of these TTSBs closest to each other. We create a copy L' of list $L_{\rho,s_j}(S_j)$ and subtract the distance $d_{T'}(s_j, s_i)$ from all sorting keys in this list. As a result L' contains all nodes u of S_j sorted with respect to the key $\rho(u) - d_{T'}(u, s_i)$. At this point we can compute $t(v, S_j)$ for *all* $v \in S_i$ by using exactly the same merge-and-scan technique as in the algorithm of Kim et al. described in Section 8.2.3. The running time is now $O(|S_i| + |S_j|)$ since the necessary sorted lists have already been computed. Thus we can compute $t'(v, S)$ for all $v \in S$ in total time $O(|S|)$ once we know the t'-values and respective lists for all child TTSBs of S.

Note that the bottom of the recursion, that is, when T'_{st} consists merely of the pair (s, t) and (t, s) of anti-parallel arcs can easily be handled in constant

time.

To sum up, this leads us to an algorithm whose running time $h(|S|)$ can be described by the following recurrence

$$h(|S|) = O(|S|) + \sum_{i=1}^{k} h(|S_i|)\,,$$

where $k \leq 5$, $\sum_{i=1}^{k} |S_i| = |S|$ and $|S_i| \leq \frac{1}{2}|S| + 1$. This implies that $h(n)$ is $O(n \log n)$.

Theorem 8.2.5 *The indirect covering subtree problem and hence also the discrete single maximum coverage location and the discrete $(1, X_p)$-medianoid problem can be solved in time $O(n \log n)$.* □

Kim et al. show in Section 3.3 of [KLTW96] that a set of $O(n)$ critical points for *absolute* single maximum coverage location can be found in time $O(n \log n)$. We can infer immediately

Corollary 8.2.6 *The absolute single maximum coverage location problem and the absolute $(1, X_p)$-medianoid problem can be solved in time $O(n \log n)$ on any n-node tree.* □

8.3. Concluding Remarks

In this chapter we have investigated the complexity of the follower problem on trees. We have started with the observation that (r, X_p)-medianoid can be solved on trees in polynomial time $O(rn^2)$ by a dynamic programming algorithm of Tamir [Tam96]. We have then examined the case of a single follower, that is, the $(1, X_p)$-medianoid problem on trees. First, we have developed a recursive coarsening algorithm that solves the problem in $O(n \log^2 n / \log\log n)$ time and is significantly faster than Tamir's algorithm for the special case $r = 1$. Then we have considered a general model of Kim et al. for locating tree-shaped facilities on a tree. We have briefly described an algorithm of Kim et al. based on the above model for solving the indirect covering subtree problem in $O(n \log^2 n)$ time, which contains $(1, X_p)$-medianoid as a special case. Finally, we have presented an improvement of their approach that results in a surprisingly simple algorithm for solving indirect covering subtree in $O(n \log n)$ time.

It would be interesting to investigate the existence of lower bounds for indirect covering subtree as we have done for the problem of computing

the set of all Φ-solutions in Chapter 5. It would also be worth investigating its complexity on paths.

9. The Leader Problem on Trees and Paths

This chapter is devoted to analyze the complexity of the leader problem, that is, of (r, p)-centroid on trees and path graphs. Indeed, the positive results for the follower problem on trees that were presented in the preceding chapter may awake the hope to obtain similar results for the leader problem, too. But we have also seen in the preceding chapters that the leader problems is almost always significantly harder than the follower problem. In fact, the question of whether (r, p)-centroid is efficiently solvable on trees has been a long-standing open question [Hak90, EL96, Ben00]. In this chapter we show that absolute (r, p)-centroid is already NP-hard on paths. On the other hand, we give a polynomial-time algorithm for *discrete* (r, p)-centroid on paths. To the best of our knowledge this is the first, non-trivial special case where (r, p)-centroid is efficiently solvable. This positive result, however, cannot be generalized significantly. Specifically, we prove that *discrete* (r, p)-centroid is already hard on slightly more complex graphs, namely spiders.

Finally, we will shed some light on the approximability of the leader problem on simple graphs. Specifically, we provide a fully polynomial time approximation scheme (FPTAS) for absolute (r, p)-centroid on paths.

9.1. Absolute (r, p)-Centroid on Paths

In this section we show that the absolute (r, p)-centroid problem is already NP-hard when the underlying graph forms a path. In our hardness proofs we make use of a reduction from the well-known PARTITION problem, which is NP-complete [GJ79]. PARTITION is defined as follows. Given a multiset $S = \{s_1, \ldots, s_n\}$ of integers with total sum $S^* := \sum S$, is there a sub-multiset $S' \subset S$ such that $\sum S' = \frac{1}{2}S^*$?

Let a path graph $G = (V, E)$ be given by its node set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{(v_1, v_2), \ldots, (v_{n-1}, v_n)\}$. We consider the path as a closed interval $[0, c(E)]$ on the real line with length $c(E) = \sum_{e \in E} c(e)$.

Consider a leader placement $X_p = \{x_1, \ldots, x_p\} \subset G$ of $|X_p| = p$ points sorted such that $d(v_1, x_1) < \ldots < d(v_1, x_p)$. This defines a partition of the path into at most $p + 1$ interior-disjoint intervals $T_0 = [v_1, x_1]$, $T_i = [x_i, x_{i+1}]$ for $i = 1, \ldots, p-1$, and $T_p = [x_p, v_n]$.

Consider an interval $[x_i, x_{i+1}]$ of size $t := d(x_i, x_{i+1}) = x_{i+1} - x_i$ induced by some leader placement. By placing one server into that interval, the follower can gain all nodes of any open interval $]a, b[\subset [x_i, x_{i+1}]$ of size $d(a, b) = t/2$. An optimal placement of the follower can be found with a simple linear time sweep algorithm [MZH83].

Theorem 9.1.1 (Absolute (r, p)-centroid on path) *The absolute (r, p)-centroid problem is NP-hard on a path.*

Proof. Let an instance of problem PARTITION be given. Construct a path $P = (a, u_1, v_1, z_1, \ldots, u_n, v_n, z_n, b)$ with $3n+2$ nodes (confer Figure 9.1). To define the weights, let $s_{max} := \max_i s_i$, let $D := 2ns_{max} + 1$, and let $\Omega := 2nD + 1$. Let $w(a) := w(b) := \Omega$, and for $i = 1, \ldots, n$ set $w(u_i) := D$, $w(v_i) := s_i$ and $w(z_i) := \bar{s}_i := D - s_i$. The nodes $u_1, \ldots, u_n$ and $z_1, \ldots, z_n$ are referred to as *heavy*, whereas $v_1, \ldots, v_n$ are called *light* nodes.

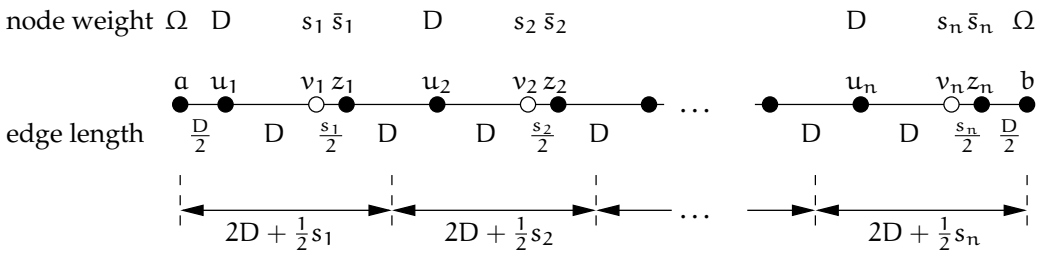

FIG. 9.1.: Illustration of the path construction.

We define the edge lengths as follows: $d(a, u_1) := D/2$, $d(u_i, v_i) := D$, $d(v_i, z_i) := s_i/2$, $d(z_i, u_{i+1}) := D$, and $d(z_n, b) := D/2$. The total length of the path is thus $2nD + S^*/2$.

We set the number of leader positions to $p := n + 1$ and the number of follower positions to $r := n$. We will show in the sequel that there is an (r, p)-centroid of gain $w_{r,p} \leq n \cdot D + \frac{1}{2}S^*$ if and only if the instance of PARTITION admits a subset S' of sum $S^*/2$.

"If": Assume that the instance of PARTITION is solvable with solution S', that is, $\sum S' = \frac{1}{2}S^*$. Place two servers of the leader at the border nodes a, b. The remaining $n - 1$ leader servers partition the path into n intervals T_i of length t_i $(i = 1, \ldots, n)$. The path partition is called *valid* if for each

$i = 1, \ldots, n$ the interval T_i contains the three nodes u_i, v_i, z_i as inner nodes. Choose the server positions such that $t_i := 2D + s_i$ if $s_i \in S'$ ("long interval") and $t_i := 2D$ otherwise ("short interval"). Observe that this yields a valid interval division. The gain of the follower in interval T_i when placing one server is D if it is a short interval and $D + s_i$ if it is a long interval (we assume that the follower chooses positions maximizing his gain). There is no advantage in placing two servers into the same interval as the gain would be $2D$ in that case. Hence we can assume w.l.o.g. that the follower places exactly one server per interval and thus achieves the total gain $nD + \frac{1}{2}S^*$.

"Only if": Consider the case of a leader placement with follower gain $w_{r,p} \leq n \cdot D + \frac{1}{2}S^*$. We claim that the leader chooses a valid interval division.

It is clear that the leader places two servers at the two nodes a, b of weight Ω. Let $(t_1, \ldots, t_n)$ be the sequence of interval lengths of the leader's placement.

Assume for contradiction that the right endpoint of some interval T_i is at the node z_i or to the left of it. The remaining $n - i$ intervals to the right of interval T_i cover a path length of at least $d(z_i, b) > 2(n-i)D + \frac{1}{2}D$, so by averaging there must be an interval of length larger than

$$\Big(2 + \frac{1}{2(n-i)}\Big)D > \Big(2 + \frac{1}{2n}\Big)D > 2D + s_{max}\,.$$

By construction of the path, any interval of length larger than $2D + s_{max}$ contains at least two heavy nodes that are inner nodes and that are within maximum distance of $D + \frac{1}{2}s_{max}$. Hence in that particular interval the follower can gain both heavy nodes by placing a single server. Let $H := \min_i w(z_i) = D - s_{max}$ be the smallest weight among the heavy nodes. Placing the remaining $n - 1$ servers at free heavy nodes, this yields a total gain of at least

$$2H + (n-1)H = nD + \overbrace{D - (n+1)s_{max}}^{(n-1)s_{max}+1} > nD + \frac{1}{2}S^*$$

for the follower, contradicting the premise. By an analogous argument we can show that the left endpoint of interval T_i does not lie at u_i or to the right of it. This shows the claim.

From this property we deduce that each interval left by the leader has inner nodes of total weight $2D$. Since the follower can always gain weight D by placing at u_i, we can assume w.l.o.g. that the follower places exactly one server into each interval. Moreover the length of each interval T_i is bounded from above by $2D + s_i$, otherwise the follower could cover all inner nodes

of T_i with a single server which would lead to a total gain of at least $2D + (n-1)H > (n+1)H$ contradicting the premise.

We distinguish two kinds of intervals, namely those of length $t_i \leq 2D$, which we call *short intervals*, and those of length $2D < t_i \leq 2D + s_i$, called *long intervals*. We define the set $S' \subseteq S$ to be the set of those s_i where T_i is a long interval. As argued above the follower places exactly one server into each interval T_i. This defines for each interval a number w_i denoting the follower's gain in that interval. Obviously $w_i = D$ for short intervals and $w_i = D + s_i$ for long intervals. This yields $t_i - D \leq w_i$. Hence

$$\frac{S^*}{2} = \sum_{i=1}^{n} (t_i - 2D) \leq \sum_{i=1}^{n} (w_i - D) \leq \frac{S^*}{2}$$

where the first equality follows from the fact that the path has length $2nD + \frac{1}{2}S^*$, and the last inequality follows from the premise $w_{r,p} \leq nD + S^*/2$. Thus we can conclude that

$$\sum S' = \sum_{i=1}^{n} \underbrace{(w_i - D)}_{\substack{0 \text{ for short int.} \\ s_i \text{ for long int.}}} = \frac{S^*}{2}$$

which completes the proof. □

9.2. Discrete (r, p)-Centroid on Paths

Many optimization problems exhibit an *optimal substructure property* [CLR90] (or *principle of optimality* [ACG+99]): essentially this means that a problem instance can be separated into independent subproblems such that optimal solutions of these subproblems can be combined to solve the original problem optimally. This property is exploited by widespread algorithmic techniques like divide and conquer, greedy, or dynamic programming.

In the case of the discrete (r, p)-centroid problem on a path this suggests the following approach. Consider a path P with an (r, p)-centroid X_p and a node $x \in X_p$. Let P_1, P_2 be the subpaths resulting from splitting P at x. One could suspect that for suitable p_1, p_2, r_1, r_2 there are (r_i, p_i)-centroids on P_i such that their union forms an (r, p)-centroid on P, with the reasoning that no user in one subpath ever patronizes any server on the other subpath.

The following example, however, shows that the (r, p)-centroid problem on paths does not exhibit the optimal substructure property even when $r =$

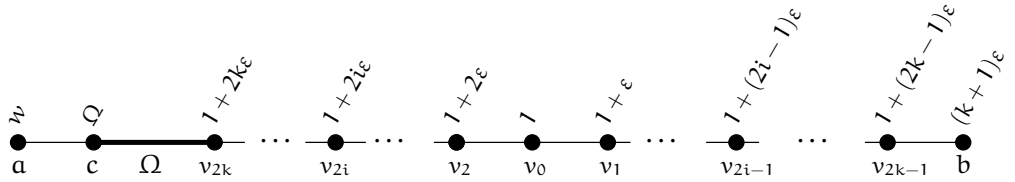

FIG. 9.2.: $(2,2)$-centroid does not satisfy the optimal substructure property.

$p = 2$. Confer Figure 9.2. The path consists of $2k+1$ nodes $v_0, \ldots, v_{2k}$ ordered such that v_0 is the central node and all nodes with even index are ascending to the left and those with odd index ascending to the right. For $0 \leq i \leq 2k$, node v_i has weight $1+i\cdot\varepsilon$ for some small $\varepsilon > 0$. The left end is augmented by two nodes a, c of weight w and some large constant Ω, respectively, and the right end by a node b of weight $(k+1)\varepsilon$. The edge (c, v_{2k}) has length Ω while all other edges are of length 1. Let $W := \sum_{i=0}^{2k} w(v_i) + w(b)$. We are going to show that changing the weight $w = w(a)$ within the interval $[1, \frac{1}{2}(W-1-\varepsilon)]$ can enforce any node v_i to become part of the $(2,2)$-centroid.

Since $w \leq \frac{1}{2}(W-1-\varepsilon)$ one can see that the leader always places one server at node c and the other server at one of the nodes v_i. For $r = 1, 2$ let $w_r(i)$ be the maximum weight that the follower can claim when the leader places at v_i and the follower places r servers on the node set $V - \{a\}$. By elementary calculations it follows that

$$\begin{aligned} w_1(2i-1) &= W - k + i - k(k+1)\varepsilon + (i^2-1)\varepsilon \\ w_1(2i) &= W - k + i - k(k+1)\varepsilon + i(i+1)\varepsilon \\ w_2(i) &= W - 1 - i\varepsilon \end{aligned}$$

which shows that w_1 is strictly increasing with i while w_2 is strictly decreasing. On the subpath $V-\{a\}$ the $(1,2)$-centroid is $\{c, v_0\}$ and the $(2,2)$-centroid is $\{c, v_{2k}\}$.

We now turn our view back to the whole path. The optimal substructure property would imply that regardless of the weight w of node a there is a $(2,2)$-centroid that contains either v_0 or v_{2k}. This, however, is not true. If the weight of a is set to $w := w_2(i) - w_1(i)$ for some i then $\{c, v_i\}$ is the unique $(2,2)$-centroid on the whole path. This is easy to verify: First it is clear that $w_2(\{c, v_i\}) = w_2(i)$. Consider $w_2(\{c, v_j\})$ for $j \neq i$. If $j > i$ then the follower places at a and gains $w + w_1(j) = w_2(i) - w_1(i) + w_1(j) > w_2(i)$. If $j < i$ then the follower places both servers near v_j and gains at least $w_2(j) > w_2(i)$.

This is a surprising paradox: When the path is split at node c, which is always part of a $(2,2)$-centroid, changes in the weight of node a affect the so-

lution in the other subpath. Moreover, from the view of node a a user on this node never connects to any server placed on the right subpath $V - \{a\}$; one thus would not expect it to have any influence on the decisions local to that subpath. As a consequence, a straightforward application of divide-and-conquer techniques cannot be successful in attacking the centroid problem on a path.

The Algorithm

Let G be the input path with ordered vertex set $V = \{v_1, \ldots, v_n\}$. In order to compute a discrete (r, p)-centroid, we reduce this problem to the k-*sum shortest path problem*, which has been solved by Punnen and Anneja [PA96] within a framework for general k-*sum optimization* problems where the underlying *minisum* problem is efficiently computable. We will formally define k-sum optimization problems and minisum problems later (see Definition 9.4.1).

Definition 9.2.1 (k-sum shortest path) Given a positive integer k, a digraph G with positive arc lengths, and nodes s and t in G. A k-sum shortest (s, t)-path is a path from s to t where the sum of the k longest arcs is as small as possible.

We define a new digraph G' as depicted in Figure 9.3. Start with a node set $V' := \{u_{ij} \mid i = 1, \ldots, n \text{ and } j = 1, \ldots, p\}$. For any $i, j \in \{1, \ldots, n\}$, $i < j$, and any $k \in \{1, \ldots, p-1\}$ add a path of two consecutive arcs (introducing a new vertex in the middle) from $u_{i,k}$ to $u_{j,k+1}$. This models the case that the leader places the kth server at v_i and the next server at v_j. Moreover, add new super nodes s, t to the graph and add arcs from s to $u_{11}, \ldots, u_{n1}$ and from $u_{1p}, \ldots, u_{np}$ to t.

The lengths of the arcs are determined by the gain of the follower on partial intervals. Let $w_1(i, j)$ denote the maximum weight that a single follower server can claim on the partial interval between two consecutive leader servers placed at v_i and v_j. Similarly, let $w_2(i, j) = \sum_{l=i+1}^{j-1} w(v_l)$ be the maximum weight that can be claimed with two follower servers. For any path of two arcs connecting u_{ik} to $u_{j,k+1}$, set the length of the first arc to $w_1(i, j)$ and the length of the second arc to $w_2(i, j) - w_1(i, j)$. Finally, for $i = 1, \ldots, n$, set the length of arc (s, u_{i1}) to $\sum_{l=1}^{i-1} w(v_l)$ and that of arc (u_{ip}, t) to $\sum_{l=i+1}^{n} w(v_l)$. This completes the construction of the acyclic graph G'.

Lemma 9.2.2 *The r-sum length of an s-t-path through nodes $u_{i_1,1}, \ldots, u_{i_p,p}$ equals the follower gain $w_r(X_p)$ where $X_p = \{v_{i_1}, \ldots, v_{i_p}\}$.*

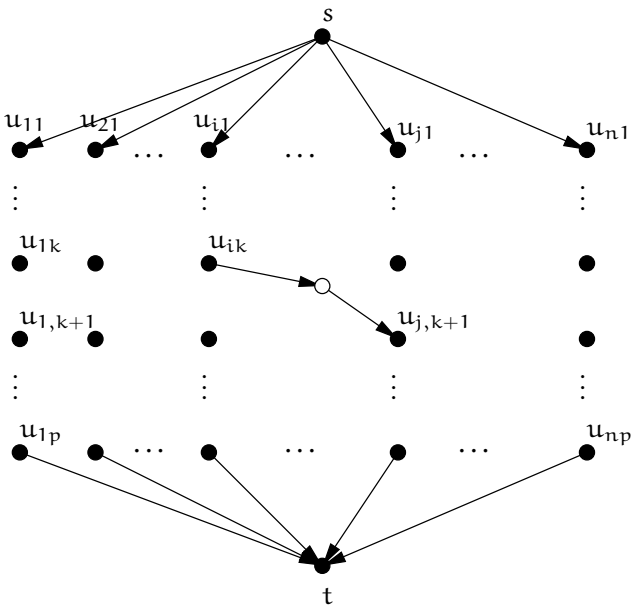

FIG. 9.3.: Auxiliary graph to solve the discrete (r, p)-centroid on a path.

Proof. By construction, any (s, t)-path in G' meets exactly p nodes of the initial node set V'. This establishes a one-to-one relationship between placements of the p servers of the leader and (s, t)-paths in the auxiliary graph.

First, we observe that the follower has no incentive to place more than two servers into one interval because two servers are already sufficient to gain interior nodes of that interval. Now observe that for any $i < j$, $w_1(i, j) \leq w_2(i, j) \leq 2w_1(i, j)$. Therefore the follower can achieve the maximum gain by a simple greedy strategy: given the $p + 1$ intervals left by the leader, determine for each interval (u_i, u_j) the (incremental) gain $\delta_1(i, j) := w_1(i, j)$ of placing one server and the incremental gain $\delta_2(i, j) := w_2(i, j) - w_1(i, j) \leq \delta_1(i, j)$ of placing two servers. The follower gain $w_r(X_p)$ is the sum w_{max} of the r largest numbers out of the multiset of all incremental gains, which is also the r-sum length of the s-t-path in G'. This can be seen as follows. The gain $w_r(X_p)$ can be expressed as the sum of r incremental gains. Hence the follower gain cannot be larger than w_{max}. On the other hand, this value can indeed be realized by a follower placement since $\delta_1(i, j) \geq \delta_2(i, j)$ for each interval (u_i, u_j) as observed above. (If $\delta_2(i, j) > \delta_1(i, j)$ then it could happen that $\delta_2(i, j)$ is included in the sum achieving w_{max} but $\delta_1(i, j)$ is not. Then w_{max} could not be realized by an r-element follower placement.) □

The (r,p)-centroid minimizes the follower gain $w_r(X_p)$ over all server placements X_p, which corresponds to an r-sum minimization of paths in the graph G': An r-sum shortest (s,t)-path in graph G' is equivalent to a solution of the (r,p)-centroid problem on path G.

Theorem 9.2.3 (Discrete (r,p)-centroid on path) *A discrete (r,p)-centroid of an n-node path can be found in $O(pn^4)$ time.*

Proof. Punnen and Aneja [PA96] showed that the k-sum optimization problem can be solved in $O(M \cdot t)$ time where M is the number of different weights of items in the ground set and t is the time needed for solving one instance of the underlying minisum problem. In our setting the set of ground elements is the set of arcs of size $O(pn^2)$ but with only $O(n^2)$ different weights. The minisum problem (shortest s-t-path in an acyclic graph of $O(pn^2)$ arcs) can be solved in time $O(pn^2)$. □

In Theorem 7.2.2 we have characterized a set of so-called *critical points* for any instance (r,p)-centroid problem, which is guaranteed to contain an optimal leader placement. Unfortunately this point set is exponentially in size and thus not immediately amenable for algorithmic purposes. It would be much more desirable to have an efficiently computable set of critical points at hand such as the X_p-isodistant points for the follower problem. Santos-Peñate et al. [SSD07] give an overview of discretization results for more sophisticated variants of the follower problem. The authors also remark that, in contrast to the follower problem, discretization results are rather scarce for (r,p)-centroid. The results on absolute and discrete (r,p)-centroid on paths justify this observation. In fact it follows immediately that no efficient discretization is possible even on paths: If there were an algorithm enumerating critical points for any instance of the absolute leader problem then the absolute leader problem could be solved efficiently by the algorithm for the discrete case.

9.3. Discrete (r,p)-Centroid on Trees

In this section we are going to show that determining a discrete (r,p)-centroid is NP-hard on a spider, that is, a tree where only one node has degree larger than 2.

Theorem 9.3.1 (Hardness of (r,p)-centroid on a spider) *The problem of determining a discrete (r,p)-centroid on a spider is NP-hard.*

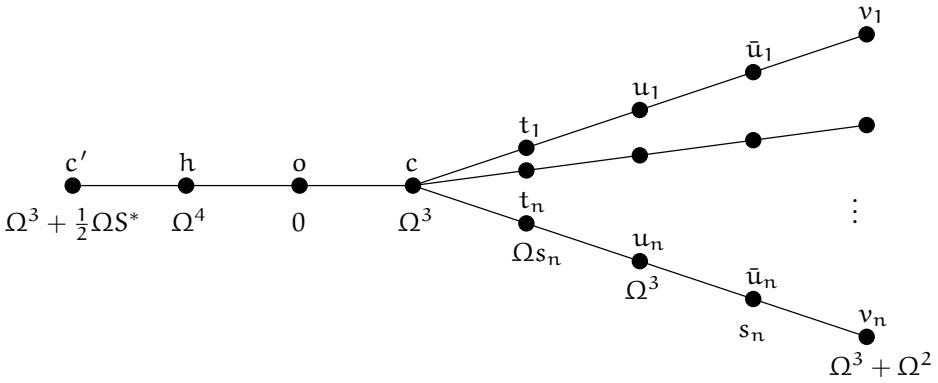

FIG. 9.4.: Discrete (r, p)-centroid is NP-hard on a spider.

Proof. Let an instance of the NP-complete problem PARTITION be given as in Section 9.1. We construct a spider as depicted in Figure 9.4. The node set consists of a central node c and for $i = 1, \ldots, n$, of a leg with nodes c–t_i–u_i–$\bar{u}_i$–v_i. The weights of the nodes are $w(c) := \Omega^3$, $w(t_i) := \Omega s_i$, $w(u_i) := \Omega^3$, $w(\bar{u}_i) := s_i$, and $w(v_i) := \Omega^3 + \Omega^2$. Finally, we add a special leg c–o–h–c' of weight $w(h) := \Omega^4$, $w(o) := 0$, and $w(c') := \Omega^3 + \frac{1}{2}\Omega S^*$. Here we choose $\Omega := 1 + nS^*$. All edges have unit length.

We set $r := p := n + 1$ and claim that there is an $(n + 1, n + 1)$-centroid of weight

$$W := (n + 1)\Omega^3 + n\Omega^2 + \frac{1}{2}S^*(\Omega + 1)$$

if and only if the instance of PARTITION is a "yes"-instance.

"If": Let $S' \subseteq S$ with $\sum S' = \frac{1}{2}S^*$. Place leader facilities at h, and, for each i at $\bar{u}_i$ if $s_i \in S'$ and at u_i otherwise. Considering the gain of the follower observe that it is not possible that the follower claims c and one of the u_i with a single server only. Since

$$w(c) + \sum_j w(t_j) = \Omega^3 + \Omega S^* < \Omega^3 + \Omega^2 = w(v_i),$$

for any i, it is optimal to claim all peripheral nodes v_i. This is accomplished by placing at v_i if $s_i \in S'$ and at $\bar{u}_i$ otherwise. This way the follower claims all nodes v_i, $i = 1, \ldots, n$, and the nodes $\bar{u}_i$ where $s_i \notin S'$, with a total weight of

$$n(\Omega^3 + \Omega^2) + S^* - \sum S' = n(\Omega^3 + \Omega^2) + \frac{1}{2}S^*.$$

The remaining server can be placed either at c' or at the central node c where it claims c and each node t_i with $s_i \in S'$. This contributes a weight of

$$\Omega^3 + \Omega \sum S' = \Omega^3 + \frac{1}{2}\Omega S^*$$

which is the same for both cases. Adding both terms shows that the total weight of the (r, p)-centroid is exactly equal to W.

"Only if": In an optimal solution it is obvious that the leader places one server at the node h of weight Ω^4. Further observe that there are enough nodes of weight Ω^3 or greater (namely the $2n + 2$ nodes u_i, v_i, c, c') such that the follower can always place only at those nodes and thus gain at least Ω^3 per server.

We claim that the leader chooses on each leg either the node u_i or $\bar{u}_i$: If the leader places a server at central node c or at one of the t_i, then there are $n-1$ additional servers left to place. This would leave at least one leg j free to the follower so that he could place at node u_j and gain both u_j and v_j of weight more than $2\Omega^3$ with a single server, resulting in a total of more than $(n+2)\Omega^3$. As a consequence, the leader must choose, on each leg, either u_i, or $\bar{u}_i$, or v_i. If the leader would place at the peripheral node v_i, then the follower could place at t_i which would claim both u_i and the central node c with this server, which yields a similar contradiction. This shows the claim.

Let $S' := \{ s_i \mid \text{leader places server at } \bar{u}_i \} \subseteq S$ the set of items where the leader places a server at the outer node in the corresponding leg. Suppose $\sum S' > \frac{1}{2}S^*$. Then the follower places, for $i = 1, \ldots, n$, on leg i next to the leader. In doing so, he claims the node v_i, and, if $s_i \notin S'$, node $\bar{u}_i$. The remaining server is placed at the central node c and claims the nodes t_i where $s_i \in S'$. This yields a follower gain of

$$n(\Omega^3+\Omega^2)+\underbrace{(S^* - \sum S')}_{>0}+\Omega^3+\Omega \underbrace{\sum S'}_{\geq S^*/2+1} > n(\Omega^3+\Omega^2)+\Omega^3+(\Omega+1)\frac{1}{2}S^* = W$$

where we make use of $\sum S' \geq \frac{1}{2}S^* + 1$ and $\Omega > S^*$. Suppose $\sum S' < \frac{1}{2}S^*$. Like above the follower places n servers on the periphery; the remaining server is placed at c'. This yields a gain of

$$n(\Omega^3 + \Omega^2) + \underbrace{(S^* - \sum S')}_{>S^*/2} + \Omega^3 + \frac{1}{2}\Omega S^* > W\,.$$

This completes the proof. □

9.4. An FPTAS for (r, p)-Centroid on Paths

Recall that the leader problem has quite unsatisfactory approximability properties on general graphs. Specifically, we have shown in Section 7.4.2 that (r, p)-centroid is not approximable within a factor of $n^{1-\varepsilon}$ unless $P = NP$. This suggests to investigate the approximability on simpler graph classes. In this section we consider the problem on path graphs since it is NP-hard even in this simple special case as shown in Section 9.1. In particular, we show that absolute (r, p)-centroid admits a fully polynomial time approximation scheme (FPTAS) on a path graph. This means that we can find an $(1 + \varepsilon)$-approximate solution for any $\varepsilon > 0$ in time polynomial in the input size and $\frac{1}{\varepsilon}$. This is, in a certain sense, the best possible approximative behavior we may expect from an NP-hard optimization problem. The main idea of our approach is to develop first a pseudo-polynomial dynamic program for absolute (r, p)-centroid which can then be extended to an FPTAS by means of scaling techniques [PS98].

In the sequel we shall assume that all node weights and edge lengths are positive integers which can be achieved by multiplying all fractions in the input with their least common denominator. Now let $G = (v_1, \ldots, v_n)$ be the input path, $w\colon V \to \mathbb{N}_+$ the node weights, and $c\colon E \to \mathbb{N}_+$ the edge lengths.

As in the algorithm for discrete (r, p)-centroid on a path we will use the framework for k-*sum optimization* of Punnen and Anneja [PA96], which allows us to decompose any problem instance into smaller subproblems. Recall that a naive implementation of such an approach does not work as pointed out in Section 9.2.

The general definition of a k-sum optimization problem is as follows:

Definition 9.4.1 (k-sum optimization problem [PA96]) An instance of a k-*sum optimization problem* is given by a set M of *ground elements*, a family $\mathcal{F} \subseteq 2^M$ of subsets of M and a cost function $c\colon M \to \mathbb{N}^+$. Let $S = \{s_1, \ldots, s_l\}$ be a set in $\mathcal{F}$. Assume that $c(s_1) \geq c(s_2) \geq \ldots \geq c(s_l)$ holds. Then we denote by $c_k(S)$ the sum $\sum_{i=1}^{k} c(s_i)$ of the k most expensive elements of S. The goal is to identify a set $S \in \mathcal{F}$ such that $c_k(S)$ is minimum.

In most applications of k-sum optimization the set $\mathcal{F}$ is not given explicitly. Consider for example the k-sum shortest path problem (which we used in Section 9.2 to solve discrete (r, p)-centroid) where M is the set of edges of the input graph and $\mathcal{F}$ is the family of all edge sets of paths connecting some given node pair s and t.

The main result of Punnen and Anneja [PA96] is that a k-sum problem is efficiently solvable if this holds for the *corresponding minisum problem*. An

instance of the corresponding minisum problem is specified by the input data of the k-sum problem and additionally by a non-negative integer z. Given some ground element m we define its *z-reduced cost* by $c^{(z)}(m) := c(m) \dot{-} z$. Here, $x \dot{-} y := \max\{x - y, 0\}$ denotes the *asymmetric difference* of x and y. The goal of the minisum problem is to identify a set $S \in \mathcal{F}$ such that $c^{(z)}(S) := \sum_{s \in S} c^{(z)}(s)$ is minimum.

Punnen and Anneja show the following main theorem.

Theorem 9.4.2 ([PA96]) *Let a* k*-sum optimization problem with ground set* M *be given and assume that we can solve the corresponding minisum problem in* $O(f(|M|))$ *time. Then the* k*-sum problem can be solved in* $O(|M|f(|M|) + |M|^2)$ *time.* □

We are now going to formulate (r, p)-centroid as a k-sum optimization problem in order to apply the above theorem of Punnen et al. As in the NP-hardness proof of (r, p)-centroid on paths in Section 9.1 we identify the path G with an interval $[0, c(G)]$ on the real line. Now consider an arbitrary leader positioning $X_p = \{x_1, \ldots, x_p\}$. We assume w.l.o.g. that $x_1 < \ldots < x_p$ holds where we identify the points x_i with real numbers. Moreover, we assume that $x_1 = v_1$ and $x_p = v_n$, that is, that the leader occupies the end nodes of path G. This can be enforced by adding dummy nodes that are sufficiently far away and have sufficiently large weight, and by incrementing p accordingly.

Consider now any leader placement $X_p = \{x_1, \ldots, x_p\}$. For any interval $[x_i, x_{i+1}]$ we denote by $w_j(x_i, x_{i+1})$ the maximum gain of the follower when he places $j = 1, 2$ servers into the interval $[x_i, x_{i+1}]$ after the leader has located servers at the endpoints x_i, x_{i+1} of the interval. Obviously, $w_2(x_i, x_{i+1})$ equals the total demand within the open interval $]x_i, x_{i+1}[$, that is, the weight of all nodes lying in the interior of this interval. It is easy to see (confer also Section 9.1) that the value $w_1(x_i, x_{i+1})$ equals the weight of the heaviest open sub-interval $]a, b[\subset [x_i, x_{i+1}]$ of length $d(a, b) = d(x_i, x_{i+1})/2$. It is therefore clear that $w_1(x_i, x_{i+1}) \geq \frac{1}{2} w_2(x_i, x_{i+1})$.

We define now the incremental follower gain (confer also proof of Lemma 9.2.2) $\delta_1(x_i, x_{i+1}) := w_1(x_i, x_{i+1})$ and $\delta_2(x_i, x_{i+1}) := w_2(x_i, x_{i+1}) - w_1(x_i, x_{i+1})$ when the follower places one and two servers into the interval, respectively. The above considerations imply that $\delta_1(x_i, x_{i+1}) \geq \delta_2(x_i, x_{i+1})$. Hence, the maximum gain $w_r(X_p)$ of the follower when he locates r servers in the presence of placement X_p is exactly the sum w_{max} of the r largest elements of the multiset

$$\delta(X_p) := \{\, \delta_j(x_i, x_{i+1}) \mid i = 1, \ldots, p-1 \text{ and } j = 1, 2 \,\}$$

of all incremental follower gains. This can be seen as follows: The gain $w_r(X_p)$ can be expressed as the sum of r incremental gains from the multiset $\delta(X_p)$. Hence the follower gain can not be larger than w_{max}. On the other hand, this value can indeed be achieved since $\delta_1(x_i, x_{i+1}) \geq \delta_2(x_i, x_{i+1})$ for each interval $[x_i, x_{i+1}]$ as observed above.

This suggests to formulate (r, p)-centroid as a k-sum optimization problem in the following way: The family $\mathcal{F}$ consists of all of multisets $\delta(X_p)$ where $X_p \subset G$ is some p-element point set containing the end nodes of the input path. The set $M := \bigcup \mathcal{F} \subset \mathbb{N}$ of ground elements contains all incremental gains associated with some placement. Since the ground elements represent the costs themselves the cost function $c\colon M \to \mathbb{N}$ is simply the identity. Finding an (r, p)-centroid is then equivalent to identifying some set $\delta(X_p)$ for which $c_r(X_p)$ is minimum. In the terminology of Punnen et al. (r, p)-centroid is an r-*sum* optimization problem. Note that our algorithm will not deal with this representation in an explicit way.

Given a placement X_p we will denote the elements of $\delta(X_p)$ by $\delta_1(X_p) \geq \ldots \geq \delta_{2p-2}(X_p)$. Note that $\delta_i(X_p) = w_i(X_p) - w_{i-1}(X_p)$. Now, we are going to consider the *minisum problem* corresponding to (r, p)-centroid. In our context we define the *z-reduced follower gain* by

$$w^{(z)}(X_p) := \sum_{i=1}^{2p-2} (\delta_i(X_p) \dot{-} z)$$

for any $z \in \mathbb{N}$. The goal of the corresponding minisum problem is to find a placement X_p such that $w^{(z)}(X_p)$ is minimum. According to Theorem 9.4.2 it suffices to solve this minisum problem. We will show that this can be done in pseudo-polynomial time by means of a dynamic programming approach.

Consider two points a, b lying on the input path G and a point set $X \subseteq P$ where $P = [a, b]$ is the subpath between a and b. Then we denote by $w_P^{(z)}(X)$ the z-reduced follower gain of X locally on path P, that is, when P is considered as a separate path independent from G. Let x be some point in X such that $a < x < b$. The point x partitions X into two sets X_l and X_r containing all elements $\leq x$ and $\geq x$, respectively. The crucial point is now that the minisum formulation, in contrast to the r-sum version, does satisfy the optimal substructure property (confer also Section 9.2) in the sense that the relation $w_{[a,b]}^{(z)}(X) = w_{[a,x]}^{(z)}(X_l) + w_{[x,b]}^{(z)}(X_r)$ holds.

Recall that an analogous relation is not true for the r-sum formulation since here, the follower has to distribute his r facilities to both sub-intervals. Therefore, the respective gains in these intervals does not need to sum up when switching to the complete interval $[a, b]$.

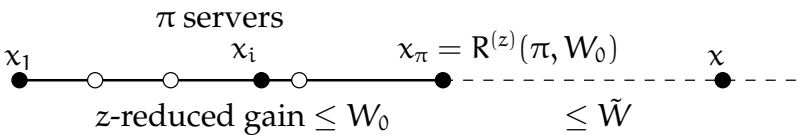

FIG. 9.5.: Illustration of Equation (9.1)

We are now going to exploit the decomposability property of the minisum problem to devise a dynamic program for solving it in pseudo-polynomial time. We use the notation $w^{(z)}(x_i, x_{i+1}) := w^{(z)}_{[x_i,x_{i+1}]}(\{x_i, x_{i+1}\})$ to denote the z-reduced gain in interval $[x_i, x_{i+1}]$ when the leader occupies its endpoints x_i, x_{i+1}.

Assume the path is populated incrementally with leader servers from left to right. Let $R^{(z)}(\pi, W) =: R$ denote the rightmost feasible leader position of placing π servers such that $w^{(z)}$ restricted to the interval $[0, R]$ does not exceed W. More formally, R is the maximum real number that fulfills the following property: Consider the subpath P of the input path induced by the interval $[0, R]$. Then there is a π-element leader placement X_π containing the point R such that $w^{(z)}_P(X_\pi) \leq W$.

By the optimal substructure property we obtain the following simple equation for any $\pi = 1, \ldots, p-1$ and $W = 0, \ldots, w(G)$,

$$R^{(z)}(\pi+1, W) = \max \left\{ x \in G \;\middle|\; \begin{array}{l} \exists (W_0, \tilde{W}) \colon W_0 + \tilde{W} = W \quad \text{and} \\ w^{(z)}\big(R^{(z)}(\pi, W_0), x\big) \leq \tilde{W} \end{array} \right\}. \tag{9.1}$$

For an illustration of the situation referred to by this equation confer Figure 9.5.

We are now going to compute the values $R^{(z)}(\cdot,\cdot)$ systematically by means of dynamic programming techniques. The relevance of this should be clear since the $w^{(z)}$-optimum weight can be derived from the outcome vector $R^{(z)}(p, \cdot)$ of the dynamic programming table by

$$\min_{X_p} w^{(z)}(X_p) = \min\{\, W \mid R^{(z)}(p, W) = d(v_1, v_n) \,\};$$

a corresponding leader server placement X_p can be derived from maintaining the positions during the dynamic program.

The dynamic program is depicted in Figure 9.6 and 9.7. It iterates over increasing values for $\pi = 1, \ldots, p$. For $\pi = 1$ we observe that a facility is always located at v_1 and thus we need only to store the entry $R^{(z)}(1, 0) = 0$ in the dynamic programming table. Now we assume that we have already filled our table for parameter π. We proceed to $\pi + 1$ by iterating over all

```
1  input: instance of (r, p)-centroid with path G, a non-
   negative integer z
2
3  W̄ ← w(V − v_1 − v_n)
4  for all W_0 = 1, ..., W̄ and π = 1, ..., p
5      R(π, W_0) ←⊥
6  end for
7
8  R(1, 0) ← 0
9  for π := 1 to p − 1
10     for W_0 := 0 to W̄
11         if R(π, W_0) = ⊥ then
12             continue with next W_0
13         x_π ← R(π, W_0)
14         scanPathStartingFrom(x_π)
15     end for
16 end for
17
18 output min{ W | R(p, W) = d(v_1, v_n) }
```

FIG. 9.6.: Dynamic program for solving the minisum problem corresponding to (r, p)-centroid

entries $R^{(z)}(\pi, W_0)$ stored in the table. For each such entry $x_\pi := R^{(z)}(\pi, W_0)$ we systematically *scan* (confer Figure 9.7) the points $x > x_\pi$ from left to right and calculate the z-reduced weight $\tilde{W} := w^{(z)}(x_\pi, x)$ of the interval $[x_\pi, x]$. Whenever we reach a point x where the z-reduced weight changes this defines one more interesting position $x_{\pi+1} := x$ for the $(\pi+1)$st server, and we store the entry $R^{(z)}(\pi+1, W_0 + \tilde{W}) := x_{\pi+1}$.

It remains to describe in detail how the *scans* mentioned above can be performed efficiently. An implementation of this procedure is depicted in Figure 9.7 on page 165. To this end consider a fixed point x_π. We generate a list of $O(n^2)$ points which is guaranteed to contain all points x where $w^{(z)}(x_\pi, x)$ changes. These points are stored in the list L_{scan} in the algorithm. It is clear that $w^{(z)}(x_\pi, \cdot)$ can only change at positions where so does $w_1(x_\pi, \cdot)$ or $w_2(x_\pi, \cdot)$.

Recall that $w_2(x_\pi, x)$ equals the weight of the open interval $]x_\pi, x[$, that is, the weight of all nodes lying in the interior of this interval. Thus there are

```
1  procedure scanPathStartingFrom(x_π)
2
3  L_scan ← { v ∈ V | v ≥ x_π }
4  sort L_scan
5  let L_pairs be the list of all node pairs (u, v)
6          sorted increasingly w.r.t. primary sort key d(u, v)
7          and decreasingly w.r.t. secondary sort key w(P(u, v))
8  for all pairs (u, v) in L_pairs in sorting order order
9      x_{π+1} ← x_π + 2 · d(u, v)
10     if x_π ≤ u ≤ v ≤ x_{π+1} and x_{π+1} ∉ L_scan
11         insert x_{π+1} into L_scan
12     end if
13 end for
14
15 W̃ ← 0
16 w_i ← 0 for i = 1, 2
17 for all points x_{π+1} ∈ L_scan in sorting order
18     R(π + 1, W_0 + W̃) ← x_{π+1}
19     if x_{π+1} is a node
20         w_2 ← w_2 + w(x_{π+1})
21     end if
22     if x_{π+1} has been induced by pair (u, v)
23         w_1 ← max{w_1, w(P(u, v))}
24     end if
25     W̃ ← (w_1 ∸ z) + ((w_2 − w_1) ∸ z)
26 end for
```

FIG. 9.7.: Subroutine scanPathStartingFrom

$O(n)$ points, namely all nodes $v \geq x_\pi$, where $w_2(x_\pi, \cdot)$ can change (confer line 3).

The value $w_1(x_\pi, x)$ equals the weight of the heaviest open sub-interval $]a, b[\subset [x_\pi, x]$ of length $d(a, b) = d(x_\pi, x)/2$. To enumerate all critical points corresponding to changes of $w_1(x_\pi, \cdot)$ we use a list L_{pairs} of all node pairs (u, v). This list is created in a preprocessing step and sorted with respect to $d(u, v)$. The list L_{scan} of points where $w_1(x_\pi, \cdot)$ could change contains all $x_{\pi+1} := x_\pi + 2 \cdot d(u, v)$ where $x_\pi \leq u \leq v \leq x_{\pi+1}$ (confer loop in line 8). We say that (u, v) *induces* $x_{\pi+1}$ and remember for each such $x_{\pi+1}$ by which pair (u, v) it has been induced. The secondary sort key $w(P(u, v))$ of list L_{pairs} ensures that we remember only the pair (u, v) with the heaviest subpath $P(u, v)$. In fact, the gain $w_1(x_\pi, x_{\pi+1})$ is just the weight of the heaviest sub-path $P(u, v)$ inducing critical points in the interval $]x_\pi, x_{\pi+1}[$. Thus we can compute all the reduced follower gains $w^{(z)}(x_\pi, \cdot)$ in total time $O(n^2)$ by a linear scan through the sorted list L_{scan} of all critical points (confer loop in line 17) while maintaining the weight w_1 the heaviest of such subpaths $P(u, v)$ inspected so far as well as the weight w_2 of the path $]x_\pi, x_{\pi+1}[$ (confer lines 20 and 23).

The pseudo-polynomial running time of the above dynamic program follows easily from Theorem 9.4.2.

Lemma 9.4.3 *The problem of determining $w_{r,p}$ and a corresponding leader placement X_p (that is, where $w_r(X_p) = w_{r,p}$ is attained) can be solved in pseudo-polynomial running time $O(p \cdot n^2 \cdot w(G)^2)$.*

Proof. The correctness follows from the above discussion.

To analyze the running time we consider any inductive step $\pi \mapsto \pi + 1$ described above. During such a step we inspect $O(w(G))$ entries $R(\pi + 1, \cdot)$ and perform the scan routine for each of them. This takes $O(w(G) \cdot n^2)$. Since we perform p such inductive steps we obtain time $O(pn^2 w(G))$ for solving the minisum problem. The preprocessing of sorting the all node pairs takes $O(n^2 \log n)$ and is dominated by the main routine since we may assume that $w(G)$ is $\Omega(n)$.

The time needed to solve the r-sum problem, which is the actual (r, p)-centroid problem, follows immediately from Theorem 9.4.2 since the ground elements of the minisum problem are the incremental gains which vary in a domain of $O(w(G))$ elements. □

From this result we can derive a fully polynomial time approximation scheme (FPTAS) applying a scaling technique to the weights of the nodes. The algorithm is based on the general approach for deriving an FPTAS from

an pseudo-polynomial time algorithm [PS98]. However, there are some details that require additional work.

Theorem 9.4.4 (Approximation) *There is an FPTAS for absolute* (r, p)*-centroid on a path with running time* $O(pn^6\frac{1}{\varepsilon^2})$.

Proof. Let $K \in \mathbb{N}$ and assume that we scale down (with rounding) the node weights w by factor K, that is, we use the weight function $w' \equiv \lfloor \frac{w}{K} \rfloor$ rather than w. We use the notations $w'(Y_r \prec X_p)$, $w'_r(X_p)$ and $w'_{r,p}$ when referring to these modified weights. We will show that K can be chosen to ensure an approximation ratio of $1+\varepsilon$ and simultaneously a running time polynomial in n and $\frac{1}{\varepsilon}$.

To this end let X_p and X'_p be the (r, p) centroid w.r.t. weights w and w', respectively. Consider the (r, X'_p)-medianoid Y_r w.r.t. the *unmodified* weights w. Then we have

$$\begin{aligned} w_r(X'_p) &= w(Y_r \prec X'_p) \\ &\leq K \cdot (w'(Y_r \prec X'_p) + n) \\ &\leq K \cdot w'_r(X'_p) + nK \\ &\leq K \cdot w'_r(X_p) + nK \\ &\leq w_r(X_p) + nK\,. \end{aligned}$$

Here, the inequality in the second line follows from the observations that the term $w_r(Y_r \prec X'_p)$ is a sum of at most n node weights and that $w(v) \leq K \cdot w'(v) + 1$ holds for any node v.

From this equation we infer immediately that X'_p is a $(1+\varepsilon)$-approximation if $nK \leq \varepsilon \cdot w_r(X_p)$. Of course, we do not know $w_r(X_p)$ so we need a reasonable lower bound for it. To this end consider the set X''_p consisting of the heaviest p nodes on path G. Moreover let $\tilde{v}$ be an heaviest node in $V - X''_p$. It is now easy to see that $w_r(X_p) \geq w(\tilde{v})$ and moreover that $w_r(X_p) \leq w_r(X''_p) \leq n \cdot w(\tilde{v})$. Due to the lower bound we may set $K := \lfloor \frac{\varepsilon \cdot w(\tilde{v})}{n} \rfloor$ and guarantee the desired approximation ratio. The upper bound helps us achieve a polynomial running time. In fact, we can now restrict ourselves to entries $R^{(z)}(\pi, W)$ with $W \leq n \cdot w(\tilde{v})$ in the dynamic program which brings down its running time to $O(p \cdot n^4 \cdot w(\tilde{v})^2)$. By using the scaled weights w' we obtain time $O(pn^6\frac{1}{\varepsilon^2})$. □

We remark that the running time of the above FPTAS could be improved if we could quickly compute lower and upper bounds on $w_{r,p}$. For example if we had constant factor estimations then the running time could be brought down to $O(pn^4\frac{1}{\varepsilon^2})$. I believe that this is possible.

9.5. $(1,p)$-Centroid on Trees

In Section 8.2 we have studied the single follower problem $(1,X_p)$-medianoid, that is, when the leader has placed a plurality of servers. We have seen that there are algorithms for this problem that are significantly faster than the general follower problem on trees. In this section we examine the other side of the coin. We investigate the (multiple) leader problem under the requirement that the follower places only one server. We will see that this problem can be solved efficiently on trees. In our algorithm we exploit the fact that $(1,p)$-centroid can be viewed as a minimax problem, and such problems are easier to attack than r-sum problems in general.

9.5.1. Discrete $(1,p)$-Centroid on a Tree

At first we consider the discrete $(1,p)$-centroid problem. Choose an arbitrary node $s \in V$, and connect s to a new node s_0 of weight 0 by an edge of length ∞. Then choose s_0 as the root of the tree. We can assume w.l.o.g. that the leader does not place at node s_0 of zero weight.

Let $X \subseteq V - s_0$ be a node subset and $W \in \mathbb{N}$. Set X is called *W-bounding* if

1. $w_1(X) \leq W$ and
2. for all $x \in X$ with father x' we have $w_1(X - x + x') > W$.

Lemma 9.5.1 *If $w_{1,p} \leq W$ then $|X| \leq p$ for all W-bounding sets $X \subseteq V$.*

Proof. Assume that $w_{1,p} \leq W$ and let X^* with $|X^*| \leq p$ be an optimal leader placement. Consider an arbitrary W-bounding set X. Map each node from X^* to its closest ancestor in X (this allows us in particular to map a node to itself). We claim that this mapping is surjective which completes the proof.

Assume for contradiction that there is a node $v \in X$ which is not in the image of the mapping, and let u be the father of v. By property 2 there is an $y \in T_u$ such that $w(y \prec X - v + u) > W$. Consider the maximal subtrees T' and T^* that contain the node y but no node from $X - v + u$ and X^*, respectively, as inner nodes. First, y lies in the subtree T_u. Moreover, the closest ancestor of y in X is v (else $w(y \prec X) = w(y \prec X - v + u) > W$). This implies that no inner node of T' can be part of X^*, for, otherwise v would be the image of this node contradicting the premise. Hence T' is a subtree of T^*. Moreover, $w(y \prec X^*) \geq w(y \prec X - v + u) > W$ which is a contradiction. □

We propose the following algorithm: Initialize the node set X which shall be W-bounding at the end to $X \leftarrow \emptyset$. Start at the newly introduced root node s_0 and perform a depth first search traversal of the tree. Whenever the traversal returns from a node v back to its father u perform the test whether there is an $y \in T_v$ such that $w(y \prec X + u) > W$. If this is the case, then add the node $X \leftarrow X + v$.

Lemma 9.5.2 *Given a positive integer W, the algorithm constructs a W-bounding set.*

Proof. To show property 1 assume for contradiction that $w(y \prec X) > W$ for some y at the end of the algorithm. Consider the maximal subtree of T which contains y and does not contain nodes from X as inner nodes. Let $u \in X \cup \{s_0\}$ be the root of this subtree, and $v \notin X$ be its son in the subtree. At the time where the above test was executed for the edge (u, v) the result was $w(y \prec X' + u) \leq W$. Since $X' + u \subseteq X + s_0$ we have also $w(y \prec X) = w(y \prec X + s_0) \leq w(y \prec X' + u) \leq W$ which contradicts the premise.

Property 2 is immediate from the construction of the test, since it can be observed that after the test for a node v has been performed, no more nodes from the subtree T_v are later added to X. □

Theorem 9.5.3 (Discrete $(1, p)$-centroid on a tree) *A discrete $(1, p)$-centroid on a tree can be found in time $O(n^2 \log n \log w(T))$.*

Proof. We perform a binary search to find the smallest weight $W \in [0, w(T)]$ such that there is a W-bounding set X with at most p elements. By 9.5.1 and 9.5.2 the set found by this approach has follower gain $w_{1,r}$ and is therefore an $(1, p)$-centroid.

A straightforward implementation would compute a $(1, X)$-medianoid in the current subtree below each single edge. Using Theorem 8.2.5 on page 148 this yields the proposed running time. □

9.5.2. Absolute $(1, p)$-Centroid on a Tree

In order to solve the problem in the absolute case, we make use of the discretization result stated in Theorem 7.2.2. We point out that from this result one can only derive that the positions of the leader are discretized to positions in $\frac{1}{2}\mathbb{N}$, while the positions of the follower are still unrestricted.

Since the number of critical points in this theorem is possibly exponentially large a direct application to the algorithm stated in the previous section would yield a new instance where the node number and thus the running time of the algorithm would no longer necessarily be polynomially bounded. Hence we propose a modification of the previous algorithm.

We start the algorithm on the unaltered input tree. Whenever in the original algorithm there is a test on an edge (u, v) to be performed, we now essentially have to determine a point on that edge which is W-bounding. By our discretization result it turns out that it is sufficient to restrict the tests to (exponentially many) discrete points on that edge. Since all those sub-edges are threaded on the original edge, the interesting point which is W-bounding can be found by a binary search without actually creating all those points as real nodes. This shows the following result:

Corollary 9.5.4 (Absolute $(1, p)$-centroid on a tree) *An absolute $(1, p)$-centroid on a tree can be found in time $O(n^2 \log n \log w(T) \log D)$ where $D := \max_e d(e)$.*

Proof. The running time follows from similar arguments as above. Notice that the absolute $(1, X)$-medianoid can be computed in $O(n \log n)$ according to Corollary 8.2.6. □

We remark that the polynomial solvability of $(1, p)$-centroid on trees leads to a very simple approximation algorithm for arbitrary $r \geq 1$.

Corollary 9.5.5 *There is an r-approximation algorithm for absolute and discrete (r, p)-centroid on a tree.*

Proof. Let X_p be an (r, p)-centroid and X'_p and $(1, p)$-centroid. Since the incremental gains $w_{i+1}(X_p) - w_i(X_p)$ decrease in general for increasing i. We have that $w_r(X'_p) \leq r\, w_1(X'_p) \leq r\, w_r(X_p)$. Since an $(1, p)$-centroid can be computed in polynomial time the theorem follows. □

9.5.3. Discrete $(1, p)$-Centroid on a Pathwidth-Bounded Graph

In this section we oppose the positive results for the $(1, p)$-centroid on trees with a hardness result for a slightly more complex graph class, namely the class of pathwidth-bounded graphs. A *path decomposition* of a graph (V, E) is a path with node set V' and a mapping $p\colon V \to 2^{V'}$ such that $p(v)$ is a path for all nodes $v \in V$ and $p(v_1) \cap p(v_2) \neq \emptyset$ for all edges $(v_1, v_2) \in E$. The *width*

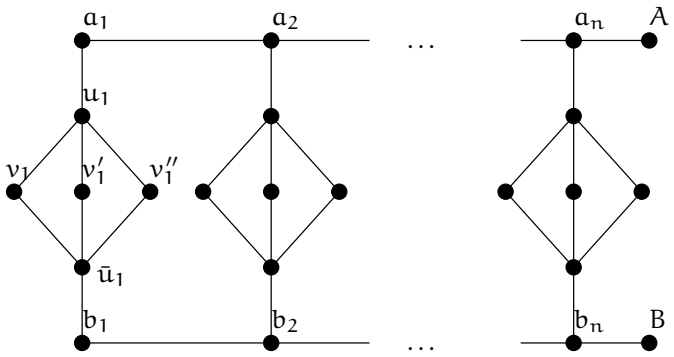

FIG. 9.8.: Discrete $(1, p)$-centroid is NP-hard on a pathwidth bounded graph.

of the decomposition is $\max_{v' \in V'} \left|\{ v \in V \mid p(v) \ni v' \}\right| - 1$. The *pathwidth* of a graph is the minimum width of a path decomposition.

Theorem 9.5.6 (Hardness on pathwidth-bounded graphs) *Determining a discrete or an absolute $(1, p)$-centroid on a pathwidth bounded graph is NP-hard.*

Proof. Let an instance of problem PARTITION be given as in Section 9.1. Construct a graph as follows (confer Figure 9.8): Start with two paths a_1–a_2–…–a_n–A and b_1–b_2–…–b_n–B. For each $i = 1, \ldots, n$, add a connecting path a_i–u_i–v_i–$\bar{u}_i$–b_i and complement it by u_i–v_i'–$\bar{u}_i$ and u_i–v_i''–$\bar{u}_i$ to form a diamond. All edges have unit length except for the edges on the initial a-path and b-path which have length $< \frac{1}{n}$. The node weights are set to $w(u_i) := w(\bar{u}_i) := s_i$ and $w(v_i) := w(v_i') := w(v_i'') = \Omega$ for an $\Omega > S^*$. The weights of the a_i, b_i nodes is set to 1 and finally $w(A) := w(B) := \Omega + 1$.

We claim: For $p := n$ there is a discrete $(1, p)$-centroid of weight $W := \frac{1}{2}S^* + n + \Omega + 1$ if and only if the PARTITION instance is solvable. (The proof for the absolute case is identical.)

"If": Let $S' \subset S$ be a subset with $\sum S' = \frac{1}{2}S^*$. For each $i = 1, \ldots, n$ place the leader at u_i if $s_i \in S'$ and at $\bar{u}_i$ otherwise. The follower places a server at B and claims all b-nodes, plus those nodes $\bar{u}_i$ where $s_i \in S'$ which results in a total gain of W.

"Only if": Consider diamond i. If the leader places no server, the follower could claim more than 3Ω. Hence there must be one server per diamond. If the leader places a server at a v-node, the follower could still claim more than 2Ω. As a consequence, the leader places either at u_i or at $\bar{u}_i$. Let $S' := \{ s_i \mid$ the leader places server at $u_i \}$.

The follower can not claim two or more v-nodes with a single server. Hence it is optimal to place a server on A or B which claims a fixed weight

of $\Omega + 1 + n$, plus the weight $\sum S'$ (if the follower places at B) or $S^* - \sum S'$ (if the follower places a server at A). If $\sum S' \neq \frac{1}{2}S^*$ this is larger than W.

The proof is completed by the observation that the constructed graph has pathwidth 7. □

10. Summary and Further Remarks

10.1. Summary

The second part of this thesis has been devoted to analyzing the complexity and approximability of *multiple* competitive location problems. We have started by examining (r, p)-centroid and (r, X_p)-medianoid on general graphs in Chapter 7. We have seen that the follower problem is NP-hard and that we can give a tight bound of approximability by proving equivalence to the maximum coverage problem. Moreover, it has turned out, that the leader problem is much more difficult both in terms of complexity and of approximability. Specifically, we have shown that (r, p)-centroid is Σ_p^2-complete on general graphs and cannot be approximated within a factor of $n^{1-\varepsilon}$.

These negative results have guided us to examine the problems on simpler graph classes like trees and paths. Here, it has turned out that the follower problem can be efficiently solved on trees by an algorithm of Tamir. We have then concentrated on the case of a single follower who faced a plurality of leader servers. Here, we have proposed an $O(n \log n)$-time algorithm for the more general indirect covering subtree problem which is faster than an existing algorithm of Kim et al. [KLTW96] for the same problem.

Finally, in the previous chapter we have investigated the (r, p)-centroid problem on tree and path graphs. Specifically, we have shown that the *absolute* version is NP-hard even for the simple case of a path, which resolves a long-standing open problem [Hak90, EL96, Ben00]. On the positive side we demonstrate that the *discrete* version of this problem is polynomially solvable on a path by reducing it to a k-sum shortest path problem [PA96] on an acyclic graph. But already for a slightly more complex graph class, namely spiders, the discrete version also becomes intractable. Thus we resolve the complexity status on trees also for the discrete case.

Santos-Peñate et al. [SSD07] approach the absolute (r, X_p)-medianoid

problems by *polynomial* discretization, that is, in the infinite set of points one can identify polynomially many points and solve the discrete problem on this finite set. Since we have shown that on paths the absolute (r, p)-centroid is NP-hard while the discrete is not, we conjecture that such a polynomial discretization is unlikely to work for the absolute (r, p)-centroid problem in general. (Notice that the discretization provided in Theorem 7.2.2 on page 114 is not polynomial.)

We have also shed some light on the approximability of the leader problem on path graphs which was motivated by the hardness on paths and the bad approximation behavior on general graphs. Specifically, we have shown that the leader problem allows an FPTAS on paths.

Finally, we have investigated the $(1, p)$-centroid problem on trees, which has turned out to be solvable in polynomial time for both the absolute and the discrete case. We have opposed this positive result by showing that $(1, p)$-centroid becomes NP-hard on graphs of bounded pathwidth.

Table 10.1 provides an overview on the complexity status of the (r, p)-centroid problem, that is, the problem of optimally placing the leader. For completeness we have added the known results for the corresponding follower problem variants. Notice that in the follower problem we do not distinguish between absolute and discrete model since the complexity is the same in both cases. An overview of the approximability is given in Table 10.2.

In the following section we are going to discuss some extended issues related with competitive location. These topics have not been systematically studied in this thesis and could be subject to further research.

10.2. Incremental Aspects

In all problem settings we have considered so far we were assuming that leader and follower know the exact numbers of facilities in advance. While the decisions on the location of a facility is often irrevocable, the *number* of facilities to be opened is not. In fact, we often encounter *incremental behavior* of providers. This means that a certain number of facilities is opened at the beginning; and later this number is incremented, for example, due to increased budget or demand. Of course, it is usually undesirable to close existing facilities. So the problem amounts to *extending* a given placement by a certain number of additional facilities so as to maximize the demand served.

For example, Chrobak et al. [CKNY08] considered the *incremental me-*

	(r,p)-centroid *(the leader problem)*			(r,X_p)-medianoid
	absolute	discrete		*(the follower problem)*
arb. r arb. p	NP-hard on path *[9.1.1]*	$O(pn^4)$ on path *[9.2.3]*		$O(n)$ on path *[MZH83]*
		NP-hard on spider *[9.3.1]*		$O(rn^2)$ on tree *[MZH83]*
		Σ_2^p-complete on graph *[7.3.3]*		NP-hard on graph *[MZH83]*
$r=1$ arb. p	$O(n^2 \log n \log W \log D)$ on tree *[9.5.4]*	$O(n^2 \log n \log W)$ on tree *[9.5.3]*		$O(n \log n)$ on tree *[8.2.6]*
		NP-hard on pathwidth bounded graph *[9.5.6]*		$O(n^2 \log n + nm)$ on graph *[by enumeration]*
$r=1$ $p=1$	$O(n^4 m^2 \log mn \log W)$ on graph *[HL88]*	$O(n^3)$ on graph *[CM03]* for trees, cf. Part I		

TABLE 10.1.: Complexity of the (r,p)-centroid problem. $W := \sum w(v)$ and $D := \max_{e \in E} c(e)$. The hardness results from the discrete case also apply to the absolute case.

	(r,p)-centroid				(r,X_p)-medianoid
	hardness		approximability		approximability
Graph	Σ_2^p-complete	*[7.3.3]*	lower bound $n^{1-\varepsilon}$	*[7.4.2]*	tight bound $(1-\frac{1}{e})$ *[7.4.1]*
Tree	NP-hard	*[9.1.1]*	r	*[9.5.5]*	1 *[Tam96]*
Path	NP-hard	*[9.1.1]*	FPTAS	*[9.4.4]*	

TABLE 10.2.: Approximability of the (r,p)-centroid and the (r,X_p)-medianoid problem.

dian problem which addresses such a situation. The authors assume in their model that a sequence $F_1 \subseteq F_2 \subseteq \ldots F_n$ is generated by the algorithm such that $|F_k| = k$. The goal is to construct a sequence in which F_k is a good approximation of the optimal k-median solution for any k. The authors gave an incremental algorithm that guarantees a constant relative deviation from the optimum.

Of course, there is no hope to devise an algorithm with similar properties for our (r,p)-centroid problem since the requirement for an incremental algorithm is even stronger than for offline approximation algorithms from which we may not expect good performances.

Therefore, we will consider a relaxed problem: Assume we are given an optimal (r,p)-centroid. Is it possible to extend it to a good leader placement for $(r,p+1)$-centroid? Since it is already difficult to determine an (r,p)-centroid a positive result would not immediately contradict our lower bounds on approximability.

Definition 10.2.1 Let X_p be an (r,p)-centroid and let $X_{p\oplus 1}$ be a $(p+1)$-element leader placement such that $w_r(X_{p\oplus 1})$ is minimum among all $(p+1)$-element placements containing X_p. Then we call $X_{p\oplus 1}$ an incremental $(r, p\oplus 1)$-centroid.

We remark that some caution is needed for this definition since different $(r,p\oplus 1)$-centroids of the same instance need *not* have the same measurement $w_r(\cdot)$. This is because there might be distinct (r,p)-centroids inducing different sets of feasible placements in the preceding definition.

Our question is whether *each* $(r,p\oplus 1)$-centroid constitutes a good approximation for $(r,p+1)$-centroid which would guarantee good extensibility properties of our problem. Unfortunately, our hopes are quickly deceived

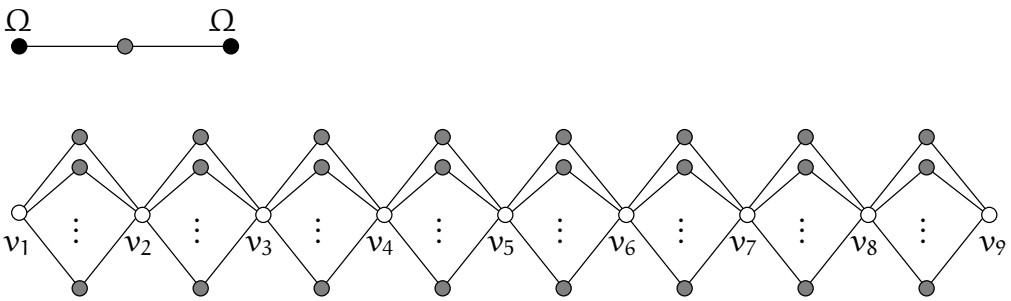

FIG. 10.1.: Examples for bad incremental behavior. All edges and gray nodes have unit weight. White nodes have zero weight. The weight Ω is chosen suitably large. The upper example refers to a $(1, 1 \oplus 1)$-centroid and the lower one to a $(2, 3 \oplus 1)$-centroid. Detailed descriptions can be found in the text.

by the example depicted in the left part of Figure 10.1. Here, the only $(1, 1)$-centroid takes the node in the middle of the path. Thus any extension to a two-element placement grants the follower a gain of Ω. On the other hand, when placing at both end nodes, the follower gets only one user. Hence the deviation of the $(1, 1 \oplus 1)$-centroid from the $(1, 2)$-centroid can become arbitrarily large.

One can, however, observe that the validity example hinges strongly on the fact that the follower places only one server. In fact, we will show in the following theorem that for $r \geq 2$ the deviation can be bounded in terms of the *maximum degree* $\Delta(G)$ of the input graph. Thus we obtain constant factors for degree-bounded graphs and in particular for paths where the case $r = 1$ already behaves arbitrarily bad.

Theorem 10.2.2 *Let $X_{p\oplus 1}$ be any incremental $(r, p \oplus 1)$-centroid where $r \geq 2$. Then*

$$w_r(X_{p\oplus 1}) \leq 2 \cdot \left(1 + \max\left\{1, \frac{\Delta(G)}{r}\right\}\right) \cdot w_{r,p+1}\,.$$

Proof. Let X_p be any (r, p)-centroid and X_{p+1} be any $(r, p+1)$-centroid. Suppose that we remove some point $x \in X_{p+1}$ from X_{p+1} which yields the placement $X'_p := X_{p+1} - x$.

The proof is based on the observation that $w_r(X'_p)$ is an upper bound on $w_r(X_p)$. Thus the deviation of an incremental centroid from the optimum can not be larger than the benefit the follower gets when x is removed x from X_{p+1}. We shall see that if this benefit were very large then it would mainly be caused by the large weight of x itself whereas the residual benefit

would be comparatively small. As a consequence, all points in X_{p+1} would have to be nodes of such a large weight that X_p would inevitably be a subset of X_{p+1}. Then X_p could be extended to X_{p+1}, which is a contradiction.

Consider now some (r, X'_p)-medianoid Y_r. Let $U' := U(Y_r \prec X'_p) - x$ and $B := w(U') - w_r(X_{p+1})$ be the above mentioned *residual benefit*. Clearly, $w_r(X_p) \leq w(X'_p) \leq w_r(X_{p+1}) + B + w(x)$.

It is not hard to see that the benefit B can only be caused by users u switching from the leader to the follower party when removing x. Thus for all such users u the point x must be the closest one in placement X_{p+1}. Therefore, if we define the set U'_x to contain all nodes u from U' such that $d(u, X_{p+1}) = d(u, x)$ then $B \leq w(U'_x)$ holds.

Let $v_1, \ldots, v_k$ be the neighbors of x where $k := \deg(x)$ is the degree of x. We partition U'_x into sets $U'_1, \ldots, U'_k$ where U_i contains all nodes $u \in U'$ such that v_i is met by some shortest path from u to x. Note that the sets U'_i can be made pairwisely disjoint by breaking ties arbitrarily. Suppose that these sets are ordered such that $w(U'_1) \geq \ldots \geq w(U'_k)$ and that the follower places $l := \min\{r, k\}$ facilities at the nodes $v_1, \ldots, v_l$ in the presence of the leader placement X_{p+1}. Then the follower gains at least $\sum_{i=1}^{l} w(U'_i)$, which is surely smaller than the optimal follower gain $w_{r,p+1}$. This implies that $B \leq w(U'_x) \leq \frac{k}{l} w_{r,p+1}$.

We obtain

$$\begin{aligned}
\frac{w_r(X_p)}{w_{r,p+1}} &\leq \frac{w_r(X'_p)}{w_{r,p+1}} \\
&\leq \frac{w_{r,p+1} + B + w(x)}{w_{r,p+1}} \\
&\leq \left(1 + \frac{k}{l}\right) + \frac{w(x)}{w_{r,p+1}} \\
&\leq \left(1 + \max\left\{1, \frac{\Delta(G)}{r}\right\}\right) + \frac{w(x)}{w_{r,p+1}}\,.
\end{aligned}$$

We infer from this equation that the deviation of X_p from X_{p+1} depends on the weight $w(x)$. It is intuitively clear that if all weights of points in X_{p+1} were large enough the optimal leader placement X_p would have to occupy p of them. More formally, let x be the lightest point in X_{p+1} and assume for contradiction that $w(x) > c_0 w_{r,p+1}$ holds where $c_0 := 1 + \max\left\{1, \frac{\Delta(G)}{r}\right\}$. Due to the above inequality $w(X'_p)$ is bounded from above by $w(x) + c_0 w_{r,p+1}$. Moreover, if X_p were not a subset of X_{p+1} the follower would gain at least $2\,w(x)$, which is strictly larger than the gain for X'_p. (Here, we exploit that

$r \geq 2$.) This, however, is a contradiction since X_p is optimum by premise. We conclude that $w(x) \leq c_0 w_{r,p+1}$. Together with the above inequality the claim follows. □

In Figure 10.1 we give an instance which demonstrates that the estimation given in the preceding theorem is not that bad. In this example we demonstrate that there is indeed a class of instances where the deviation is $\Omega(\Delta(G))$. Each graph in this class consist of eight s-diamonds for some $s \in \mathbb{N}$. Clearly, $\Delta(G) = 2s$. It is easy to observe that the $(2,4)$-centroid chooses the nodes v_2, v_4, v_6, v_8 so as to occupy each of the eight diamonds. Hence the follower achieves a gain of at most two. On the other hand, the $(2,3)$-centroid $\{v_2, v_5, v_8\}$ cannot be extended to a node set occupying all diamonds and therefore leaves a gain of at least $\frac{\Delta(G)}{2}$ to the follower. Thus the deviation is at least $\frac{\Delta(G)}{4} = \Omega(\Delta(G))$. Note that the $(2,3)$-centroid (v_2, v_5, v_8) is not unique. This could, however, be achieved by giving the nodes v_2, v_5, v_8 a small weight.

We remark finally that Theorem 10.2.2 yields a factor 4 for path graphs. For this class Wiefel [Wie08] has constructed instances with a deviation of 3 which is tighter than the general lower bound provided in Figure 10.1. Moreover, he has shown that the factor 4 still holds on spiders of unbounded maximum degree.

10.3. Future Research

This short section is devoted to future research related to the second part of this thesis, that is, to *multiple* competitive location. For open problems in the single location case we refer to Section 6.5.

For the *follower* problem, a series of positive results have been developed. We have provided approximation algorithms for general graphs and exact efficient algorithms for special cases such as trees. Therefore, future research could extend such results: One could investigate in how far more complex graph classes (for example, tree-width bounded graphs) admit efficient algorithms. It would also be interesting to investigate the follower problem in more realistic models involving, for example, attraction functions rather than functions only based on distance, or models taking into account purchase price and transportation costs. Certainly, a whole series of further extensions are imaginable here.

The *leader* problem, however, seems extremely difficult even in very simple cases. Future algorithmic oriented research could therefore explore more

sophisticated graph classes. It seems, however, less promising to concentrate on more powerful location models, at least, using the analysis techniques employed in this thesis.

Some specific problems that could be tackled are the following. Study the approximability of (r, p)-centroid on trees or tree-width bounded graphs. One could also investigate the fixed parameter tractability of the leader problem: What happens if r is a parameter and thus not part of the input? Notice that the problem is efficiently solvable for $r = 1$. Finally, we remark that the main purpose of the research in the second part has been to distinguish NP-hard from polynomial-time solvable problems and I believe that the algorithms proposed here can be improved in terms of running time.

Another interesting direction for future research would be to develop incremental *algorithms* that are capable of reporting reasonable approximations. At least for path graphs such algorithms are not ruled out by our lower bounds.

Bibliography

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, Berlin, Heidelberg, New York, 1999. [pp. 18, 153]

[AHT00] S. Alstrup, J. Holm, and M. Thorup. Maintaining center and median in dynamic trees. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory (SWAT 2000)*, volume 1851 of *Lecture Notes in Computer Science*, pages 46–56. Springer-Verlag, 2000. [p. 101]

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, USA, 1974. [p. 145]

[BE95] J. Bhadury and HA Eiselt. Stability of Nash equilibria in locational games. *RAIRO. Recherche opérationnelle*, 29(1):19–33, 1995. [p. 50]

[Ben00] S. Benati. NP-hardness of some competitive location models with probabilistic choice rules. *Studies in Locational Analysis*, 14:211–231, 2000. [pp. 35, 150, 173]

[BFP+73] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973. [p. 77]

[BG01] A. M. Ben-Amram and Z. Galil. Topological lower bounds on algebraic random access machines. *SIAM Journal on Computing*, 31(3):722–761, 2001. [pp. 87, 88]

[BK67] O. Bilde and J. Krarup. Bestemmelse af optimal beliggenhed af produktionssteder. *Research Report, IMSOR, Danmarks tekniske Hojskole*, 1967. [p. 29]

[BK77] O. Bilde and J. Krarup. Sharp lower bounds and efficient algorithms for the simple plant location problem. *Studies in integer programming*, page 79, 1977. [p. 29]

[BL86] M. W. Broin and T. J. Lowe. A dynamic programming algorithm for covering problems with (greedy) totally balanced constraint matrices. *SIAM Journal Algebraic Discrete Methods*, 7(3):348–357, 1986. [p. 132]

[CC93] R. Church and J. Current. Maximal covering tree problems. *Naval Research Logistics*, 40(1):129–142, 1993. [p. 141]

[CKNY08] M. Chrobak, C. Kenyon, J. Noga, and N. E. Young. Incremental medians via online bidding. *Algorithmica*, 50(4):455–478, 2008. [p. 174]

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 1990. [p. 153]

[CM03] C. M. Campos Rodríguez and J. A. Moreno Pérez. Relaxation of the condorcet and simpson conditions in voting location. *European Journal of Operational Research*, 145:673–683, 2003. [pp. 28, 33, 38, 44, 46, 47, 54, 56, 105, 106, 107, 175]

[CM08] C. M. Campos Rodríguez and J. A. Moreno Pérez. Multiple voting location problems. *European Journal of Operational Research*, 191(2):437–453, 2008. [pp. 33, 35, 46, 47, 112]

[CRRS06] S. Chawla, U. Rajan, R. Ravi, and A. Sinha. Min-max payoffs in a two-player location game. *Operations Research Letters*, 35(5):499–507, 2006. [p. 110]

[Dre09] Z. Drezner. *Facility Location: A Survey of Applications and Methods*. Springer, Berlin, 2009. [p. 27]

[Eis92] H. A. Eiselt. Hotelling's duopoly on a tree. *Annals of Operations Research*, 40:195–207, 1992. [pp. 33, 34, 57, 58]

[EL93] H. A. Eiselt and G. Laporte. The existence of equilibria in the 3-facility hotelling model in a tree. *Transportation Science*, 27:39–43, 1993. [p. 33]

[EL96] H. A. Eiselt and G. Laporte. Sequential location problems. *European Journal of Operational Research*, 96:217–231, 1996. [pp. 31, 33, 34, 35, 40, 150, 173]

[ELT93] H. A. Eiselt, G. Laporte, and J.-F. Thisse. Competitive location models: A framework and bibliography. *Transportation Science*, 27(1):44–54, 1993. [pp. 28, 108]

[Eve79] S. Even. *Graph Algorithms*. Pitman, London, 1979. [p. 18]

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability (A guide to the theory of NP-completeness)*. W.H. Freeman and Company, New York, 1979. [pp. 111, 124, 128, 150]

[Gol71] A. J. Goldman. Optimal center location in simple networks. *Transportation Science*, 5:212–221, 1971. [pp. 34, 57, 58, 67, 68, 81]

[GP03] M. D. García Pérez and B. Pelegrín Pelegrín. All Stackelberg location equilibria in the Hotelling's duopoly model on a tree with parametric prices. *Annals of Operations Research*, 122:177–192, 2003. [pp. 33, 34, 38, 49, 57, 58, 82, 83, 87, 103]

[GSG04] P. Godfrey, R. Shipley, and R. Gryz. Maximal vector computation in large data sets. Technical report, CS-2004-06, University of York, 2004. [pp. 105, 106]

[Hak64] S. L. Hakimi. Optimal locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12:450–459, 1964. [p. 26]

[Hak83] S. L. Hakimi. On locating new facilities in a competitive environment. *European Journal of Operational Research*, 12:29–35, 1983. [pp. 27, 34, 40, 41, 117, 129]

[Hak90] S. L. Hakimi. Locations with spatial interactions: Competitive locations and games. In *[MF90]*, pages 439–478. 1990. [pp. 33, 34, 35, 37, 112, 113, 117, 118, 128, 150, 173]

[Han73] G. Y. Handler. Minimax location of a facility in an undirected tree graph. *Transportation Science*, 7:287–293, 1973. [pp. 58, 100]

[Har72] F. Harary. *Graph Theory*. Addison-Wesley, Boston, MA, USA, 1972. [p. 24]

[HD01] H. W. Hamacher and Z. Drezner. *Facility Location: Applications and Theory*. Springer, Berlin, 2001. [p. 27]

[HL88] P. Hansen and M. Labbé. Algorithms for voting and competitive location on a network. *Transportation Science*, 22(4):278–288, 1988. [pp. 33, 44, 56, 103, 175]

[HN98] H. W. Hamacher and S. Nickel. Classification of location models. *Location Science*, 6:229–242, 1998. [p. 28]

[Hoc97] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston, MA, USA, 1997. [p. 126]

[Hot29] H. Hotelling. Stability in competition. *The Economic Journal*, 39(153):41–57, 1929. [p. 26]

[HT81] P. Hansen and J.-F. Thisse. Outcomes of voting and planning: Condorcet, weber and rawls locations. *Journal of Public Economics*, 16(1):1–15, 1981. [pp. 27, 44]

[HTW86] P. Hansen, J. F. Thisse, and R. W. Wendell. Equivalence of solutions to network location problems. *Mathematics of Operations Research*, 11:672–678, 1986. [pp. 33, 34]

[HTW90] P. Hansen, J.-F. Thisse, and R. E. Wendell. Equilibrium analysis for voting and competitive location problems. In *[MF90]*, pages 479–501. 1990. [pp. 28, 33, 37, 42, 43, 46, 57, 81]

[KA75] A. N. C. Kang and D. A. Ault. Some properties of a free centroid of a free tree. *Information Processing Letters*, 4:18–20, 1975. [p. 144]

[KLP75] H. T. Kung, F. Luccio, and F. Preparata. On finding the maxima on a set of vectors. *Journal of the ACM*, 22(4):469–476, 1975. [pp. 56, 105, 106]

[KLTW96] T. U. Kim, T. J. Lowe, A. Tamir, and J. E. Ward. On the location of a tree-shaped facility. *Networks*, 28(3):167–175, 1996. [pp. 35, 131, 133, 134, 141, 142, 145, 148, 173]

[KMN99] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70:39–45, 1999. [pp. 126, 128]

[MF90] P. B. Mirchandani and R. L. Francis. *Discrete Location Theory*. Series in Discrete Mathematics and Optimization. Wiley-Interscience, New York, USA, 1990. [pp. 27, 183, 184]

[MZH83] N. Megiddo, E. Zemel, and S. Hakimi. The maximum coverage location problem. *SIAM Journal on Algebraic and Discrete Methods*, 4(2):253–261, 1983. [pp. 33, 113, 114, 132, 151, 175]

[NSW07] H. Noltemeier, J. Spoerhase, and H.-C. Wirth. Multiple voting location and single voting location on trees. *European Journal of Operational Research*, 181(2):654–667, 2007. [p. 18]

[PA96] A. P. Punnen and Y. P. Aneja. On k-sum optimization. *Operations Research Letters*, 18:233–236, 1996. [pp. 155, 157, 160, 161, 173]

[Pap94] C. M. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, USA, 1994. [pp. 18, 19, 20, 21]

[PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Dover, Mineola, NY, USA, 1998. [pp. 160, 167]

[Sla75] P. J. Slater. Maximin facility location. *Journal of National Bureau of Standards*, 79B:107–115, 1975. [pp. 45, 57]

[Slo78] J. L. Sloss. Stable outcomes in majority voting games. *Public Choice*, 15:19–48, 1978. [pp. 46, 107]

[SSD07] D. R. Santos-Peñate, R. Suárez-Vega, and P. Dorta-González. The leader-follower location model. *Networks and Spatial Economics*, 7:45–61, 2007. [pp. 34, 113, 114, 157, 173]

[SU02] M. Schaefer and C. Umans. Completeness in the Polynomial-time Hierarchy: Part I: A Compendium. *ACM Sigact News, Complexity Theory Column 37*, 33(3):32–49, 2002. [pp. 22, 118, 119]

[SW07] J. Spoerhase and H.-C. Wirth. Security score, plurality solution, and Nash equilibrium in multiple location problems. In *ECCO XX*, ECCO XX, 2007. [p. 18]

[SW08] J. Spoerhase and H.-C. Wirth. Approximating (r, p)-centroid on a path. In *CTW*, CTW, 2008. [p. 18]

[SW09a] J. Spoerhase and H.-C. Wirth. An $O(n (\log n)^2 / \log \log n)$ algorithm for the single maximum coverage location or the $(1, X_p)$-medianoid problem on trees. *Information Processing Letters*, 109(8):391–394, 2009. [p. 18]

[SW09b] J. Spoerhase and H.-C. Wirth. Optimally computing all solutions of Stackelberg with parametric prices and of general monotonous gain functions on a tree. *Journal of Discrete Algorithms*, 7(2):256–266, 2009. [p. 18]

[SW09c] J. Spoerhase and H.-C. Wirth. (r, p)-centroid problems on paths and trees. *Theoretical Computer Science*, 410(47–49):5128–5137, 2009. [p. 18]

[SW10] J. Spoerhase and H.-C. Wirth. Relaxed voting and competitive location under monotonous gain functions on trees. *Discrete Applied Mathematics*, 158:361–373, 2010. [p. 18]

[Tam96] A. Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19:59–64, 1996. [pp. 35, 131, 132, 133, 148, 176]

[vT26] J. H. von Thünen. *Der isolierte Staat*. 1826. [p. 26]

[Web29] A. Weber. *Theory of Location of Industries*. Chicago University Press, Chicago, 1929. [p. 26]

[Wie08] J. Wiefel. Approximationsalgorithmen und inkrementelles Verhalten für kompetitive Lokationsprobleme. Diploma Thesis, University of Würzburg, 2008. [pp. 18, 179]

[WM81] R. E. Wendell and R. D. McKelvey. New perspectives in competitive location theory. *European Journal of Operational Research*, 6(2):174–182, 1981. [pp. 45, 57]

[Yao91] A. C.-C. Yao. Lower bounds for algebraic computation trees with integer inputs. *SIAM Journal on Computing*, 20(4):655–668, 1991. [p. 87]