Diss. ETH No. 14794

Distributed Data & Resources: Models, Tractability, and Complexity

A dissertation submitted to the

SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZÜRICH

for the degree of

Doctor of Technical Sciences

presented by

Konrad August Schlude
Dipl.–Mathematiker, Albert–Ludwigs–Universität Freiburg

born May 24, 1968

citizen of
Germany

accepted on the recommendation of

Prof. Dr. Peter Widmayer, Examiner
Prof. Dr. Roger Wattenhofer, Co-Examiner

2002

Abstract

There are many examples for the fact that the growth of a system is limited if this system relies on important central instances for supply or control. In a growing system, the central instances turn into bottlenecks, thus making the system inefficient. Examples for this phenomenon can be found in computer science, in history, and even in the behavior of dinosaurs. Distributed systems have great potential to bypass such bottlenecks, but due to "friction" issues such as incomplete knowledge, we may not be able to exploit their potential.

In this thesis we show that some of these issues can be handled quite well in weak and realistic models. More specifically, we present and discuss three examples:

1. Distributing Rare Resources: Roman Domination and Win–Win
   Given a graph, servers (resources) should be placed on nodes to service a pair of requests that can occur at nodes. Since resources are rare, the number of used resources should be minimized. What is the computational complexity of this problems, and how much control, communication, or interaction is needed to organize the service?

   We give detailed characterization of the computational complexity of these problems. Especially, we show that these problems are NP–hard; and for Planar Roman Domination, a Polynomial Time Approximation Scheme (PTAS) is presented.

2. Distributed Data Structure
   Given a distributed system of processors/computers/servers with weak assumptions, is it possible to build and maintain a distributed data structure in this system?

   We propose a distributed dictionary that supports insert and search operations and that tolerates arbitrary single server crashes. In contrast to other proposals of distributed fault tolerant search structures, our solution works in an asynchronous and weak environmental setting.

3. Distributed Coordination: Point Formation
   There are n robots in the Euclidean plane that should move to one destination point. What is a minimal set of abilities that the robots have to have in order to complete this task?

   We define the concept of *contraction functions* and show that with this concept, the point formation problem can be solved in an asynchronous model.

The results show that these problems can be solved in weak and realistic settings. We will discuss these problems individually, but we will discuss the connections between them as well. Interestingly, we will find connections between the point formation problem and facility location as well.

Zusammenfassung

Es gibt viele Beispiele dafür, dass das Wachstum eines Systems beschränkt ist, falls dieses System auf zentralen Einheiten für Versorgung oder Kontrolle basiert. Wenn das System wächst, werden die zentralen Einheiten zu Engpässen, die das System ineffizient machen. Beispiele für dieses Phänomen finden sich in der Informatik, in der Geschichte und auch im Verhalten von Dinosauriern. Verteilte Systeme haben die Möglichkeit, solche Engpässe zu umgehen, aber auf Grund von "Reibungsverlusten", wie zum Beispiel unvollständigem Wissen, sind wir eventuell nicht in der Lage, diese Möglichkeiten auszuschöpfen.

Das Ziel dieser Arbeit ist es, zu zeigen, dass mit solchen "Reibungsverlusten" recht gut umgegangen werden kann. Dies soll an drei Beispielen gezeigt werden:

1. Verteilung rarer Ressourcen: Roman Domination und Win–Win
   Gegeben sei ein Graph, Server (Ressourcen) sollen auf Knoten des Graphs plaziert werden, um jedes Paar von Anfragen bedienen zu können, die an Knoten auftreten können. Da die Ressourcen rar sind, soll die Anzahl der verwendeten Ressourcen minimiert werden. Wie gross ist die Berechnungskomplexität, und wie viel Kontrolle, Kommunikation oder Interaktion wird benötigt, um die Bedienung der Anfrage zu organisieren?

   Wir präsentieren eine detailierte Charakterisierung der Berechnungskomplexität dieser Probleme. Insbesondere zeigen wir, dass diese Probleme NP-hart sind, und für Planares Roman Domination präsentieren wir ein Polynomielles Approximationsschema (PTAS).

2. Verteilte Datenstruktur
   Gegeben sei ein Computernetzwerk mit schwachen Modellannahmen. Ist es möglich, in einem solchen Netzwerk eine verteilte Datenstruktur zu unterhalten.

   Wir präsentieren ein verteiltes Wörterbuch, das Einfüge– und Suchoperationen ermöglicht und den Ausfall eines beliebigen Servers toleriert. Im Gegensatz zu anderen vorgeschlagenen verteilten fehlertoleranten Suchstrukturen arbeitet unser Lösungsvorschlag in einem stark asynchronen Modell.

3. Verteilte Koordination: Punkt-Formation
   Es seien n Roboter in der euklidischen Ebene verteilt. Die Roboter sollen sich an einem Punkt treffen. Was ist ein Mindestmass an Fähigkeiten, die die Roboter haben müssen, um diese Aufgabe zu bewältigen?

   Wir definieren das Konzept der *Zusammenziehfunktionen* und zeigen, dass das Punkt-Formations Problem mit diesem Konzept in einem asynchronen Modell gelöst werden kann.

Die Resultate zeigen, dass diese Probleme auch in realistischen Modellen gelöst werden können. Die Beispiele werden separat diskutiert, es werden aber auch die Zusammenhänge zwischen ihnen erklärt. Interessanterweise werden wir auch Zusammenhänge zwischen dem Punkt-Formations Problem und der Plazierungstheorie (facility location) finden.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die positiv auf diese Arbeit eingewirkt haben.

Mein erster Dank gilt Professor Widmayer für die Motivation und die Begleitung dieser Arbeit. Professor Roger Wattenhofer danke ich für seine Bereitschaft, sich als Korreferent mit meiner Arbeit zu beschäftigen.

Auch vielen anderen Mitgliedern der Gruppe Widmayer bin ich zu Dank verpflichtet. Ohne UNIX–, Latex– und xfig–Tricks und Hilfen meiner Arbeitskollegen wäre es wohl schwieriger für mich gewesen. Für die nette Arbeitsatmosphäre bedanke ich mich insbesondere bei meinen Bürokollegen Stephan Eidenbenz und Aris Pagourtzis; Stephan Eidenbenz hat mich ferner sehr durch Aufmunterung und Korrekturlesen unterstützt. Ein Teil dieser Arbeit entstand in Kooperation. Neben den schon genannten möchte ich mich bei den folgenden Mitautoren für die gute Zusammenarbeit bedanken: Martin Gantenbein, Paolo Penna, Eljas Soisalon–Soininen, Kathleen Steinhöfel und David Taylor. Ein weiterer Teil entstand während eines dreiwöchigen Forschungsaufenthalt in Ottawa. Für die Einladung dazu bedanke ich mich bei den Professoren Paola Flocchini und Nicola Santoro. Weitere mich begleitende oder unterstützende Weggefährten waren Christoph Stamm, Birgitta Weber, Luzi Anderegg, Jörg Derungs, Daniele Degiorgi, Mark Cieliebak, Conrad Pomm, Zsusanna Liptak, Gabor Szabo und Marc Nunkesser.

Besonders bedanken möchte mich mich bei Professor Ernst Specker. Seine Hilfe und Motivation waren für das Gelingen dieser Arbeit von grosser Bedeutung.

Im Rahmen dieser Arbeit habe ich viel über Verteilte Systeme, Berechnungskomplexität und andere Gebiete gelernt; für mich besonders wichtig ist jedoch die trivial anmutende Aussage: 17 ist prim!

Meine Familie war mir schon immer eine wichtige Stütze in meinem Leben, dankbar bin ich daher meinen Eltern, meiner Frau Karin und unserem Sohn Simon.

Contents

Chapter 1

Introduction

Earth's gravity limits the height of animals [LS00a, Lav01]. The taller an animal is, the higher the blood pressure of the animal must be in order to pump blood through the whole body. For instance, reptiles (e.g., crocodiles) have blood pressures between 35 and 75 mmHg[1], the taller humans need pressures between 100 and 150 mmHg to supply the body with fresh blood, and giraffes need pressures around 300 mmHg [Lav01]. Strong muscles are needed to produce high pressure, and in order to resist that pressure, the heart and the veins have to have thick walls which consume a lot of energy. Thus, from an energetic point of view, tallness is not attractive. [LS00a] illustrates this with a comparison between sauropod dinosaurs and whales. If an upright neck posture of the dinosaur is assumed, then the sauropod dinosaurs' blood pressure would have to be around 700 mmHg and the left heart ventricles would weigh 15 times more than those of similarly sized whales. The large ventricles would consume about 62% of the dinosaurs' resting metabolic rate; for humans, this rate is at about 10%. For [LS00a], this contradicts the upright neck postures assumption. Thus, the fact that the blood of an organism is pumped by a single, central pump (i.e., the heart) limits the potential variety of shapes of animals.

In the second century BC, there were several wars between Rome and the (Greek) Hellenistic kingdoms Macedonia and Seleucia. In a battle, the Greeks organized their troops as a *phalanx*, i.e., as one long line, 8 to 10 men deep. A moving phalanx was hard to control, especially in a hilly region. When trying to advance quickly over rough ground, the phalanx line tended to break up. This made the phalanx prone to flanking and vulnerable. The Romans used a much more flexible arrangement: the Roman legions had a hierarchical structure and acted with less central control. In the battles, the Roman armies consistently broke up the phalanx and defeated the Greek troops. Within a few decades, the Romans conquered the Hellenistic kingdoms that had been very powerful before. For the ancient Greek historian Polybius [Pol80], this proved the superiority of the legion over the phalanx.

Their military superiority enabled the Romans to conquer a huge empire. For centuries, the empire grew and grew. Emperor Trajan (98–117 AD) expanded the Roman empire to its greatest geographic size, but then, the long decline of the empire began (see for example [Gib95]). The central government was not powerful enough to control the huge area. The empire had many enemies, and in order to organize the defense, the emperors had to spend a lot of time in the provinces, far away from Rome. In the third century AD, there were several revolts and civil wars. Especially, there were separate empires of Gaul (261–274 AD) and Palmyra (260–272 AD), seizing large areas of the

[1] millimeters (mm) of mercury (Hg)

empire. To stop the decline, emperor Diocletian (284–305 AD) split the empire and imperial responsibility: Diocletian established a *co–emperor* (286 AD) and created *tetrarchy* consisting of two *co–Augusti* and two *co–Caesars* (293 AD). Although there were four emperors, the empire was still considered to be a single unit. After Diocletian's abdication in 305 AD, there were civil wars between co–emperors. Emperor Constantine (306–337 AD) won these wars and became sole emperor. Constantine reorganized the military forces and placed the armies in a way such that the whole empire could be protected. He applied a very interesting strategy in placing his armies that we will discuss later. When Constantine died, the empire was split again, and there were civil wars between the co–emperors, which led – after several decades of instability – to the definite division of the empire into two separated parts in 395 AD.

These examples illustrate the fact that the growth of a system is limited if this system relies on important central instances for supply or control. If the system grows, the central instances become bottlenecks, and therefore, the system becomes ineffective. However, such bottlenecks can be bypassed in many cases. For instance, the height limit mentioned above can be bypassed in pipelines as follows: Instead of one gigantic pump there are many small pumps to transport liquid or gas.

There are more modern examples, too: [Par96] argues that teams of cooperative robots have capabilities of very high sophistication level such that in reality no robot can be designed with these capabilities; routing in the internet is done without central control and can be modeled as actions of independent selfish agents that try to maximize their benefit. It is obvious that a system without central control does not always give a global optimal solution, and one can ask for the "price of anarchy" compared to the price of an optimal solution [KP99]. However, in the case of internet routing, it is practically impossible to collect all the necessary information, compute an optimal solution, and spread this solution. Therefore, it is much more practical to have a solution generated by independent agents.

These examples show that distributed systems occur in numerous application contexts, which motivates our theoretical study of such systems. We call a system distributed, if there are several interacting entities, e.g., agents, computers, peers, etc., without any or with very limited central control. Distributed systems have the potential to overcome issues encountered by central systems, but they also present challenges of a new kind. In particular, there usually is some "friction" in distributed system, i.e., the power of a system is not linear in its size. We illustrate this with architectures for parallel computers (see for example [Smi98, Sto01]). A parallel computer consists of several synchronized processors, and two processors can communicate directly, if there is a communication channel between them. The most powerful parallel architecture is the one in which each processor can communicate with each other processor directly, i.e., the underlying network topology is a complete graph. This architecture is a nice tool for theoretical studies, but it seems to be impossible to build such a computer for an arbitrary number of processors, because it is hard to provide all the possibly needed (fast) communication channels. A different network topology is the *Hypercube*. A k–dimensional hypercube consists of 2^k processors and every processor has communication channels to k other processors. The diameter is k, i.e., there are nodes with distance k (but not with bigger distance). Therefore, communication between two processors takes more time and the k–dimensional hypercube is clearly less powerful than 2^k processors arranged in a complete graph. However, since the number of communication channels per processor grows only logarithmically in the number of processors, the hypercube can be used to build bigger computers. For instance, if we allow only four communication channels per processor, it is possible to arrange five processors

in a complete graph or to build a 4–dimensional hypercube with 16 processors. Still, the number of communication channels grows with the size of the system, and therefore, there is a size limit. *Cube Connected Cycles (CCC)* and *Butterflies* are alternative network topologies. Both topologies have diameters logarithmic in the number of processors, but the number of communication channels per processor is bounded. These topologies can be used to build arbitrarily large computers, but they are less powerful than the hypercube. Thus, the power of a distributed system depends on its size as well as on its level of connectedness.

In a distributed system of processors with asynchronous message passing, there is "friction", too. [Wat98] shows that in every possible implementation of distributed counters, there is at least one bottleneck processor that has to do a lot of work. To be more precise, in a distributed counter consisting of n processors, it is impossible that for every processor the workload is not bigger than $\frac{1}{n}$ of the whole workload. This is a performance result, but there are also impossibility results: The consensus problem is the problem that processors should agree on one value. Consensus is impossible, if processors can fail [FLP85].

Thus, while distributed systems have great potential to bypass limits encountered by centrally controlled systems, numerous issues arise in practice that seem to limit the applicability of distributed systems. The conclusion is the following. In order to study the aspects of distributed systems that are relevant in practice, one has to take into account "friction" issues (such as incomplete knowledge or message delays). One approach when studying distributed systems consists of modifying the model by making unrealistic assumptions such that the problem can be solved easily. An alternative approach, which we propose in this thesis, is the following: Given a distributed system with weak assumptions, what can be done in this system? In particular, we consider the following three topics.

1. Distributing Rare Resources: Roman Domination and Win–Win
   Given a graph, servers (resources) should be placed on nodes to service a pair of requests that can occur at nodes. Since resources are rare, the number of used resources should be minimized. What is the computational complexity of this problems, and how much control, communication, or interaction is needed to organize the service?

2. Distributed Data Structure
   Given a distributed system of processors/computers/servers with weak assumptions, is it possible to build and maintain a distributed data structure in this system?

3. Distributed Coordination: Point Formation
   There are n processors in the Euclidean plane that should move to one destination point. What is a minimal set of abilities that the robots have to have in order to complete this task?

We will look at these topics individually, but we will show the connections between them as well. In the following, we describe the topics in more detail.

1.1 Distribution of Rare Resources

Distributing resources is a challenging problem: Where should resources (e.g., schools, ambulances) be placed such that a desired property is fulfilled and the number of used resources is minimized.

Interestingly, a very old question has also triggered new research on the topic [AF95, RR00, Ste99]:

> Roman Domination : Where should the armies of the Roman empire be placed so that a smallest number of armies can protect the whole empire (see Figure 1.1)?

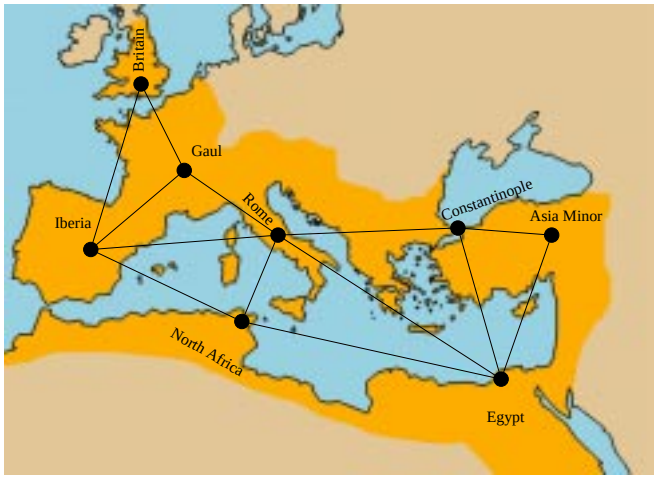

Figure 1.1: The Roman empire around 300 AD

The assumption is that an area can be protected either by one army located inside the area, or by an army in a neighbor area that comes over for the defense in case of an attack. In the latter case it is required that a second army remains in the neighbor area, so that it can quickly confront a second attack. A reason for the historical 1-2 requirement (one army here or two at a neighbor) is that we want to be able to service two requests in one time unit (provided that no two requests can come from the same point at the same time).

The question goes back to the emperor Constantine (306–337 AD). In the third century AD, the Roman empire had lost much of its economic and therefore military power; furthermore, the imperial government had had problems to control the huge empire and to organize defense against attacks from outside. Constantine's goal was to place the weakened remaining military power such that this power would still suffice to defend the empire. To reach this goal, Constantine proposed the 1-2 requirement mentioned above. One important aspect of this requirement is that the decision which army should repulse an attack can be made locally, i.e., only armies from the attacked areas and their neighbor areas are influenced, and armies from other areas do not have to be rearranged (shifted to other areas).

In chapter 2, we formalize this as a variation of dominating sets: When a request can occur at any node in a graph and requires a server at that node, a minimum dominating set represents a minimum set of servers that serve an arbitrary single request by moving a server along at most one edge. We study domination problems for two requests. For the problem of placing a minimum number of servers such that two requests at different nodes can be served with two different servers (called win-win), we present a logarithmic approximation algorithm based on a greedy strategy, and we prove that nothing better is possible by proposing a gap–preserving reduction from Dominating Set. We show that the same results hold for Roman domination, the well studied problem variant that asks for each vertex to either possess its own server or to have a neighbor with two servers.

Still the same is true if each idle server can move along one edge while the first of both requests is being served. For planar graphs, we propose a Polynomial Time Approximation Scheme (PTAS) for Roman domination (and show that nothing better exists), and we get a constant approximation for win-win. The content of this chapter is based on joint work together with Aris Pagourtzis, Paolo Penna, Kathleen Steinhöfel, David Scot Taylor, and Peter Widmayer, published as [PPS+01, PPS+02].

1.2 Distributed Data Structure

The amount of digital information grows at a breathtaking pace, and constantly improving networking technology makes distributed computing power readily available. Modern databases make use of today's technology and develop into increasingly global information systems. The efficient storage and retrieval of data becomes a critical issue, and as a consequence, distributed data structures have gained considerable attention [BDSW99, KW94, Krö97, Lam96, LN96, LNS93, Lom96, PLN00, SW98, VBW98]. Without these distributed data structures, efficient search for the desired data is next to impossible.

It has been shown to what extent classical central concepts carry over to the distributed setting: [LNS93] presents a distributed hash file LH*, whereas [KW94] proposes a scalable data structure based on random binary leaf search trees. One drawback of the LH* approach is that the address of a server corresponds to its position in the hash file One drawback of the LH* approach is the following: To perform an operation, a client uses a hash function to compute an index i and sends a message to the server with index i, i.e., the address of a server corresponds with its position in the hash file. In a globally distributed data base, this requirement is unrealistic. A different approach is *CHORD* [SMK+01]. The servers are arranged in a ring, shortcut links inside the ring help to avoid long search paths. CHORD uses hashing, too. The disadvantage of CHORD is that the information the servers have to store grows logarithmically in the size of the structure. For many data warehousing and multidimensional database applications, the important access primitives include a variety of similarity search operations, such as nearest neighbor queries or the enumeration of the data in a close neighborhood of the query. Since these operations cannot be supported with hashing, this leaves distributed search trees as the only viable alternative.

In their seminal paper [LNS93], Litwin et al. coined the term *Scalable Distributed Data Structure (SDDS)* for structures that satisfy the following requirements: a file expands to new servers gracefully; there is no master site that must be accessed for virtually each operation; the file access and maintenance primitives never require atomic updates to multiple workstations.

A distributed system offers the chance to remain operational even if an individual computer fails. For a large distributed system, such a failure must indeed be expected from time to time. Therefore, distributed data structures have been proposed that tolerate limited hardware failures and still support access to all data at all times — they are called *highly available* data structures [LMRS99, LN96, LS00b]. This failure resilience is achieved by means of data replication [LN96], or by applying the technique of parity records and buckets [LMRS99, LS00b]. In these proposals an essential feature is that a server can detect rapidly whether some other server is operational or crashed. This is certainly a reasonable assumption for local networks, but it is unrealistic for globally distributed databases.

In chapter 3, we propose a distributed dictionary that allows insert and search opera-

tions and that tolerates arbitrary single server crashes. The distinguishing feature of our approach is that high availability can be achieved even if no server can ever detect whether some other server is operational or has crashed. This takes asynchronicity seriously, but makes the design of a data structure quite complicated and fundamentally different from the situation where a server crash can be detected [Lam96, LMRS99, LN96, LS00b]. For instance, locks are not allowed to synchronize servers, since the server that has to unlock another server could have a crash failure, thereby letting the other wait — clearly an unacceptable situation. The content of this chapter is based on joint work together with Eljas Soisalon–Soininen and Peter Widmayer, published as [SSSW01, SSSW02].

1.3 Distributed Coordination

Point formation is the problem that n mobile robots in the plane should meet in one destination point. In accordance with our approach of defining weak but realistic models, this destination point should be independent of coordinate systems, and it should not rely on an ordering of the robots, especially, there is no leader among the robots. The question is the following: Which algorithm can the robots use to complete the task?

This problem comes from the field of robotics. Teams of cooperative robots are assumed to have capabilities of very high sophistication level ([Par96]) and the goal is to coordinate the robots in a distributed way. AI researchers are interested in teams of robots as well, because they want to get a better understanding of intelligence [PS98].

Besides practical studies, the point formation problem has been investigated in a more theoretical way [SY93, SS96, SY99, Pre01, FPSW99, FPSW00, Pre00, GP01]. The solutions presented in these publications are complex, and some of them are hard to understand or verify. We present a new approach whose most distinguishing feature is its simplicity.

Chapter 4 gives the definition of contraction functions and shows that with this concept, the point formation problem can be solved. Furthermore, we show a strong connection between contraction functions and the Weber problem: Given n points in the plane, which is the point that has the smallest sum of distances to the given points. The Weber problem plays an important role in the field of facility location. In chapter 5, we extend the point formation problem with the restriction that the robots have to meet on a given line. We give several results for the extended problem and discuss differences to chapter 4. Chapter 6 shows that the concept of contraction functions is useful for generalized Weber problems as well.

1.4 Mathematical Concepts

This section presents some mathematical concepts that will be used in the thesis. The first such concept is the concept of multi set. We will use multi sets in distributed coordination and in distributing resources. We give a definition for *multi sets of points*, multi sets of other objects can be defined in a similar way.

Definition 1.1 *A mapping* $X : \mathbb{R}^2 \to \mathbb{N}$ *is called a* multi set *of points. For a point* $p \in \mathbb{R}^2$*, we define the corresponding multi set*

$$\{p\} := \mathbb{1}_p \quad \textit{where} \quad \mathbb{1}_p(q) = \begin{cases} 0 & \textit{if } q \neq p \\ 1 & \textit{if } q = p \end{cases}$$

Throughout this thesis, we only consider finite multi sets, i.e., the cardinality

$$|X| := \sum_{p \in \mathbb{R}^2} X(p)$$

is finite.

Let X, X' be two multi sets, we define the following operations.

- *Union $\uplus$:* $(X \uplus X')(p) := X(p) + X'(p)$
- *Intersection $\cap$:* $(X \cap X')(p) := \min\{X(p), X'(p)\}$
- *Setminus $\setminus$:* $(X \setminus X')(p) := \max\{X(p) - X'(p), 0\}$

X is called a set, *iff $X(p) \leq 1$. Given a multi set X,* uniq(X) *denotes the set resulting by removing multiplicities, i.e., $p \in$ uniq(X), iff $X(p) \geq 1$.*

For a multi set X of points, we define the convex hull $CH(X)$ as the smallest convex set $K \subset \mathbb{R}^2$ with uniq$(X) \subset K$.

Sometimes, we will use the more familiar notation $X = \{p_1, \ldots, p_n\}$, where it is possible that $p_i = p_j$ for $i \neq j$. We have to mention that the enumeration in this notation is not a part of the structure of the multi set X, the enumeration is arbitrary and only a help to handle multi sets.

In the chapters about distributed coordination, multi sets of collinear points will play an important role. For such a multi set, it will be crucial, whether it has a median or not. Therefore, we give a formal definition of median.

Definition 1.2 *Let $X = \{p_1, \ldots, p_n\}$ be a multi set of collinear points, i.e., there is a straight line l such that $p_i \in l$ for all i. A point $q \in l$ is called a* median *of X, iff*

$$\left| \sum_{\substack{i=1 \\ p_i \neq q}}^{n} \frac{p_i - q}{|p_i - q|} \right| < X(q)$$

A basic lemma about the existence of medians will be used frequently and is therefore shown here.

Lemma 1.3 *If a multi set X of collinear points has a median q, then it is unique and $q \in X$, i.e., $X(q) \geq 1$. If $n := |X|$ is even, then $X(q) \geq 2$.*

Furthermore, if n is odd, then the median exists.

Proof: Due to definition, for a median q, we obtain that $0 < X(q)$, i.e., $X(q) \geq 1$. If n is even, the sum of $n - 1$ unit vectors $\pm v$ cannot be zero, therefore $X(q) \geq 2$.

In order to show uniqueness, we choose $q', q'' \in X$ such that q', q'' are on different sides of q and there are no points in between q', q'' than q. There are integers $k', k'' \geq 0$ with $k' + k'' + X(q) = n$ and

$$\sum_{\substack{i=1 \\ p_i \neq q}}^{n} \frac{p_i - q}{|p_i - q|} = k' \frac{q' - q}{|q' - q|} + k'' \frac{q'' - q}{|q'' - q|}$$

Since $|k' - k''| < X(q)$ we obtain $\pm(k' - k'') < X(q)$.

$$\begin{aligned}\sum_{\substack{i=1\\p_i \neq q'}}^{n} \frac{p_i - q'}{|p_i - q'|} &= (k' - X(q'))\frac{q' - q}{|q' - q|} + (k'' + X(q))\frac{q'' - q}{|q'' - q|} \\ &= (X(q') + \underbrace{X(q) + k'' - k'}_{>0})\frac{q - q'}{|q - q'|}\end{aligned}$$

This implies, that q' is not a median.

Let n be odd. If X is a set, then it has a median $q \in X$. With the following operations, every multi set of collinear points can be constructed from a set, and the median remains constant. Assume that X is a multi set with median q. Let $p_i, p_j \in X$ be two points with the property $\frac{p_i - q}{|p_i - q|} = \frac{p_j - q}{|p_j - q|}$, q is the median of the multi set $X' := (X \setminus \{p_j\}) \uplus \{p_i\}$. Furthermore, for a point $p_i \in X$ with $p_i \neq q$, q is the median of the multi set $X'' := (X \setminus \{p_i\}) \uplus \{q\}$. □

For our robot model we need to define distance preserving functions.

Definition 1.4 *A function $o : \mathbb{R}^2 \to \mathbb{R}^2$ is called* isometric, if for all pair of points $p, q \in \mathbb{R}^2$, $|o(p) - o(q)| = |p - q|$ holds.

Note that, there are two different definitions of isometry. Functions that are isometric according to the above definitions are not forced to be angle preserving, i.e., an oriented angle α can be mapped to $-\alpha$. In Riemannian geometry, this is not possible [O'N83].

In this thesis, we use several isometric functions:

- Rotation: For an angle $\varphi \in [0^o, 360^o]$ and for a point $p \in \mathbb{R}^2$, $R_{\varphi,p}$ denotes a rotation by φ about p.
- Mirroring or Reflection: Given a straight line l, M_l is the mirroring about l.
- Translation: Let $v \in \mathbb{R}^2$, the translation T_v is defined as $T_v(z) := z + v$.

Rotation and translation are angle preserving, while mirroring maps an angle α to $-\alpha$. The concatenation of two isometric functions is an isometric function, too. This implies that isometries under the operation of function composition form a group.

We used the distance function to define isometric functions. The distance function is not differentiable, but it fulfills a weaker notion of differentiability.

Definition 1.5 *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a function, and let $v \in \mathbb{R}^2$ be a vector. If the limes exists, we call*

$$f_v(q) := \lim_{t \searrow 0} \frac{f(q + tv) - f(q)}{t}$$

the Directional Derivative *in direction v.*

Example: Let $p \in \mathbb{R}^2$ be fixed. The function $f(q) := \mathrm{dist}(q, p)$ is differentiable in $q \neq p$ with gradient $\frac{q-p}{|q-p|}$. The function is not differentiable in p, but for every vector v, the directional derivative exists and $f_v(p) = |v|$.

In chapter 2, we use the notion of *treewidth* to derive a Polynomial Time Approximation Scheme (PTAS) for an NP–hard optimization problem. We recall the definition of treewidth from [ABFN00].

Definition 1.6 *Let $G = (V, E)$ be a graph. A* tree decomposition *of G is a pair $\langle \{X_i, | i \in I\}, T\rangle$, where each X_i is a subset of V, called a bag, and T is a rooted tree with the elements of I as nodes. The following three properties should hold:*

1. $\bigcup_{i \in I} X_i = V$;
2. *for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$;*
3. *for all $i, j, k \in I$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.*

The width *of $\langle \{X_i, | i \in I\}, T\rangle$ equals $max\{|X_i| | i \in I\} - 1$. The* treewidth *of G is the minimum k such that G has a tree decomposition of width k.*

The treewidth of a graph is always bigger than 0, except for the case that $E = \emptyset$. On the other hand, the size of a bag is bounded by the number of nodes in the graph. Therefore, the treewidth of a graph is less than the number of nodes.
Example: Consider the graph G in Figure 1.2. We define the bags $X_1 := \{v_1, v_3\}$ and $X_2 := \{v_2, v_3\}$ and a tree T with 1 as the root and 2 as a child. The pair $\langle \{X_1, X_2\}, T\rangle$

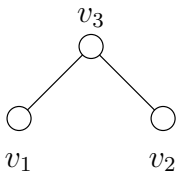

Figure 1.2: Graph G with treewidth 1.

is a tree decomposition. Since the size of both bags is 2, G has a treewidth of 1.

G is a tree. It is easy to show that every tree has treewidth 1. The situation changes, if we add the edge $\{v_1, v_2\}$ to G, i.e., we deal with the complete graph K_3. There is no tree T', such that $\langle \{X_1, X_2, \{v_1, v_2\}\}, T'\rangle$ is a tree decomposition of K_3 (Contradiction to property 3). Therefore, the treewidth of K_3 is 2. This result can be extended, the treewidth of the complete graph K_{n+1} is n.

Chapter 2

Distributing Rare Resources: Roman Domination and Win–Win

2.1 Introduction

In this chapter, we reverse the problem from the previous chapters: Instead of bringing robots together at one point, we spread resources over a graph. The goal is that every node of the graph has sufficient "access" to the resources; and since the resources are expensive or rare, we want to reach this goal with the smallest possible amount of resources.

To be more precise, we study a generalization of the *dominating set problem* [GJ79]. We are given a graph, and at every node of this graph a request can appear. We want to service such requests. To do so, we place servers at nodes. The request at a node v is serviced, if there is a server on v, or if a server in its neighborhood is moved to v. Clearly, if we want to be able to service one request, then the multi set of server locations must contain a dominating set of nodes. However, there are applications in which we want to ensure that more than one request can be serviced. In this chapter, we study the case of two requests. Imagine, e.g., that two requests occur simultaneously and a server can satisfy only one at a time. The study of such dominating set problems is motivated by their applications in facility location (minimizing the number of facilities, subject to every demand being close enough to some facility), file sharing in distributed systems [NR95], game theory [dJ62], etc.

In particular, we consider the case in which there are two requests we want to service and no two requests appear at the same node. Moreover, a server that is used to service the first request cannot be used to service another request. A solution to our problem for a given graph is a set of servers at nodes; since all servers are identical, a multi set of nodes (where the multiplicity of a node is the number of servers at that node) represents a server placement. One possible solution is the Roman Dominating set [Dre00, Ste99], where for every node, there is one server on this node or there is a neighbor node with two servers.

Two factors we will consider are: (i) whether the two requests are known before the first one must be serviced (Offline), or the first one must be serviced before the second one is known (Online), and (ii) whether servers must stay in place unless they service a request (Static), or we allow for a rearrangement (Dynamic): as one server services the first request, all other servers are allowed to move to a neighbor node. The goal of the move is to guarantee that any second request can be handled, too, in the Online case (that is, the resulting server placement is a dominating set if we ignore the first requesting node and its server). The Online Static Win-Win version has been discussed earlier

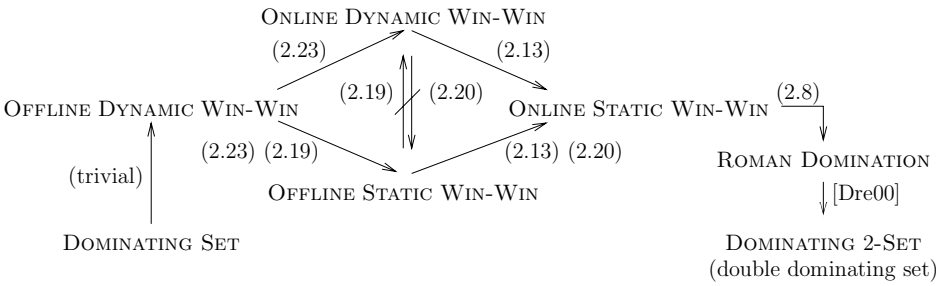

Figure 2.1: Relationships between the problems: arrows represent '$\prec$' and they are numbered according to the corresponding theorem.

[Och96] and called *Win-Win* there. (Unlike in Roman Domination, in this case we only require to be able to win against any two consecutive attacks.) Since our problems also deal with two consecutive requests, we adopt the name terminology and we denote the four problem variants as Online Static Win-Win, Online Dynamic Win-Win, Offline Static Win-Win, and Offline Dynamic Win-Win.

2.1.1 Our (and Previous) Results

In this chapter we investigate the relationships between the above problems (including Roman Domination), as well as the complexity of computing exact and approximate solutions. In particular, we consider the following questions:

1. Given a multi set S, is S a feasible solution to (one of) the above problem variants? Is there a combinatorial characterization for those S?

2. Let varA Win-Win and varB Win-Win denote any two problem variants. If S is a solution for varB Win-Win, does this imply that S is also a solution to varA Win-Win?

3. A positive answer to the above question implies that $\mathsf{opt}_{\mathrm{A}}(G) \leq \mathsf{opt}_{\mathrm{B}}(G)$, where $\mathsf{opt}_{\mathrm{A}}$ and $\mathsf{opt}_{\mathrm{B}}$ denote the minimum size multi set solving the two variants, resp. Is there a graph for which the inequality is strict?

Let varA Win-Win$\preceq$ varB Win-Win denote the fact that Question 2 has a positive answer, and let varA Win-Win$\prec$ varB Win-Win denote the fact that Question 3 does too. It turns out that the problems we look at form the partial order in Figure 2.1.[1] Noticeably, this relationship also holds when we restrict ourselves to planar graphs.

As for Question 1, for two out of the four win-win problems we provide a characterization of those multi sets corresponding to each problem. For the Dynamic Win-Win, we prove the NP-hardness of the rearrangement step after the first request. This result seems to denote that such a characterization for this problem version does not exist, or at least is different from those given for the other two problems (those can be checked in polynomial time).

This leads us to complexity and (non-) approximability issues. Intuitively, the $\preceq$ relationship may have some consequences on the (non-) approximability of those problems. Indeed, the order in Figure 2.1, combined with the fact that "doubling" a dominating set (the Dominating 2-Set problem in Figure 2.1) yields a feasible solution for *all* of the

[1]Figure 2.1 contains a new problem (Dominating 2-Set) which we introduce to prove some of our results.

Problem Version	General Graphs	Planar Graphs
Roman Domination	$(2+2\ln n)$-APX, not $c\log n$-APX (NP-hard [Dre00])	PTAS, in P for r-outerplanar (NP-hard [Hed00]) (in P for trees & $(r \times n)$-grids [Dre00])
Online Sta. Win-Win, Online Dyn. Win-Win, Offline Sta. Win-Win	$(2+2\ln n)$-APX, not $c\log n$-APX	$(2+\epsilon)$-APX, for any $\epsilon > 0$

Table 2.1: Hardness and approximability: Our and previous results. (All NP-hardness results are in strong sense, thus implying the non-existence of a FPTAS. Previous results are displayed between brackets.)

problems, implies an approximation preserving reduction ($\leq_{\mathsf{AP}}$, see [ACG+99]) between all these problems. Let $f(n)$-APX denote the class of problems that admit a polynomial-time $f(n)$-approximation algorithm [ACG+99]. In Table 2.1 we summarize the complexity and (non-) approximability results of this work.

As for the results on planar graphs, our technical contribution is a Polynomial-Time Approximation Scheme (PTAS) for Roman Domination. This result is based on an exact polynomial-time algorithm for r-outerplanar graphs. The latter improves over the previous results in [Dre00]: in this work only trees and $r \times n$-grids (for any *fixed* r) are shown to be exactly solvable. Our result subsumes both of them (an $r \times n$-grid is clearly an r-outerplanar graph).

This chapter is organized as follows. In section 2.2, we present the definition of *Online Static Win–Win* and discuss the complexity. Section 2.3 does the same for Roman Domination. Additionally, for planar Roman Domination, a Polynomial Time Approximation Scheme (PTAS) is given. In sections 2.4,2.5, variants of Win-Win are discussed. The chapter is summarized in section 2.6.

2.2 Online Static Win–Win

In this section, we assume that we know nothing about the requests in advance. This is formalized in the following definition.

Definition 2.1 (online static) *Given a graph $G = (V, E)$, a* server placement *for G is a multi set S of nodes. A server placement S is a* win–win *for G, if for all $v \in V$ there is an $u_v \in S$ with the properties:*

1. *$v = u_v$ or $(u_v, v) \in E$,*
2. *for all $v' \in V \setminus \{v\}$ there is an $u_{v'} \in S \setminus \{u_v\}$ with*

$$v' = u_{v'} \text{ or } (u_{v'}, v') \in E.$$

The definition leads to the following lemma.

Lemma 2.2 (sandwich) *Any graph G has the following properties:*

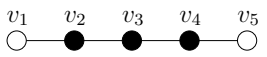

Figure 2.2: A win–win.

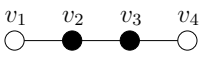

Figure 2.3: Not a win–win.

1. *For every dominating set DS, the server placement $SP := DS \uplus DS$ is a* win–win *for G, where $\uplus$ denotes the multi-union.*

2. *For every* win–win *WW, the set* uniq(WW) *is a dominating set of G.*

3. *For every minimum dominating set MDS and for every minimum* win–win *MWW, $|MDS| \leq |MWW| \leq 2|MDS|$ hold.*

Proof: For Property 1, let $v_1, v_2 \in V$ be a pair of nodes with $v_1 \neq v_2$. Since DS is a dominating set, there are $u_{v_1}, u_{v_2} \in DS$, such that

$$\begin{aligned} v_1 &= u_{v_1} \text{ or } (v_1, u_{v_1}) \in E, \text{ and} \\ v_2 &= u_{v_2} \text{ or } (v_2, u_{v_2}) \in E \end{aligned}$$

hold. Due to the definition of SP, $\{u_{v_1}, u_{v_2}\} \subset SP$ holds. This implies that requests at v_1, v_2 can be serviced.

For Property 2, let $v \in V$ be a node. There is a $u_v \in WW$ with $v = u_v$ or $(v, u_v) \in E$. Since $u_v \in$ uniq(WW), uniq(WW) is a dominating set.

For Property 3, it suffices to consider the win–win $WW := MDS \uplus MDS$ and the dominating set $DS :=$ uniq(MWW). Clearly, $|MDS| \leq |DS| \leq |MWW| \leq |WW| = 2|MDS|$. □

2.2.1 Characterization of win–win Multi sets

The property of being a win–win does not depend only on a node and its neighbors. Furthermore, it is not enough that for every pair of nodes there are two different adjacent servers. This is illustrated by the example in Figure 2.3. The server placement $S = \{v_2, v_3\}$ is not a win–win. If the first request is at v_2, then there are two cases. Case 1, the request is serviced by v_2, then a second request at v_1 cannot be serviced. Case 2, the request is serviced by v_3, then a second request at v_4 cannot be serviced.

This observation lead us to the following characterization of the server placements that are win–win.

Definition 2.3 *Given a graph $G(V, E)$ and a multi set D for it, a vertex $v \in V$ is* weak *if D dominates v only once. A vertex $u \in D$ is* safe *if every $v \in N(u)^+$ is not weak, where $N(u)^+ = N(u) \cup \{u\}$.*

Lemma 2.4 *A multi set D for $G(V, E)$ is a* win–win *if and only if the following two properties hold:*

at-most-1-weak *Every $u \in D$ does not dominate more than one weak node;*

at-least-1-safe *Every non weak node $v \in V$ is dominated by at least one safe node $u \in D$.*

Proof:
($\Rightarrow$) By contradiction, assume that some $u \in D$ does not satisfy Property at-most-1-weak. Then, there exist two weak nodes w_1 and w_2 dominated *only* by u. After a first request at w_1, w_2 is no longer dominated (we must have used u for the first request). This contradicts the hypothesis that D is a win–win. Now suppose (again by contradiction) that a non weak node v is not adjacent to any safe node (thus contradicting Property at-least-1-safe). Let $u_1, \ldots, u_k$ be the nodes of D adjacent to v, for some $k \geq 2$ (this follows from the fact that v is not weak). By hypothesis, none of $u_1, \ldots, u_k$ is safe. So, there exist $w_1, \ldots, w_k$ distinct weak nodes, with w_i adjacent to u_i, for $1 \leq i \leq k$. Now consider a first request at node v. For this request we must use one among $u_1, \ldots, u_k$, let us say u_j. Then, if the second request is at the weak node w_j we do not have any server to react. Again, this contradicts the hypothesis.
($\Leftarrow$) Let v_1 be the position of the first request. We have two cases: v_1 is weak, or v_1 is not weak. In the first case, we must use the only node $u \in D$ that is adjacent to v_1; Property at-most-1-weak guarantees that every node in $N(u)^+ \setminus \{v_1\}$ will still be dominated. So, any second request can be handled. Otherwise, that is, v_1 is not weak, Property at-least-1-safe implies that there exists a $u \in D$ which is safe; we use such a u for this request. At this point all the nodes in $N(u)^+ \setminus \{v_1\}$ will still be dominated by some $u' \in D$. Also in this case any second request can be handled. □

2.2.2 Complexity

We are interested in the complexity of the Online Static Win-Win problem. We discuss hardness and approximation of this problem. Both NP-hardness and approximation hardness can be proved using the following lemma.

Lemma 2.5 *Any $f(n)$-approximation algorithm A for* Min Dominating Set *implies a $2f(n)$-approximation algorithm for* Min Online Static Win-Win. *Conversely, any $g(n)$-approximation algorithm B for* Min Online Static Win-Win *implies a $2g(n)$-approximation algorithm for* Min Dominating Set.

Proof: Applying A to any graph G we can find a dominating set DS of size $|DS| \leq f(n)|MDS_G|$. By Lemma 2.2 the server placement $SP = DS \uplus DS$ is a win–win for G of size $|SP| = 2|DS| \leq 2f(n)|MDS_G| \leq 2f(n)|MWW_G|$.

Conversely, applying B to any graph G we obtain a win–win SP of size $|SP| \leq g(n)|MWW_G|$. Then, according to Lemma 2.2 the set $DS = \text{uniq}(SP)$ is a dominating set of size $|DS| \leq |SP| \leq g(n)|MWW_G| \leq 2g(n)|MDS_G|$. □

We know that Min Dominating Set is not approximable within $c \log n$ for some $c > 0$ [RS97] (unless P=NP) and that it is approximable within $1 + \ln n$ [Joh74]. From these facts and the above lemma one can easily prove the following.

Theorem 2.6 *The* Min Online Static Win-Win *problem in general graphs can be approximated within $2 + 2\ln n$, but (unless P=NP) cannot be approximated within $c \log n$ for some $c > 0$.*

For Min Dominating Set in planar graphs a Polynomial Time Approximation Scheme (PTAS) is known [Bak94]. Therefore, Lemma 2.5 implies an approximation algorithm for Min Online Static Win-Win in planar graphs, called Min Planar Online Static Win-Win, with ratio $2 + \epsilon$ for every $\epsilon > 0$.

Moreover, this approximation ratio is tight for the approach of "doubling" a dominating set to construct the solution. We illustrate this by the example in Figure 2.4. For this graph, the set $M := \{v_1, \ldots, v_8\}$ is a minimum dominating set. Doubling it gives a solution WW with $|WW| = 16$. On the other hand, the server placement $MWW = \{w, v_1, v_2, \ldots, v_8\}$ is a minimum win–win with $|MWW| = 9$. In this case, the approximation ratio is 16/9. If we increase the number of rays from 8 to k, then we get

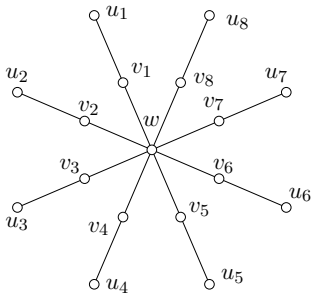

Figure 2.4: Doubling a dominating set gives a win–win of cost roughly twice the optimum.

$|WW|/|MSP| = 2k/(k+1)$. This shows that there exist graphs for which the simple doubling algorithm has approximation ratio greater than $2 - \epsilon$, for any $\epsilon > 0$.

2.3 Roman Domination

We come back to the original problem of the so called Roman Domination. On every node, we can place none, one, or two servers.

Definition 2.7 (roman domination) *Given a graph $G = (V, E)$, a* roman *for G is a server placement S such that every node v in V either belongs to S or has a neighbor u in S whose multiplicity in S is at least 2. Formally, $\forall v \in V, v \notin S \rightarrow \exists u : (v, u) \in E \wedge \{u, u\} \subset S$.*

Clearly, every roman S is a win–win: If the first request is at a node $v \in S$, then v is serviced by its own server; if $v \notin S$, then v is serviced by a neighbor u with $\{u, u\} \in S$. This implies that a minimum win–win does not have cardinality larger than a minimum roman. The next result shows that the '$\preceq$' relationship between those two problems is actually strict:[2]

***Strict Inclusion* 2.8** Online Static Win-Win$\prec$ Roman Domination*:*
For the graph in Figure 2.2, the server placement $S' = \{v_2, v_2, v_4, v_4\}$ is a minimum roman. On the other hand, $S = \{v_2, v_3, v_4\}$ is a minimum win–win: if the first request is at v_2, then this request is serviced by v_3; if after that the second request is at v_3, then it is serviced by v_2 or by v_4. ♢

It is known that Min Roman Domination is NP-hard for arbitrary graphs [Dre00]. We strengthen this result and show that the problem is also hard to approximate. As a by–product, we get a new proof for the NP-hardness. In particular, Lemma 2.2 remains true if we replace the notion of win–win by roman (see also [Dre00, Proposition 2.1]). Hence, we get the following theorem:

[2]Since in all cases '$\preceq$' is trivial, in the sequel we will only show that '=' does not hold.

Theorem 2.9 *The* Min Roman Domination *problem in general graphs can be approximated within* $2 + 2\ln n$*, but (unless* $P = NP$*) cannot be approximated within* $c \log n$ *for some* $c > 0$*.*

2.3.1 Planar Graphs

Often, our problem instances are not arbitrary graphs; planarity is quite a natural condition (see Figure 1.1). It is therefore interesting to study the problem complexity for planar graphs, since we know that minimum dominating set can be approximated well for planar graphs. It turns out that Min Roman Domination is NP-hard for planar graphs.[3]

Theorem 2.10 Min Roman Domination *is strongly NP-hard even if the input graph* G *is planar.*

Proof Sketch. We show the NP-hardness of Min Roman Domination by reducing Planar Vertex Cover [GJ79]. Indeed, in an adaptation of the well known reduction from vertex cover to dominating set, we can make the local transformation upon an edge in Figure 2.5. The resulting graph, with $|V| + 2|E|$ vertices and $5|E|$ edges is still planar,

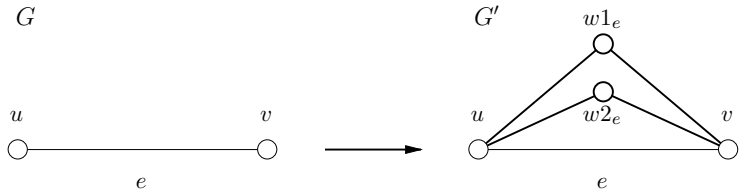

Figure 2.5: Reduction from Planar Vertex Cover to Roman Domination.

and it is straightforward to show that a vertex cover with k nodes in the original graph exists if and only if a roman with $2k$ nodes exists in the second. □

The results from the previous section show that the planar Min Roman Domination can be approximated within $2 + \epsilon$. The next theorem shows that we can find a better approximation. Its proof follows the ideas from [Bak94, ABFN00] which have become a well known standard method to get PTASs for many problems on planar graphs. Those approximations schemes look very similar; the only specific part is that the problem has to be solved optimally on r–outerplanar graphs. We use dynamic programming and the notion of *bounded treewidth* [ABFN00] to show how this can be done for the Min Roman Domination problem.

Theorem 2.11 (PTAS) Min Planar Roman Domination *has a Polynomial Time Approximation Scheme* (PTAS)*, but (unless* $P = NP$*) it does not have a Fully Polynomial Time Approximation Scheme* (FPTAS)*.*

Proof: Let G be a r–outerplanar graph. This implies that G has a treewidth l of at most $3r - 1$ ([ABFN00], Theorem 9). A tree decomposition $\langle \{X_i | i \in I\}, T \rangle$, with width

[3] In [Dre00, page 68], the NP-hardness of the planar graph case is also mentioned. At the writing time the chapter cited in [Dre00] is unpublished, so for the sake of completeness, we include a reduction from vertex cover. This reduction is also used to prove the "tightness" of our approximability results.

at most $3r-1$ and with $|I| = O(|V|)$ of G, can be found in $O(r|V|)$ time ([ABFN00], Theorem 12).

Let $\langle\{X_i | i \in I\}, T\rangle$ be a tree decomposition for the graph $G = (V, E)$. Let $X_i = \{x_1^{(i)}, \ldots, x_{n_i}^{(i)}\}$ be a bag [ABFN00] with $n_i := |X_i|$. A number $j \in \{0, \ldots, 3^{n_i} - 1\}$ can be identified with a server placement $S_j^{(i)}$ in the following way. We write j in ternary arithmetic, i.e., $j = \sum_{\nu=1}^{n_i} 3^{\nu-1} j_\nu$, where $j_\nu \in \{0, 1, 2\}$. Every node $x_\nu \in X_i$ occurs with multiplicity j_ν in $S_j^{(i)}$.

The algorithm we will describe visits the vertices of T from the leaves to the root. For every server placement $S_j^{(i)}$ of a bag X_i, the algorithm computes a server placement $\overline{S}_j^{(i)}$ for the bags in the subtree rooted at i as a partial solution.

The dynamic programming algorithm proceeds in three steps.

Step 1: For every leaf X_i, for every $j \in \{0, \ldots, 3^{n_i} - 1\}$, we define $\overline{S}_j^{(i)} := S_j^{(i)}$.

Step 2: After this initialization, we visit the vertices of our tree decomposition from the leaves to the root. Suppose node i has a child k in the tree T. In the case that i has several children $k_1, \ldots, k_s$ in the tree T, this step has to be repeated for each child.

1. Determine the intersection $Y := X_i \cap X_k$.
2. For every server placement $S_j^{(i)}$ of X_i, we choose a server placement $S_{j'}^{(k)}$ of X_k such that the following properties hold:
   (a) ${S_j^{(i)}}_{|Y} = {S_{j'}^{(k)}}_{|Y}$.
   (b) For every $v \in X_k \setminus Y$ with $v \notin S_{j'}^{(k)}$, there is a u_v with $\{u_v, u_v\} \subset \overline{S}_{j'}^{(k)}$ and $(v, u_v) \in E$.
   (c) The number $|(S_j^{(i)} \uplus \overline{S}_{j'}^{(k)}) \setminus ({S_j^{(i)}}_{|Y})|$ is minimized.

   Then, we define $\overline{S}_j^{(i)} := (S_j^{(i)} \uplus \overline{S}_{j'}^{(k)}) \setminus {S_j^{(i)}}_{|Y}$. For different $j_1, j_2 \in \{0, \ldots, 3^{n_i}\}$ with ${S_{j_1}^{(i)}}_{|Y} = {S_{j_2}^{(i)}}_{|Y}$, the same $j_1' = j_2'$ can be chosen.

   Note that, from Property 3 of a tree decomposition, we know that none of the nodes $v \in X_k \setminus Y$ will appear in a bag that has not been visited up to this point. Otherwise, such a node would also appear in X_i.

Step 3: Let X_R be the root of T, let $n := |X_R|$. Choose a $j \in \{0, \ldots, 3^n - 1\}$, such that

1. $\overline{S}_j^{(R)}$ is a roman for G, and
2. $|\overline{S}_j^{(R)}|$ is minimum.

The algorithm described above runs in time polynomial in the size of G and in 3^{3r}. Due to construction, for every vertex $i \in T$ and for every $j \in \{0, \ldots, 3^{n_i} - 1\}$, $\overline{S}_j^{(i)}$ is a smallest server placement such that property 2 (b) of step 2 is fulfilled. This implies that $\overline{S}_j^{(R)}$ is a minimum roman for G.

Finally, the strong NP-hardness proof of Theorem 2.10 implies that Roman Domination is not in FPTAS (see [GJ79] for the definition of strong NP-hardness and its implications). □

2.4 Online Dynamic Win–Win

In this section, we assume that after the first request, there is enough time to move the servers from one node to a neighbor before the second request occurs. This leads to the following definition.

Definition 2.12 (online dynamic) *Given a graph $G = (V, E)$ and a server placement S. A function*[4] *$r : S \to V$ is called* rearrangement *for G, S, if for every server $v \in S$*

$$r(v) = v \text{ or } (v, r(v)) \in E$$

holds. We say that S is a dynamic win–win *for G, if for every $u \in V$ there is a rearrangement r_u with the properties:*

- *There is $v \in S$ with $r_u(v) = u$, i.e., the first request at u can be serviced.*
- *For all $u' \in V \setminus \{u\}$, there is a $v' \in S \setminus \{v\}$ with $r_u(v') = u'$ or $(r_u(v'), u') \in E$.*

***Strict Inclusion* 2.13** Online Dynamic Win-Win$\prec$ Online Static Win-Win*:* Consider the cycle of length 4, $(v_1, \ldots, v_4, v_1)$. By one hand, the server placement $S = \{v_1, v_3\}$ is a dynamic win–win. For instance, if the first request is at v_2, then this request is serviced by v_1 and v_3 moves to v_4. On the other hand, there is no server placement S' which is a win–win with $|S'| = 2$. To see this, we consider two cases. Case 1, $S' = S$. A first request at v_2 must be serviced by v_1 or v_3, let us say v_1. Then a second request occurring at v_1 cannot be serviced. Case 2, $S' = \{v_1, v_4\}$. Consider a request at v_1. If we use the server at v_1, then v_2 is no longer dominated. Similarly, using the server at v_4 leaves v_3 undominated. ♢

Again, the methods from Section 2.2 can be used to show the complexity of Min Online Dynamic Win-Win.

Theorem 2.14 *The* Min Online Dynamic Win-Win *problem is NP-hard. It can be approximated within $2 + 2 \ln n$, but (unless $P = NP$) cannot be approximated within $c \log n$ for some $c > 0$.*

We know that finding a minimum dominating set is hard to do. What happens if we are given a server placement, and are asked if the arrangement is 'close to' a dominating set – that is, if each server is allowed to move at most 1 step, can a dominating set be obtained?

Definition 2.15 *Let r be a rearrangement for $\langle G, S \rangle$; r is called* dominating rearrangement *for $\langle G, S \rangle$, if the server placement $\{r(v) | v \in S\}$ contains a dominating set for G.*

Given a graph G and a server placement S, the Dominating Rearrangement problem asks whether there is a dominating rearrangement for $\langle G, S \rangle$.

Theorem 2.16 Dominating Rearrangement *is NP-complete. This remains true, even if the input graph is planar.*

[4]Note that different servers at a node can take different values.

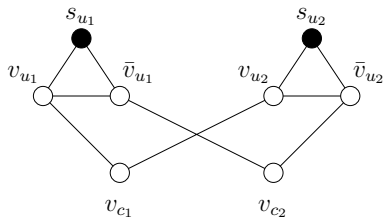

Figure 2.6: Reduction from SAT, $F = (u_1 \vee u_2) \wedge (\bar{u}_1 \vee \bar{u}_2)$.

Proof: It is obvious that this problem is in NP. We use a reduction from SAT [GJ79] to show the NP-hardness.

Let F be a Boolean formula, given as a set U of variables and a collection C of clauses over U. We define a graph $G_F = G = (V, E)$ as follows (see Figure 2.6). For every variable $u \in U$, there is a *storage node* $s_u \in V$ and two *variable nodes* $v_u, \bar{v}_u \in V$. Each such triple of nodes is connected by edges, i.e., $(s_u, v_u), (s_u, \bar{v}_u), (v_u, \bar{v}_u) \in E$.

For every clause $c \in C$, there is a *clause node* $v_c \in V$. A clause node v_c is connected to a variable node v_u ($\bar{v}_u$, resp.), iff $u \in c$ ($\bar{u} \in c$, resp.). On every storage node, a server is placed, i.e., the server placement has the form $S := \{s_u\}_{u \in U}$.

For every dominating rearrangement r and for every variable $u \in U$, either $r(s_u) = v_u$ or $r(s_u) = \bar{v}_u$ hold. It is obvious, that this corresponds to a variable assignment. The given formula F is satisfiable, iff there is a dominating rearrangement for $\langle G, S \rangle$.

To prove the NP-completeness for planar graphs, we define the subgraph $G' \subset G$ by deleting the storage nodes and the adjacent edges. G' is planar, iff G is planar. It has been shown in [Lic82, Lemma 1] that SAT is NP-complete, even if the input is restricted to formulae F with the property that G' and G are planar. □

Theorem 2.17 *Given a graph G and a server placement S. The problem to decide whether S is a* dynamic win–win *for G is NP-complete.*

Proof: We extend the definition of the graph G in the proof of Theorem 2.16. We add a *dummy node* $v_d \in V$, and we add edges from v_d to every clause node and from v_d to every variable node. The new server placement becomes $S := \{s_u\}_{u \in U} \uplus \{v_d\}$.

If the first request is at v_d, then this request has to be serviced by v_d, since no clause node and no variable node is in S. A second request can be serviced, iff there is a dominating rearrangement. We have seen that this is a NP-complete problem. □

2.5 Offline Static/Dynamic Win–Win

In this section, we consider the situation in which both requests occur at the same time (equivalently, as the first request must be serviced, it is already known where the second one will be).

Definition 2.18 (offline static) *Let $G = (V, E)$ be a graph. A server placement S is an* offline win–win *if for every pair of nodes $v_1, v_2 \in V$, $v_1 \neq v_2$, there is a pair $\{u_{v_1}, u_{v_2}\} \subset S$ with*

- *$v_1 = u_{v_1}$ or $(v_1, u_{v_1}) \in E$, and*

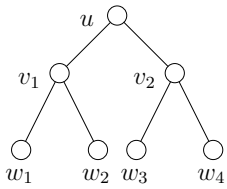

Figure 2.7: Proof of ***Non-Inclusion 2.20***

- $v_2 = u_{v_2}$ *or* $(v_2, u_{v_2}) \in E$.

***Non-Inclusion* 2.19** Online Dynamic Win-Win$\not\prec$ Offline Static Win-Win*:*
For the graph in Figure 2.3 the set $\{v_2, v_3\}$ is an offline win–win. For the same graph, no dynamic win–win can have size 2. Indeed, consider a first request at v_2. No matter what server we use to service this request, the remaining one cannot cover the nodes $\{v_1, v_3, v_4\}$, where a second request can occur.

***Non-Inclusion* 2.20** Offline Static Win-Win$\not\prec$ Online Dynamic Win-Win*:*
It is easy to verify that $\{u, v_1, v_2\}$ is a dynamic win–win for the graph in Figure 2.7. On the other hand, there is no offline win–win multi set of size less than 4: each of the subtrees rooted at v_1 or v_2 must contain at least two servers. ♢

Again, Min Offline Static Win-Win is an NP-hard problem, illustrated by the techniques of Section 2.2. Moreover, we can give the following characterization of the offline win–win multi sets:

Lemma 2.21 *A server placement S is an* offline win–win*, iff for every pair of two different nodes there is one server in the neighborhood of one node and a different server in the neighborhood of the other node.*

We conclude this section with Offline Dynamic Win-Win. Here we combine the fact that servers can be rearranged before serving the second request (Dynamic) with the fact that the second request is known by the time we have to serve the first one (Offline). Therefore, we have the following definition for the corresponding server placement:

Definition 2.22 (offline dynamic) *Let $G = (V, E)$ be a graph. A server placement S, is an* offline dynamic win–win *for G, if for every pair of nodes $v_1, v_2 \in V$, with $v_1 \neq v_2$, there is a pair of distinct nodes $u_{v_1}, u_{v_2} \in V$ such that v_i is at distance at most i from u_{v_i}, for $i = 1, 2$.*

***Strict Inclusion* 2.23** Offline Dynamic Win-Win$\prec$ Online Dynamic Win-Win*:*
Consider the cycle of length 5, $(v_1, v_2, \ldots, v_5, v_1)$. It is easy to verify that the set $S = \{v_1, v_3\}$ is an offline dynamic win–win (S is a dominating set and both servers are at distance at most two from any other non-server node). To prove that no multi set of size two can be a dynamic win–win we use the following argument. After the first request has been serviced, the set of nodes to be considered as possible positions for the second request induce a path of length four; therefore, no matter where we place the remaining server, there is no way to dominate all such nodes. ♢

2.6 Conclusion

We have looked at several Win-Win problems individually, but have also tried to explore the connections between them. First of all, every Online version is more "difficult" (i.e. requires more servers) than the corresponding Offline one (i.e. $\succ$). Similarly, every Static problem is more "difficult" than the corresponding Dynamic one. Additionally, our results show that the Online and the Dynamic features are somehow orthogonal: Online Dynamic Win-Win and the Offline Static Win-Win are simply not comparable.

We saw that Online Dynamic Win-Win$\prec$ Online Static Win-Win$\prec$ Roman Domination. In some sense, this ordering can be reversed: It is easy to check whether a server placement is roman, it is a little bit harder to do this for Online Static Win-Win, and it is NP–complete to do it for Online Dynamic Win-Win. For the decision which server should service the first request, similar results hold. For instance, Online Dynamic Win-Win needs much or central control or communication than Roman Domination. Maybe, this was the reason why emperor Constantine used Roman Domination instead of a Win-Win variant.

Chapter 3

Distributed Highly Available Search Trees

3.1 Introduction

In this chapter, we propose a scalable, distributed dictionary supporting *insert* and a variety of *search* operations for keys from a linearly ordered universe, say integers, in a challenging environment. It works in a totally asynchronous setting, where faulty servers cannot be detected, and it tolerates crashes: A single server crash is guaranteed to do no harm at all, and simultaneous crashes of more than one server are also often harmless, but with no guarantee. More specifically: First, the dictionary remains fully operational in the presence of a single failure, i.e., all search and insert operations work correctly and efficiently. Second, it enables efficient recovery, i.e., a server that has suffered a crash failure can reinvolve itself into the data structure, and after the recovery is complete, a client cannot distinguish whether a crash occurred. In addition, the dictionary supports its operations with a very small overhead (number of messages, memory) to achieve fault tolerance.

This chapter has the following structure. Section 3.2 gives an exact definition of the distributed mode. Section 3.3 reviews the distributed binary search trees, and Section 3.4 presents our proposal for a fault tolerant distributed dictionary based on distributed binary trees. Section 3.5 proves its properties. Section 3.6 discusses modifications, and Section 3.7 concludes the chapter.

3.2 The Model

Let us now be more precise about the distributed system and its availability. Let the set of computers be connected by a network. Computers communicate by sending and receiving messages. Every computer is identified uniquely by its address. We do not focus on message size, assuming that large buffers for incoming messages are available. We express this by assuming that messages can be arbitrarily large, but we will not exploit this in any extreme way. Message transmission is asynchronous in the following sense. First, no computer has access to a global clock. Second, every sent message will be delivered eventually, but message transmission times are unpredictable and not bounded. Third, messages can pass each other on different paths. For instance, if a computer C sends a message m_1 and then a message m_2 to a computer C', then it is possible that C' receives m_2 before m_1.

We distinguish *client* computers that initiate insert or search operations on the data set from *servers* that store data. Starting with one server S_0, more and more servers become involved as needed to keep the individual server load small; we consider the set of potential servers to be arbitrarily large for this purpose. The SDDS conditions can be satisfied with the following basic strategy. An SDDS is distributed among servers and is manipulated by requests from client sites which always have their own image of the structure. However, the image of the client can be outdated, because the SDDS may have, for example, split some of its buckets and distributed them between old and new servers. The structure is designed so that with an outdated image the client can find the correct bucket but cannot send the query directly to the new server. After this, a new updated image will be sent to the client so that it cannot make the same error twice. In this way the most important property of an SDDS is achieved: no bottlenecks are created because clients with updated information can usually send their queries directly to correct servers.

In our model, communication is reliable, but a server can break down. Therefore, we distinguish *operational* and *crashed* servers. An operational server is fully functional, while a crashed server can neither receive nor send messages nor perform a computation. An operational server can crash. A crashed server loses all its data, except for some small piece of data in a secure memory. The reason is that on one hand, a server cannot recover without any information at all about its role in the data structure, and on the other hand, it is not economically feasible to protect all its data against loss. For simplicity of the discussion, we assume that the first crash of a server can only happen after the server has been involved into the data structure. This assumption is not a serious restriction, as we will show in Section 3.6.1. Since a failure cannot be detected, we have to use a technique that can be classified as *Hot Standby* [GR93]; i.e., data is duplicated in a way where every key is stored on two different servers, and without failure both copies are equally well available. A crashed server can *recover*; this will change its *state* to operational, and the server will reinvolve itself into the data structure in a way that we will propose in detail.

Server S_0 has an exceptional role and does not crash. This may appear like a strong and unrealistic assumption, but it is neither. It is realistic, because securing a single machine in a network is easy and is routinely done in practice; it is not stronger than the minimum needed, since the impossibility of distributed consensus [FLP85] suggests that without this assumption, the desired behavior cannot be achieved. Furthermore, to obtain the required properties, we will see that our data structure also needs a secure and central entity, the so called *split manager*. Although this is not desired in a distributed system, its existence is not a serious drawback. In fact, all distributed data structures in the literature need some central entity to do a small but important piece of work. For instance, in [KW94] such an entity finds the next server that can be involved into the data structure, and [LNS93] needs a central entity as split coordinator. For convenience, we propose to implement the split manager within S_0, but other implementations are possible. We avoid the discussion of operational details that are local in a server, and assume that receiving and processing a message and sending messages as a consequence is one atomic step.

3.3 Distributed Binary Search Trees

We present a solution to the fault tolerant distributed dictionary problem with distributed binary leaf search trees [KW94], in which the keys (and data) are stored in leaves that are similar to B-tree leaves and the internal nodes are binary routers. The tree structure not

only supports the access of single records but also allows a variety of efficient similarity queries. We limit ourselves to the explicit discussion of the former, since it will become clear how to perform the latter. The nodes of the tree are stored in servers, and the edges are communication links. In this section we shortly review the binary tree structure, for more details we refer to [KW94].

With every node u, we associate a nonempty *responsibility interval* $I_u \subset \mathbb{Z}$, represented by a pair $l_u, r_u \in \mathbb{Z} \cup \{-\infty, \infty\}$. Node u is *responsible* for a key k, if $k \in I_u$. Each internal node u with left child v_1 and right child v_2 has a *split value* $\sigma_u \in I_u$ such that $I_{v_1} = [l_u, \sigma_u]$ and $I_{v_2} = [\sigma_u + 1, r_u]$. For every key $k \in \mathbb{Z}$ there is exactly one leaf v that is responsible for k. Leaves store keys, with leaf v storing a set $K_v \subseteq I_v$ of keys.

In our model, each server can hold a constant number of leaves. Internal nodes contain routing information only, and there is no a-priori bound on the number of internal nodes a server can contain.

In a state in which no insert operations are under way, each client has a picture of that part of the binary tree structure above all leaves it has accessed. All searches by clients will be directly sent to leaves and servers in the known part of the tree. If queries are evenly distributed, so is the load of servers; even if the distribution of queries and insertions is skewed, the load of servers can be leveled out nicely with a concept proposed in [VBW98].

Initially, the structure contains one leaf only, and whenever, due to insertions, a split occurs a new server is taken to store one half of the records the split node was responsible for. A split also means that a new internal node must be created; this will be stored in the old server that held the split node, cf. Figure 3.1. Consider then the situation depicted

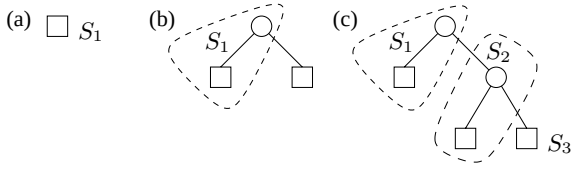

Figure 3.1: The distributed tree grows from one leaf in server S_1 (a) to two leaves and servers (b), and further to three leaves and servers (c).

in Figure 3.1, and assume that a client has an out-of-date picture of the structure telling that there is only one leaf even though the true picture is as given in Figure 3.1(c).

Assume further that the client searches for a key which has been moved to server S_3. Using its out-of-date picture the client sends the query to server S_1, and seeing that the required key is not in its own leaf, S_1 sends according to the tree structure the query to S_2. Similarly, S_2 sends the query to S_3 whose leaf is responsible for the key of the query.

After the query has been performed and the client has obtained the corresponding information, the search path used is also sent to the client. Then, for all keys the nodes in this path are responsible for, this path is first used in the client before accessing the global structure.

3.4 Highly Available Trees

Intuitively, the *Highly Available Tree (HAT)* that we now propose duplicates data and stores copies in two different distributed binary leaf search trees. Both trees are kept as similar as possible, but failures and different execution speeds may create differences

between the trees that will be repaired as operations progress. Every node u in one tree has an associated node u' in the other tree, its *buddy*. Every request is performed on both trees simultaneously.

To be more precise, a HAT H consists of a pair of rooted binary trees T_l and T_r with roots r_l, r_r. Initially, each of the trees $T_\bullet$ consists of its root $r_\bullet$ only. The trees grow by splitting a leaf into an internal node with two children.

We define the *buddy operator*, denoted by a prime symbol $'$, as follows. If r_l is the root of T_l and r_r the root of T_r, then $r_l' := r_r$ and $r_r' := r_l$. If u is an internal node and v is its left (right) child, then v' is defined as the left (right) child of u'. For any other pair of nodes w_1, w_2, $w_1' \neq w_2$ and $w_2' \neq w_1$ hold. Observe that for any node u, we have $u'' = (u')' = u$. The parent of a node u is denoted by $p(u)$, the sibling of u is denoted by $s(u)$, see Figure 3.2. If $u' = v$ for two nodes u, v, then v is the *buddy* of u and u is the buddy of v.

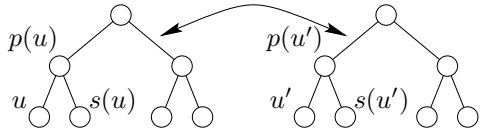

Figure 3.2: Pair of two isomorphic trees

It follows that a node has at most one buddy; a node can send a message to its buddy's address anytime, even though this buddy need not be part of the dictionary at the moment at which the message is sent. A node u knows the addresses of its sibling and its parent (if u is not a root). If u is an internal node, it knows the addresses of its children.

In H, two trees T_l, T_r are linked in the following way. Every node u is restricted to have the same responsibility interval as its buddy u', i.e., $I_u = I_{u'}$, and u knows the address of u'. Furthermore u knows the address of its buddy's parent $p(u')$ and sibling $s(u')$. If u' is an internal node with children v_1' and v_2', then u knows the addresses of v_1' and v_2' or will know these addresses eventually.

A node u can send messages to node v, if u knows the address of v.

To simplify notation, we do not distinguish between an object and its address, i.e., u denotes a node or the address of this node, c denotes a client or the address of this client. If a node u receives a message m, then u is also informed about the address of the sender of m; in particular, u knows whether m was sent from its parent $p(u)$, its buddy u', or a client c.

3.4.1 Mapping Nodes to Servers

The described structure has to be mapped to the set of servers, i.e., every node u is stored on a server $S(u)$. The set of all potential servers is enumerated $\{S_0, S_1, S_2, \ldots\}$, but this enumeration and the corresponding addresses of the servers need to be known only to S_0. On the other hand, the addresses of S_0, S_1, and S_2 are known to every computer. Server S_0 maintains a counter *new* for the relative number of the server to be involved next. Since we start with two root servers, initially $new = 3$. A server S is called *involved*, if there is a node u with $S = S(u)$. For the resilience against 1–server failures it is important that a node and its buddy are not on the same server, i.e., $S(u) \neq S(u')$ for every node u. In order to ensure scalability, no server is allowed to store more than a constant number

of leaves. Since internal nodes contain routing information only, we do not impose an a–priori bound on the number of internal nodes on a server.

The allocation rule maps the nodes as follows (Figure 3.3):

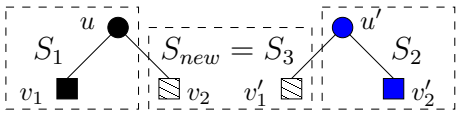

Figure 3.3: Mapping trees to servers

1. The root of T_l is stored on S_1, the root of T_r is stored on S_2.
2. Let $u \in T_l$ be a leaf that performs a split and becomes an internal node with left child v_1 and right child v_2. Then
   (a) The left child v_1 is stored on $S(u)$, i.e., $S(v_1) = S(u)$.
   (b) The right child v'_2 is stored on $S(u')$, i.e., $S(v'_2) = S(u')$.
   (c) The nodes v_2, v'_1 are stored on a new server S_{new}, i.e., $S(v_2) = S(v'_1) = S_{new}$.

The construction immediately implies:

Lemma 3.1 *The allocation rule of mapping HAT nodes to servers has the following properties:*

1. *Buddies are on different servers. More formally, for every node u, $S(u) \neq S(u')$ holds.*
2. *No server stores more than two leaves.*

Note that a message from a node u to a node v has to be sent from the server $S(u)$ to the server $S(v)$, and $S(v)$ has to locally hand the message over to v. We simplify notation by letting nodes act (instead of servers only) and therefore say that this message goes directly from u to v even if $S(u) = S(v)$.

It is possible that a node u is stored on a crashed server $S(u)$. To simplify notation, we call u *crashed*, if the server $S(u)$ is crashed or has not performed its recovery. Otherwise u is called *operational*.

3.4.2 Dictionary Operations

This section shows how the described structure can be used to implement the dictionary operations search and insert.

Initialization

A HAT H is initialized as empty. The empty structure H consists only of the roots r_l of T_l and r_r of T_r. Let r_l be stored on S_1, r_r on S_2. For the responsibility intervals, we get $I_{r_l} = I_{r_r} = \mathbb{Z}$. The two nodes do not contain keys, i.e., $K_{r_l} = K_{r_r} = \emptyset$. The address of r_l is stored in the secure memory of S_2, and the address of r_r is stored in the secure memory of S_1.

Search

A search request of a client c for a key k is performed simultaneously in both trees of H, since c cannot decide whether there is a failure in a tree. To perform an operation, c sends two messages to the roots r_l, r_r, from which the messages are forwarded to the responsible children, until the messages reach the responsible leaves. If a leaf receives a search request message, it sends its response about k to c, telling whether k has been inserted or not. Since in this naive approach the roots become bottlenecks, we apply the ideas proposed in [KW94] of informing clients about the tree structure in a lazy fashion so that they can start future searches further down in the tree. This will be discussed in Section 3.6.2.

Whereas in an ordinary leaf search tree the search request leads to a search path from the root to the responsible leaf, in a HAT such a request defines a pair of search paths, see Figure 3.4.

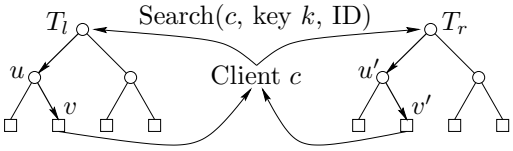

Figure 3.4: Simultaneous search in a HAT

But, as a consequence of the weakness of our assumptions about the distributed system, such a pair of independent search paths is not enough: If in Figure 3.4 the nodes $r_l = p(u)$ and u' crash, then the search request from client c cannot reach the responsible leaves. In order to safely bypass crashed nodes without loss of information, we connect the search paths in the following way. If a node u sends a message m to a child v, then u sends m to also v'. If there is no crash failure, v receives two copies of the same message, one from u, the other from u'. Theorem 3.7 proves that in this way, crashed nodes can be bypassed.

This, however, causes the next problem: If a node sends two messages for every message it receives, the number of messages doubles at every node on the search path. Since this exponential growth is totally unacceptable, we control it by keeping track of what happened as follows. If a client c wants to search for a key k, it chooses an identifier ID that is unique w.r.t. c, for instance the number of inserts and searches c has requested so far. Then c sends the message $Search(c, k, \mathrm{ID})$ to the roots of H. If a node receives a message, this node keeps track with a *message tag* for the pair c, ID of receiving this message, and it forwards the message to its corresponding child v and to v'. If a second copy of this message is received later, the node recognizes from the tag that it has forwarded the message before and hence does not forward the message again.

Now, a (smaller) problem is that a tag has to be stored in memory. To save memory, a tag should be deleted after a while. One option is to delete the tag after the node has received the second copy of the message (if the node is a root, then no tags are needed). But now we run into a further problem: It is possible that a tag is never deleted. In the example of Figure 3.5, a search message is sent to buddies u and u'. Both, u and u', are about to split. Now, assume that node u' has already performed its split, while u has not. Since u is a leaf, it responds to the client. But since u' is an internal node, it forwards the search request to the responsible child v_2' and to v_2. The corresponding tags in u and u' will be deleted. But the children v_2, v_2' receive only one copy of the message, and therefore, their tags will never be deleted. Although tags may seem like a rather small

problem, their number might increase with every search operation, and on the long run, this is clearly undesirable. We therefore choose to avoid the possible unbounded growth of the number of tags with the following concept.

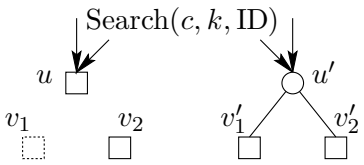

Figure 3.5: Non isomorphic trees

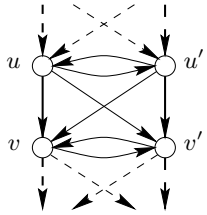

Figure 3.6: Search ladder

If u forwards the search message to children v and v', or responds to a client c, then u sends the message together with the addresses of v and v' or c to its buddy u', called a *cross message*[1]. Lemma 3.2 shows that cross messages suffice to achieve the desired properties. We call the pair of connected search paths, including cross messages, a *search ladder*, see Figure 3.6.

In more detail, the tags work as follows. If a node u receives the c, ID pair in a message $Search(c, k, \mathrm{ID})$ for the first time, then u creates a *tag*. Such a tag consists of 3 entries:

- *Message:* The message identifier c, ID.
- *Received from:* The addresses of the nodes or the client c from which u has received the message in the past. Initially, this entry is empty. After having received the message $Search(c, k, \mathrm{ID})$ from a sender, u adds the address of this sender to the entry.
- *Sent to:* The addresses of the nodes or the client c to which the message is sent. If u is a leaf, then it sends messages to c and to u'. If u is an internal node, messages are sent to children v and v' and to u'.

If u is the responsible leaf for key k, then u sends a response message to client c. If u is an internal node, then u forwards the message $Search(c, k, \mathrm{ID})$ to its child v and its buddy's child v' with $k \in I_v = I_{v'}$. In both cases a message is sent to u', too.

If u receives a message from a client, then it expects to receive the message twice, once from the client and once from its buddy u'. If u receives a message from a parent node $p(u)$ or $p(u')$, then u expects to receive this message three times: From $p(u)$, $p(u')$, and u'. The message tag is deleted after u has received the messages from all the expected senders.

This mechanism lets u forward a message only once, although u receives that same message up to three times. But unfortunately, it also introduces an extra complication in the following situation: Leaf u has performed a split, while u' has not. If u has sent the message to children v and v' while u' has sent an answer to the client c, then the following happens. By receiving the message from u, u' recognizes that its buddy has links to their children while itself has not. To remedy the situation, now u' sends the message $Search(c, k, \mathrm{ID})$ to v and v' and changes its tag entry "Sent to" from (c, u) to (v, v', u). After that u' will perform a *split*, see Section 3.4.3. Hence, the search along connected paths in this way correctly manages the tags, provided that no crash failures occur.

[1]Cross messages can be avoided if both nodes have performed their splits and have become internal nodes.

Insert

To insert key k into the HAT, client c sends the message *Insert*(c, k, ID) to the roots r_l, r_r. In the same way as a search message, this insert message is forwarded through the HAT. Each internal node u that receives *Insert*(c, k, ID) forwards the message to the buddy u' and to children v and v' with $k \in I_v$. A tag for this message is created.

Eventually the message will reach a leaf. If k is in the responsible leaf u, i.e., $k \in K_u$, then u sends the message *NotInserted*(k, ID) to c; otherwise k is inserted into u, i.e., $K_u \mapsto K_u \cup \{k\}$, and u sends the message *Inserted*(k, ID) to c.

Insertions into a leaf u could lead to a *split* of u, see section 3.4.3.

3.4.3 Split

A split of a leaf u is necessary if the set of stored keys K_u and therefore the workload becomes too big. During a split, the leaf u becomes an internal node with left child v_1 and right child v_2.

Because a node and its buddy must have the same responsibility interval, u and u' must have the same split value $\sigma_u = \sigma_{u'}$. Since the split should be adaptive, the split value has to be chosen according to the key sets $K_u, K_{u'}$, for instance as their median key (a merely space dependent split value such as $l_u + \lfloor \frac{r_u - l_u}{2} \rfloor$ is not sufficiently adaptive). But, since insertions can be made in u and u' in different order, K_u and $K_{u'}$ can differ greatly in size. Therefore, no node alone can choose the split value, we need a consensus between u and u'. In [FLP85] it was shown that such a consensus between two or more nodes is impossible in our model; there is no protocol that guarantees consensus in the presence of failures.

We arrive at a split value decision by circumventing the impossibility of consensus: The split value is chosen by a central entity, the *split manager*. The split manager is stored on the fail–safe server S_0, and hence every node knows its address.

As a suggestion for the split value σ_u, u computes the median $\tilde{\sigma}_u$ of its key set K_u. Then u sends the message *SplitRequest*$(u, u', \tilde{\sigma}_u)$ to the split manager. After having received the first split request from u or its buddy u', the split manager selects 4 new nodes v_1, v_2, v_1', v_2' (according to the scheme described in Section 3.4.1), decides the split value σ_u (picking either $\tilde{\sigma}_u$ or $\tilde{\sigma}_{u'}$), and sends the message *SplitGrant*$(u, u', v_1, v_2, v_1', v_2', \sigma_u)$ to u.

The split manager responds with *SplitGrant*$(u', u, v_1', v_2', v_1, v_2, \sigma_u)$, if it receives a split request from u'. This implies that the split manager keeps track of that split, using a *split tag*. If the split manager receives another split request from u' or u, it uses the split tag to reconstruct the *SplitGrant* message sent before and to reuse the previously chosen split value. As before, to save memory, the split tags should be deleted as soon as they have become useless.

Again, this requirement introduces an extra difficulty. Because a leaf can perform many splits, it is not enough that a tag is deleted after each of both corresponding nodes has performed a split. To see this, assume that a leaf u had performed a split before it had a crash failure. Its buddy u' has not performed a split. If u sends a recover request to u', then u is recovered as a leaf. Therefore, u can split again, and it can repeat this over and over.

We solve this problem by keeping track of the split event history in the split tags. A split tag $t(u, u')$ consists of 5 parts:

- *Nodes to split:* The addresses of the nodes u, u'.

- *Split value:* The split value σ_u that the split manager selected.
- *New nodes:* The addresses of the 4 children v_1, v_2, v_1', v_2'.
- *Split Performed:* Two flags indicating whether the split manager has received the forwarded messages about performed splits from u, u'. The flags are initialized as "no".

After having received the split grant, u uses the split value σ_u to compute new intervals $I_1 := [l_u, \sigma_u]$, $I_2 := [\sigma_u + 1, r_u]$ and key sets $K_1 := K_u \cap I_1$, $K_2 := K_u \cap I_2$. Then u sends the message $\mathit{Initialize}(u, u', v_i, v_i', s(v_i), s(v_i'), I_i, K_i)$ to v_i and the message $\mathit{Initialize}(u', u, v_i', v_i, s(v_i'), s(v_i), I_i, K_i)$ to v_i'. Then u stores the addresses of v_1, v_2 as its children and the addresses of v_1', v_2' as its buddy's children and deletes its key set K_u. Furthermore, u sends the message $\mathit{SplitPerformed}(u)$ to its buddy u'. Then u is an internal node.

Eventually u' receives the message $\mathit{SplitPerformed}(u)$. If u' is already an internal node, then u' forwards the message $\mathit{FwSplitPerformed}(u)$ to the split manager. If u' is still a leaf, $\mathit{SplitPerformed}(u)$ forces u' to begin its own split. After becoming an internal node, u' sends the message $\mathit{FwSplitPerformed}(u)$ to the split manager.

The $\mathit{Initialize}(u, u', v_i, v_i', s(v_i), s(v_i'), I_i, K_i)$ message reaches the server $S(v_i)$ eventually. If node v_i has not been initialized already by a message from u', then $S(v_i)$ initializes v_i with responsibility interval $I_{v_i} := I_i$ and key set $K_{v_i} := K_i$. The addresses $u, u', v_i', s(v_i), s(v_i')$ are the addresses of the parents, the buddy, the sibling and its buddy's sibling. The buddy address v_i' is stored in the secure memory of $S(v_i)$. If a second message $\mathit{Initialize}(u', u, v_i, v_i', s(v_i), s(v_i'), I_i, K_i')$ is received, then K_i' is inserted in K_{v_i}, i.e., $K_{v_i} := K_{v_i} \cup K_i'$.

If the split manager receives a message $\mathit{FwSplitPerformed}(u)$, then it sets the corresponding entry in the split tag $t(u, u')$ to "yes". If both entries are set to "yes", then $t(u, u')$ is deleted.

As a detail on the side that illustrates the intricacy of our mechanism, note that since every node v knows the address of its buddy v', it could happen that v sends a message to v', although the server $S(v')$ does not know of v' yet. However, the server $S(v')$ stores the message in a large enough queue for future service. After v' has been initialized, it works on the messages that have been received before by $S(v')$. Therefore, we can safely assume that every node has a buddy at all times.

3.4.4 Recovery

Let S be a server after a crash, whose state changes from crashed to operational. Then S starts the following *recovery protocol*. According to the model, S knows the address of the buddy of at least one of its nodes from the secure memory. Let u' be one of these addresses, indicating that there was a node u on S before the crash. Every such node u on the server now sends the message *RecoverRequest()* to its buddy u'. If u' is an internal node, it answers with the message $\mathit{Recover}(I_u, v_1, v_2, v_1', v_2', \sigma_u, p(u), p(u'), s(u), s(u'))$, where I_u is the responsibility interval of u and u'; v_1, v_2, v_1', v_2' are the addresses of the children and their buddies; σ_u is the split value; $p(u), p(u')$ are the addresses of the parents; $s(u), s(u')$ are the addresses of the siblings. If u' is a leaf, then the answer is $\mathit{Recover}(I_u, p(u), p(u'), s(u), s(u'), K_{u'})$, where $K_{u'}$ is the set of keys of u', and the rest is as before. After having received and processed this recover message, server S checks whether there is a child v_i, a parent $p(u)$ or a buddy's sibling $s(u')$ that should be on S

and that has no address of its buddy in the secure memory. If so, S initializes this node and sends the recover request to its buddy.

After every node on the server has received and processed the recover message from its buddy, the recovery protocol is finished.

Observe that it is enough for a server in the recovery process to know one node's buddy, even if many nodes were stored on that server before the crash. To see this, let u, v be two nodes on the same server, i.e., $S(u) = S(v)$. If both nodes are in the same tree, w.l.o.g. $u, v \in T_l$, then these nodes are on a path from the root to a leaf, i.e., one node is the descendant of the other. If $u \in T_l$ and $v \in T_r$, then there are nodes $w_l \in T_l$ and $w_r \in T_r$ with the properties:

- u is descendant of w_l.
- v is descendant of w_r.
- w_l is the sibling of the buddy of w_r, i.e., $w_l = s(w_r')$.

Therefore, the recovery protocol can recover every node on a server, if the address of only one buddy is available and none of these buddies is crashed.

3.5 Properties

Let us now argue that the mechanism described in Section 3.4 indeed leads to the desired behavior. Let H be a HAT with subtrees T_l, T_r. In its history, H starts out as empty data structure with two roots r_l, r_r. Then a finite sequence of insert and search operations $O = \{o_1, o_2, \ldots, o_n\}$ is performed on H, and there are no other operations on H. During this phase, servers are allowed to have crash failures. Eventually, H reaches a state in which no messages are sent or received any more; measured in global time, we call this moment τ_0. Let us now observe H from an external point of view. We look at the set $\mathcal{S}$ of involved servers and the set $F \subset \mathcal{S}$ of servers that have had crash failures before τ_0. For this situation, we get the following results.

Theorem 3.2 (Overhead) *If no crash failure occurred, i.e., $F = \emptyset$, then every tag (message tag or split tag) has been deleted. Furthermore, the number of messages sent to perform O on H is less than 7 times the number of messages in each one of the underlying trees T_l or T_r.*

Proof: We prove the theorem by counting the messages that are sent to perform the operations O. A message can be sent (1) from a node to another node, or (2) from a node to the split manager, or (3) from the split manager to a node, or (4) from a leaf to a client. Let $\mathcal{M}_H(O)$ be the set of these messages, and let $M_H(O) := |\mathcal{M}_H(O)|$. This number will be compared to the number of messages in the tree T_l. A message $m \in \mathcal{M}_H(O)$ is called an *internal message* of T_l, if m is sent (1) from a node $u \in T_l$ to a node $v \in T_l$, or (2) from a node $u \in T_l$ to the split manager, or (3) from the split manager to a node $u \in T_l$, or (4) from a leaf of T_l to a client. To perform O on T_l, these internal messages have to be sent. In order to compare $M_H(O)$, let $M_l(O)$ be the number of internal messages of T_l.

Lemma 3.3 *If no crash failure occurs, then $M_H(O) < 7M_l(O)$ holds.*

Proof: We regard every possible case and compute the factor that bounds the number of messages in this case.

Case 1: If u, u' are internal nodes that receive a message, e.g., a $Search(c, k, \mathrm{ID})$ message, for the first time, then due to their routing scheme u and u' send 6 messages. One of them is an internal message of T_l.

Case 2: If two leaves u, u' receive a message, then each sends 2 messages, one to the client c and one to the buddy. One of these 4 messages is counted in $M_l(O)$.

Case 3: It is possible that a node u has performed its split while u' has not. In this case at most 7 messages are sent. u sends messages to two nodes v, v' and to u'; u' sends messages to u and to the client c; by receiving the message from u, u' recognizes that it has to send the message also to v and v'. One of these 7 messages is an internal message. The same result holds for the case that u' has performed a split while u has not.

Case 4: If two leaves u, u' perform their splits 12 messages have to be sent. For each leaf there are the *SplitRequest*, *SplitGrant*, *SplitPerformed*, *FwSplitPerformed* and two Initialize messages. Only 3 of these 12 messages are internal messages of T_l.

Case 3 leads to the inequality $M_H(O) \leq 7M_l(O)$, but, since for every operation $o \in O$ case 2 occurs once, the inequality becomes $M_H(O) < 7M_l(O)$. □

Lemma 3.4 *If no server has had a crash failure, i.e., $F = \emptyset$, then at τ_0 the HAT H is in the following situation:*

1. *The trees T_l, T_r are isomorphic.*
2. *For every pair of nodes u, u', $I_u = I_{u'}$ holds.*
3. *For every pair of leaves u, u', $K_u = K_{u'}$ holds.*
4. *All tags have been deleted.*

Proof: Every insert operation is performed on both trees. Therefore, if one of the two roots has performed a split, the other has done it too. Furthermore, the splits have been done with the same split value. Induction shows the first three properties.

Let u, u' be a pair of leaves. The split of one node forces the other to split, too. The nodes u, u' perform their splits and send the messages *SplitPerformed* to its buddy, and they forward these messages to the split manager. The split manager has created the split tag $t(u, u')$, and after having received these *SplitPerformed* messages, $t(u, u')$ is deleted.

Let c be a client that sends an insert or search message to a pair of nodes u, u'. Both nodes create message tags, but these tags will be deleted after having received the second message from the buddy. If both nodes are leaves, then they send messages only back to c and no other tags are created. If both nodes have performed splits, then u, u' forward the messages to nodes v, v'. Each of these two nodes will receive messages from u, u' and one from its buddy. Therefore, the tags in v, v' will be deleted. If u has performed a split while u' has not, then u sends the message to nodes v, v' and u' sends a message back to c. The cross message $(Search(c, k, \mathrm{ID}), v, v')$ forces u' to send the message to v, v' too. Therefore, each of the nodes v, v' receives 3 messages and deletes the corresponding tag after that. □

The combination of both lemmas completes the proof of the theorem. □

It is obvious that H works well if no crash failure occurs: Every insertion and every search will be carried through, the trees tend to be isomorphic and every tag will be

deleted. Deletion of the tags cannot be guaranteed, if failures can occur. But it has to be guaranteed that a split tag is not deleted too early. Otherwise it could happen that the unique split value gets lost in a sequence of splits, crash failures and recoveries, i.e., u uses a split value different from that of u'. This would imply that the buddy condition is violated, and recovery is not possible.

Lemma 3.5 *Let u, u' be two nodes, where one of them has sent a split request to the split manager. After the corresponding split tag $t(u, u')$ is deleted, neither u nor u' will send a split request.*

Proof: Assume that the tag $t(u, u')$ is deleted. Each node must have forwarded the split performed message from its buddy. At the moment at which u has sent the message *FwSplitPerformed*(u'), it has been an internal node already. If u' sends a recover request to u, then u' will be recovered as an internal node. Therefore, no node will send a split request. □

Definition 3.6 (Recoverable crashes) *Let us call the set of crashed servers F* recoverable*, if for every node u, $S(u) \notin F$ or $S(u') \notin F$ hold.*

Note that in this very strict definition, we do not pay attention to the possibility that a crashed server could have performed its recovery before a second server crashes, without

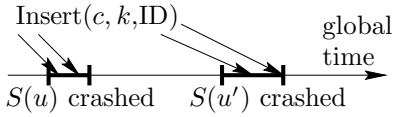

Figure 3.7: Unrecoverable situation

any loss of data or messages. But on the other hand, in contrast to what one might think at first glance, the restriction that two buddies u, u' are not crashed at the same time, is not strict enough. This is shown in the example in Figure 3.7. Messages are sent to a node u and its buddy u'. The messages get lost although the nodes u, u' are not crashed at the same time, because the messages reach crashed servers $S(u), S(u')$.

Let us only state the properties of the HAT under extreme circumstances; situations in between will lead to behaviour in between.

Theorem 3.7 (Correct operation) *If F is recoverable, then all of the following hold at τ_0:*

1. *Every search request has been answered.*
2. *Every insert request has been processed by at least one responsible leaf.*
3. *Let k be a key that has been inserted. If no more crash failure happens after τ_0 and a single client c starts a search for k, then c will be informed that k has been found.*

Proof: We can assume that there is no recovery, i.e., crashed servers remain crashed. Let u, u' be the addressees of a *Search*(c, k, ID) message. Since F is recoverable at least one of these nodes is operational and acts on the message: If it is a leaf, it sends a message to c, if it is an internal node it forwards *Search*(c, k, ID) to the children v, v'. Since O is

finite, the number of nodes in H is bounded and the message will reach a leaf eventually. This leaf sends a message back to the client c. The same holds for an $\mathit{Insert}(c, k, \mathrm{ID})$ message.

For every inserted key k there is a node u, with k has been inserted in u and $S(u) \notin F$. The $\mathit{Search}(c, k, \mathrm{ID})$ message will reach u eventually. If u is still a leaf, then u sends a response to c. If u has performed a split, then u forwards $\mathit{Search}(c, k, \mathrm{ID})$ to a child v. Eventually $\mathit{Search}(c, k, \mathrm{ID})$ reaches a leaf w. Since c has sent the $\mathit{Search}(c, k, \mathrm{ID})$ after τ_0, the key k is in w, i.e., $k \in K_w$. Then w sends a response to c. □

Theorem 3.8 (Trees are identical copies) *If F is recoverable and no more crash failure happens after τ_0, then H will eventually reach a state in which all of the following hold:*

1. *The trees T_l, T_r are isomorphic.*
2. *Both nodes in a pair of nodes u, u' has the same responsibility interval, i.e., $I_u = I_{u'}$.*
3. *Both nodes in a pair of leaves u, u' has the same data, i.e., $K_u = K_{u'}$.*

Proof: Let S be a server with $S \in F$. Due to the assumption that a server can have a crash failure only after the initialization of the first node, S had stored at least one node before the crash failure happened. Let u be such a node. Since F is recoverable, the server $S(u')$ had no crash failure. Therefore, u can determine its parent, children and sibling. This is enough information for S to reconstruct every node v with $S = S(v)$. Therefore, the trees become isomorphic.

Let u, u' be two nodes. If $S(u) \notin F$ and $S(u') \notin F$, then $I_u = I_{u'}$ due to construction. If one of these nodes is on a crashed server, e.g., $S(u) \in F$, then this node is recovered with the responsibility interval of its buddy. Therefore, the equality $I_u = I_{u'}$ holds in both cases.

Let w be a node with $S(w) \notin F$. If $k \in K_w$, every request $\mathit{Insert}(c, k, \mathrm{ID})$ has reached w or w has been initialized with $k \in K_w$. A leaf splits when its key set becomes too big. Therefore, if w is a leaf, its decision to split or not to split does not depend on the size of F. This shows that $K_u = K_{u'}$ for a pair of leaves with $S(u) \notin F$ and $S(u') \notin F$. If $S(u) \in F$ and $S(u') \notin F$, then the key sets are equal, since K_u is a copy of $K_{u'}$. □

Let us summarize: A HAT works well, if not too many servers are crashed at the same time and recovery is performed fast enough. And of course, if crashes are too frequent and recovery is too slow, no data structure whatsoever can offer a reliable service.

3.6 Improvements and Modifications

So far, we have described the HAT with very few assumptions and in its most simple form. We now discuss the flexibility of the basic HAT concept.

3.6.1 Secure Memory and Early Crashes

We have assumed that every server has a small secure memory. This is done to enable a server's recovery after a crash. But the server can obtain the information in a different way, too. Let S be a server that after having a crash failure wants to recover. If S receives

a message from a node u, then S knows that there should be a node $v \in S$ which is in some relation to u. Therefore S can send a message to u, asking for the information. Hence, an extra message exchange can replace the secure memory. The same technique can be applied to get rid of the assumption that no server crashes before it is involved in the data structure: If it instead does crash first, but later gets a message from some other server, it will know that it is involved, and it will get the necessary information with a message exchange.

3.6.2 Lazy Update

We described in Section 3.3 how clients with more activity tend to have better knowledge of the tree. Let us now explain the (fairly obvious) extension of the *lazy update concept* presented in [KW94] to the HAT. If a node u sends a message m to another node or to a client, u attaches its address and its responsibility interval to m. Every search or insert operation a client c initializes, gives information about the HAT's structure to c. If c knows the addresses of two nodes u, u' and their responsibility interval I_u, c can send every search for a key $k \in I_u$ to u, u' directly.

3.6.3 Hidden Data

In a distributed data structure, it is not always clear what the correct answer for a search request is. If one client searches for a key k and a second client wants to insert k into the structure, the answer to the search depends on factors such as the transmission time of the messages. This is a somewhat undesirable, but unavoidable property. Therefore in a HAT, a client can get two different answers for a search. If there is one positive answer, then the client knows that the key is present.

But a HAT has another strange property. It is possible that a key k is not found, although k has been inserted into a leaf earlier. This can happen if a leaf u has performed a split (and deleted its key set K_u), while the child has not received the *Initialize* message from u, i.e., the data is hidden in an unreachable message. In the following we describe a protocol that avoids this undesired and strange behaviour.

If a node u performs a split and sends *Initialize* messages to children v_i, v_i', then u does not delete its key set $K_u = K_1 \cup K_2$. Every following message $Search(c, k, \mathrm{ID})$ is forwarded according to the described routing scheme, but additionally u sends an answer to c. Also, every following message $Insert(c, k, \mathrm{ID})$ is forwarded according to the routing scheme, and k is inserted in K_u. Node u is called *schizophrenic*, since it acts on a leaf and an internal node at the same time.

If a node v receives an Initialize message, then it sends a message *InitializationOK()* back to the sender. If node u receives the message *InitializationOK()* from a node v_i or v_i', then it deletes K_i, i.e. $K_u \mapsto K_u \setminus K_i$. If $K_u = \emptyset$, then u stops its schizophrenic behavior.

It can be easily seen that if a key k has been inserted in a leaf u, then all following search requests will find at least one copy of k.

3.7 Conclusion

We have proposed a distributed dictionary that tolerates arbitrary single server crashes. The distinctive feature of our model is that the crash of a server cannot be detected. This is in contrast to all other proposals of distributed fault tolerant search structures

presented thus far. It reflects the real situation in a global data base more accurately, and is in general more suitable to complex overall conditions. This makes our solution fundamentally different from all previous ones, but also more complicated. We have presented in detail the algorithms for searching, insertion, and graceful recovery of crashed servers.

Chapter 4

Point Formation: Contraction Functions and Weber point

4.1 Introduction

The previous chapter showed that a weak environmental setting is enough to build and maintain a highly available data structure. This chapter does something similar with a different problem in the context of distributed coordination: It shows that with a simple robot behaviour, the point formation problem can be solved.

The point formation problem is as follows. There are mobile robots in the plane that should meet in one destination point. The robots cannot communicate, but they can take snapshots of the plane and thus determine the relative location of the other robots at a point of time. What is a strategy the robots should follow such that they can complete this task? The goal is to derive a distributed solution for this coordination problem, i.e., the destination point is not given by a central coordinator or a leader robot. Furthermore, the destination point should not rely on a coordinate system.

A distributed solution for the point formation problem is given in [SY99]. The authors consider the following model. It is assumed that the robots are moving points, two or more robots can occupy the same position simultaneously. Furthermore, there is discrete time. At every time step, every robot is either *active* or *inactive*. If a robot is active at a time step, it does the following. It takes a snapshot, computes the point it wants to move to and moves imeadiatly. An inactive robot does nothing. Furthermore, it is assumed that the robots have a common sense of orientation. With this model, [SY99] proved the following results: If the robots are *nonoblivious* (i.e., the robots keep every snapshot in their memory), then the point formation problem can be solved; if the robots are *oblivious* (i.e., the robots forget former snapshots), then for every set of initial positions of the robots, the point formation problem can be solved, except for the case of two robots. To prove these results, the authors have to argue on the behavior and the internal states of the robots.

Our goal is to solve the point formation problem with a simpler method and under weaker assumptions than [SY99]. A rough outline of our solution is as follows: Every robot takes a snapshot and detects the multi set of positions of the robots. This multi set is the input for a so called *contraction function*, and this function gives the destination point. Due to the definition of contraction functions, the destination point is independent of coordinate systems and invariant against straight line movement towards the destination point. All the robots have to do is to compute the destination point and move towards it. This solution works in an asynchronous model, where the robots take their

snapshots independently and there is no global clock. Moreover, we get simple and easy to understand proofs.

This chapter has the following structure. Section 4.2 gives the exact description of the model we consider. In section 4.3, contraction function are defined. Furthermore, this section presents contraction functions for small multi sets, gives some uniqueness results, and proves important properties. In section 4.4, the Weber point is introduced. It turns out that the Weber point can be used to get a contraction function for multi sets of arbitrary cardinality. Section 4.5 gives the definition of convex core and discusses the relation to contraction functions. Section 4.6 discusses the question whether the mapping to the Weber point is the only contraction function? In section 4.7, we present some model modifications. The chapter is concluded by section 4.8.

4.2 The Model

We consider the following model. There are n mobile robots in the plane. The robots are anonymous in the sense that they all execute the same algorithm and they are identical copies of each other. The robots have no access to a common coordinate system. There is no way for the robots to communicate.

A robot is able to take snapshots of the whole plane. With a snapshot, the robot is able to detect the other robots and their positions, and it can measure distances and angles. It is possible that any two robots use different unit lengths.

The robots are assumed to be moving points, and two or more robots can occupy the same position simultaneously. The robots are able to detect multiplicity, i.e., for every point in the plane, the robots can count the number of robots on this point.

Initially, all robots are sleeping. Every robot will awake eventually, independent of the others. Then, the robot can look around (i.e., it takes a snapshot), it computes the destination point and moves towards this. The robot can do the same several times, but we do not specify this. Furthermore, it is possible that the robot has breaks, i.e., for some time, it does nothing. Each robot has its personal speed, but it is assumed that every robot will reach a computed destination point, i.e., the robot needs finite time to travel a finite distance.

4.3 Contraction Functions and their properties

In this section, we give the definition of contraction functions and discuss their properties. Furthermore, we present examples of contraction functions for multi sets of small cardinality.

Definition 4.1 *Let $n \in \mathbb{N}$ be fixed, let C be a function that maps a multi set of n points to a point $c \in \mathbb{R}^2$. The function C is called a* Contraction Function*, if for all multi sets $X = \{p_1, \ldots, p_n\}$ the following properties are fulfilled:*

1. *There is no ordering among the robots, i.e., for every permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$,*
$$C(p_{\pi(1)}, \ldots, p_{\pi(n)}) = C(p_1, \ldots, p_n)$$
2. *The robots do not have a common coordinate system: For every isometric function $o : \mathbb{R}^2 \to \mathbb{R}^2$,*
$$C(o(p_1), \ldots, o(p_n)) = o(C(p_1, \ldots, p_n))$$

3. *Linear movement towards the contraction point does not change it: Let* $c := C(X)$*, for every vector* $t = (t_1, \ldots, t_n) \in [0,1]^n$*,*

$$C(\ldots, (1-t_i)p_i + t_i c, \ldots) = c$$

If $c = C(X)$ *for a contraction function* C*, then* c *is called a* Possible Contraction Point *or* Contraction Point *of* X*.*

Let $X = \{\ldots, p_i, \ldots\}$ *be a multi set. For a contraction function* C *and a vector* $t = (t_1, \ldots, t_n) \in [0,1]^n$*,* $c := C(X)$*, define the multi set* $X' := \{\ldots, (1-t_i)p_i + t_i c, \ldots\}$*.* X' *is called a* contraction *of* X*, and we say that* X *can be contracted to* X'*.*

Remark: Lemma 4.3 will show that a contraction point does not exist, if the multi set consists of collinear points without median. Therefore, the "all multi sets" in definition 4.1 means that we exclude these cases.

Property 1 says that there is no leader among the robots. Due to the definition of multi sets, this property is always fulfilled.

We start the discussion of the properties with contraction functions for small multi sets, the simplest (non empty) multi set of points consists of one point. For this case, we get the following result.

Lemma 4.2 *Let* $X = \{p\}$*, then* $c := p$ *is the only possible contraction point.*

Proof: It is obvious that c fulfills the properties of a contraction point.

Let o be a rotation by an arbitrary angle $\alpha \in]0^o, 360^o[$ about p. Since $o(p) = p$, for every contraction point c, $o(c) = c$ must hold. This can only be fulfilled with $c = p$. □

The next case is $X = \{p_1, p_2\}$. Two points are trivially collinear, and if $p_1 \neq p_2$, X has no median. The next lemma shows that for collinear points, the existence of a median is crucial.

Lemma 4.3 *Let* X *be a multi set with the property that all points of* X *are collinear, i.e., they are on a straight line* l*. Then there are two cases:*

1. X *has a median* q*. Then* q *is the only possible contraction point for* X*.*

2. X *has no median. Then there is no contraction point.*

Especially in the case $n = 2$*, there is a contraction point iff both points have the same position.*

Proof: Let M_l be the mirroring about the line l. Since every point $p_i \in X$ lies on l, $M_l(p_i) = p_i$ holds. This implies that for every contraction function C, $C(X) = C(M_l(X)) = M_l(C(X))$. Therefore, $C(X) \in l$.

If the median q exists, then a point $p_i \in X$ can move towards it, without changing the median property of q. This has been shown in the proof of Lemma 1.3. Therefore, the median is a possible contraction point.

To show the uniqueness, we choose an arbitrary vector v parallel to l with $|v| \geq |p_i - q|$ for all $p_i \in X$. If n is odd we define the multi set

$$X'(p) := \begin{cases} 1 & \text{if } p = q \\ \frac{n-1}{2} & \text{if } p = q - v \text{ or } p = q + v \\ 0 & \text{else} \end{cases}$$

If n is even, we define

$$X'(p) := \begin{cases} 2 & \text{if } p = q \\ \frac{n-2}{2} & \text{if } p = q - v \text{ or } p = q + v \\ 0 & \text{else} \end{cases}$$

Due to its definition, X' can be contracted to X. Since X' is symmetric by an 180^o–rotation about q, q is the median of X' and it is the only possible contraction point. Contraction of X' to X shows, that q is the only possible contraction point for X.

As to the next claim, assume that X has no median, this implies that n is even. Assume that the points are ordered linearly with respect to their indices. This implies that $p_{\frac{n}{2}} \neq p_{\frac{n}{2}+1}$ (otherwise, $p_{\frac{n}{2}}$ is a median). Assume for contradiction that there is a contraction function C with $c := C(X)$. The point c must be one of the points $p_{\frac{n}{2}}, p_{\frac{n}{2}+1}$ or must lie on the straight line segment between them (otherwise, $\frac{n}{2}$ points must pass through $p_{\frac{n}{2}}$ or $p_{\frac{n}{2}+1}$, such that this point becomes a median and the unique contraction point). For the multi set

$$X'(p) := \begin{cases} \frac{n}{2} & \text{if } p = p_{\frac{n}{2}} \\ \frac{n}{2} & \text{if } p = p_{\frac{n}{2}+1} \\ 0 & \text{else} \end{cases}$$

we obtain that $C(X) = C(X') = c$. Let $c' := \frac{1}{2}(p_{\frac{n}{2}} + p_{\frac{n}{2}+1})$, rotation by 180^o about c' transforms X' into itself. Therefore, $c = c'$. But since c' is not invariant under movement towards it, X' has no contraction point. This contradicts the assumption. □

Remark: In contrast to [SY99], contraction functions do not solve the problem for an even number of collinear points without median. But the problem can be solved with a little extension and one assumption from [SY99]: Initially, the positions of the robots are different. If the number of robots is bigger than 2, then the robots inside move to the middle point defined by the two outer points, and the two robots outside do not move until the multi set of positions got a median. The only not solvable instance remains the case of two different points, but this case cannot be solved for oblivious robots in [SY99], too.

The previous lemma showed that there is no contraction function for 2 different points, and there is no contraction function for 4 collinear points without median. However, similar to the case of one point, there is a unique contraction point for 3 points.

Theorem 4.4 *For $n \in \{3, 4\}$, there is a unique contraction point (except in the case of 4 collinear points without median).*

Proof: We discuss 4 cases. For every case, we present a solution and show uniqueness. Uniqueness is shown as follows: First, we prove uniqueness for symmetric multi sets. Second, the symmetric multi sets are contracted to arbitrary multi sets.

Case 3.a $n = 3$ and the three points form a triangle in which every angle is smaller than 120^o. Then there is a unique point s, the so called *Steiner point*, with the property, that the angle between every pair of two vectors $(p_i - s), (p_j - s)$, $i \neq j$, is equal to 120^o. The Steiner point s is a possible contraction point (Existence and construction of the Steiner point will be discussed after this proof).

To show uniqueness, we consider an equilateral triangle. Due to the rotation symmetry, the Steiner point is the only possible contraction point. Since a contraction function must

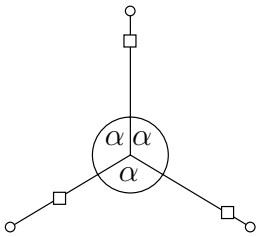

Figure 4.1: An arbitrary triangle □ derived from an equilateral triangle o, $\alpha = 120^o$.

be invariant under linear movement and each triangle with property 3.a can be obtained from an equilateral triangle (see Figure 4.1) the result holds for every of these triangles.

Case 3.b $n = 3$ and the three points form a triangle in which one angle is equal to or bigger than 120^o. Then the point at the biggest angle is a possible contraction point.

It is assumed that the triangle is isosceles, we use the notation from Figure 4.2. Due to symmetry every contraction point must lie on the straight line which goes through p_1 and is perpendicular to the straight line through the other points. Assume for contradiction,

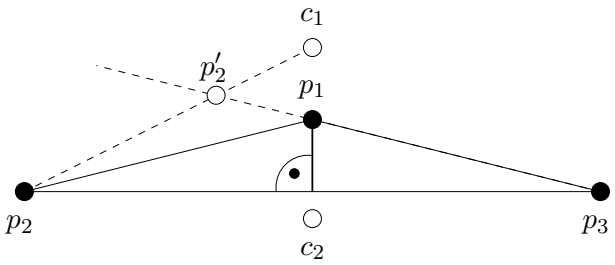

Figure 4.2: Triangle with one angle bigger than 120^o.

that there is a contraction point c above p_1, e.g., $c = c_1$. When p_2 moves to c_1, it has to cross the straight line through p_1, p_3. For the multi set $\{p_1, p_2', p_3\}$, the point p_1 is the only possible contraction point. Therefore, there is no contraction point above p_1. Assume for contradiction that there is a contraction point c below p_1, e.g., $c = c_2$. For the multi set $\{p_1, c_2, p_3\}$, the Steiner point is the only possible contraction point (case 3.a). In this situation, every contraction point lies inside the triangle and cannot be equal to c. This is a contradiction. Therefore $c := p_1$ is the only possible contraction point for an isosceles triangle. Since every triangle can be obtained from an isosceles triangle by movement towards p_1, $c := p_1$ is the only solution for every case.

Case 4.a $n = 4$ and the four points form a convex quadrangle. In this case, the intersection of the diagonals is a possible contraction point.

If the four points form a rectangle the intersection of the diagonals is the only possible contraction point. Every arbitrary convex quadrangle can be obtained from a rectangle by movement towards the diagonals. Therefore, the intersection of the diagonals is the only possible solution.

Case 4.b $n = 4$ and three points form a triangle with the fourth point inside. Then the point inside is a possible contraction point.

Figure 4.3 shows 3 different points c_1, c_2, c_3 which might be possible solutions. Assume for contradiction, that there is a contraction point $c \in \{c_1, c_2, c_3\}$. Case $c = c_1$, then in order to move to c_1, p_2 has to cross the straight line through p_1, p_4. Case $c = c_2$, p_3 has to cross the line through p_2, p_4. Case $c = c_3$, p_3 has to cross the line through p_2, p_4. In

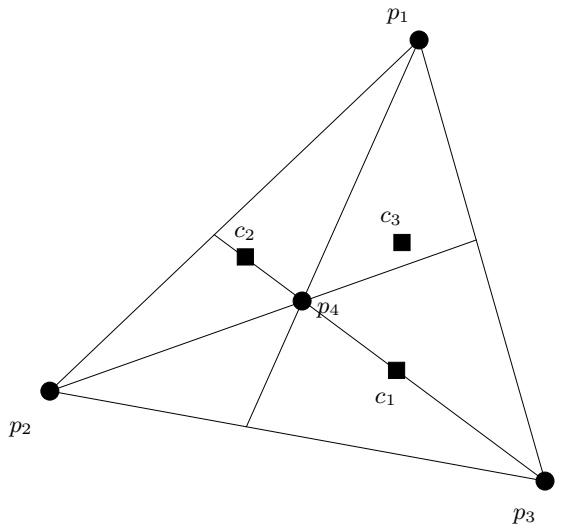

Figure 4.3: 4 points in non convex position.

each case, there are 3 points on a line, which means that every contraction point must lie on that line. But this is a contradiction. □

Construction of the Steiner point

Theorem 4.4 stated that the Steiner point can be used as a contraction point. To show the existence and to illustrate its properties, we present two standard methods to construct the Steiner point. These methods have been presented in many publications, see for example [KM97] for more details. The Steiner point has a very long history. We omit the discussion about this and refer to [KM97].

Let p_1, p_2, p_3 be 3 points that form a triangle such that every angle is less than 120^o. We do the following construction, see Figure 4.4. For every given point p_i, we define a point p_i' such that p_i', p_l, p_k form a equilateral triangle, where p_l, p_k are the two other points different from p_i. After that, we draw straight lines from p_i to p_i'. The three lines intersect in one point, this is the Steiner point s.

Figure 4.5 shows a similar way to construct the Steiner point. The point p_1' is defined as before, let $m := \frac{1}{3}(p_2 + p_3 + p_1')$ be the middle point of the equilateral triangle p_2, p_3, p_1'. We draw a circle around m with radius $|p_2 - m|$. There are two intersections of this circle with the straight line through p_1, p_1'. One of these intersection points is p_1' itself, the other point is the Steiner point s. Instead of using one line and one circle, one can use two circles (or three) circles, too.

Further Properties of Contraction Functions

We presented some existence and uniqueness results for small multi sets of cardinality less than 5. For better understanding of contraction functions, we give characterizations for multi sets of arbitrary cardinality. The first result can be regarded as a *majority lemma*.

Lemma 4.5 *If there is a* $q \in \mathbb{R}^2$ *with* $X(q) = k > \frac{n}{2}$*, then* q *is the only possible contraction point.*

Proof: The property that more than half of the points have position q is invariant under linear movement towards q. Therefore q is a possible contraction point. Let C be

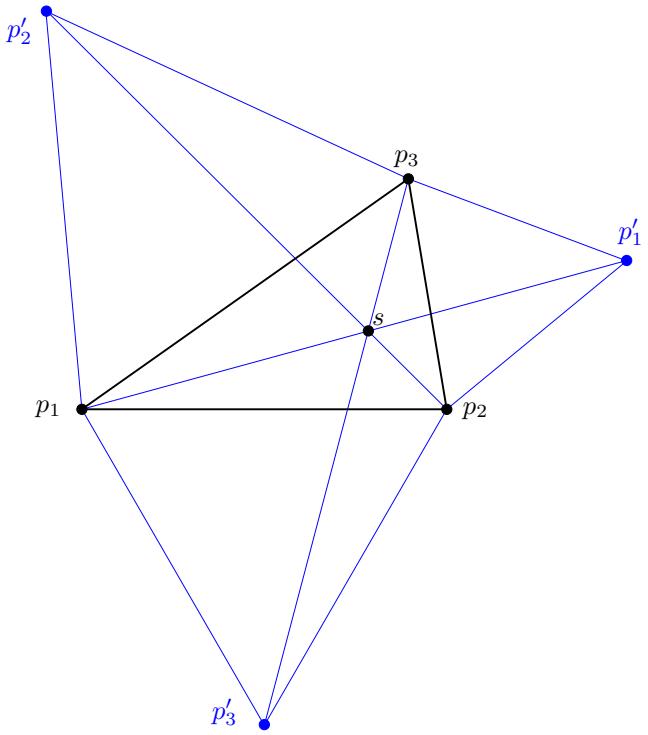

Figure 4.4: Construction of the Steiner point.

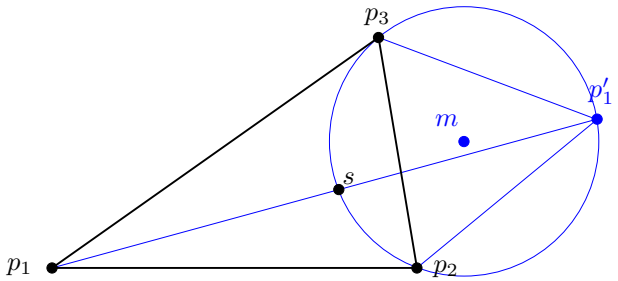

Figure 4.5: Construction of the Steiner point.

a contraction function and $c := C(X)$, then

$$\begin{aligned} c &= C(\underbrace{q,\ldots,q}_{k},p_{k+1},\ldots,p_n) = C(\underbrace{q,\ldots,q}_{k},\underbrace{c,\ldots,c}_{n-k}) = C(\underbrace{c,\ldots,c}_{k},\underbrace{q,\ldots,q}_{n-k}) \\ &= q \end{aligned}$$

□

Due to their definition, contraction functions fulfill the contraction property, i.e., if the point move towards the destination point, the destination point does not change. Contraction functions fulfill an extension property in the following sense as well. This result is interesting for its own; moreover, it is helpful to simplify some proofs.

Lemma 4.6 *Let C be a contraction function and $c := C(p_1,\ldots,p_n)$. For a $t = (t_1,\ldots,t_n) \in [-1,\infty)^n$ define $p_i' := p_i + t_i(p_i - c)$. It holds*

$$C(p_1',\ldots,p_n') = c$$

Proof: For $t \in [-1,0]^n$, the claim is the contraction property. To show the claim for $t \in [0,\infty)^n$, we show that the lemma is true for $t := (1,\ldots,1)$, i.e., the distance from c is doubled. Iteration and the contraction property show the result for arbitrary $t \in [0,\infty)^n$.

Define $p_i' := p_i + (p_i - c)$ and for $c' := C(p_1',\ldots,p_n')$, compute the vector $v := (c' - c)$, see Figure 4.6. For the points $p_i'' := \frac{1}{2}(p_i' + c')$, we obtain $p_i'' - p_i = \frac{1}{2}(c' - c)$. This leads to

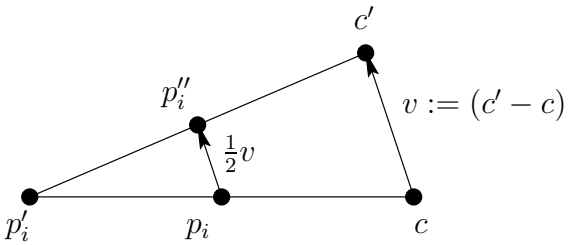

Figure 4.6: Extension property.

$$\begin{aligned} c' &= C(p_1',\ldots,p_n') = C(p_1'',\ldots,p_n'') = C(\ldots,p_i + \frac{1}{2}(c' - c),\ldots) \\ &= C(p_1,\ldots,p_n) + \frac{1}{2}(c' - c) = c + \frac{1}{2}(c' - c) \end{aligned}$$

Since $c' = c + (c' - c)$, this implies $c' - c = 0$. □

Given a multi set X of points, we want to specify the possible contraction points for X. For some special cases, we derived uniqueness results for the contraction points. For other cases, we do not have uniqueness results, but the positions of contraction points can be restricted as well. The following theorem shows that the set of contraction points is a subset of the convex hull.

Theorem 4.7 *If c is a contraction point for a multi set X, then c lies in the convex hull of X.*

Proof: Assume for contradiction that c is outside the convex hull of X. The convex hull $CH(X)$ is convex and bounded by a polygon. In order to move to c the points have to

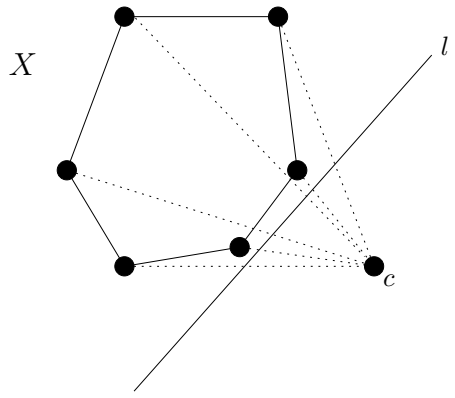

Figure 4.7: Contraction point c outside the convex hull

cross a straight line l with the property $c \notin l$, see Figure 4.7. The line l can be chosen as

the straight line defined by the corresponding line segment of the boundary. Therefore, it is possible, that all points are on l. But due to Lemma 4.3 this is either not solvable or c must lie on l: Contradiction! □

4.4 Weber Point

The previous section presented contraction functions only for small multi sets. In order to get contraction functions for multi sets of arbitrary cardinality, we look at the Weber point. This point is defined in the following way.

Definition 4.8 *Let $X = \{p_1, \ldots, p_n\}$ be a multi set of points. The* Weber point $W(X)$ *of X is defined as the point, that minimizes the function*

$$z \mapsto f(z) := \sum_{p_i \in X} |p_i - z| \quad z \in \mathbb{R}^2$$

The Weber point has a long history, and since many mathematicians worked on it, it has many different names, too, e.g., Fermat–Torricelli, Fermat–Weber. For simplicity, we avoid the discussion of this and refer to [Dör58, Wes93, Dre95, KM97, DH02]. The Weber point is named after Alfred Weber, who did some work on facility location [Web09].

If X consists of one point p_1, then p_1 is the Weber point. For a multi set of two different points, then every point on the straight line segment connecting p_1 and p_2 is a Weber point. In this case, the Weber point is not unique. To characterize such cases, we cite the following lemma.

Lemma 4.9 ([KM97]) *The Weber point is unique, except for the case of an even number of collinear points without median.*

Our interests in Weber points are based on the fundamental result that shows a strong connection between contraction functions and Weber point.

Theorem 4.10 *If the Weber Point is unique, then it is a possible contraction point.*

Proof: The only thing we have to show, is that the Weber point is invariant against linear movement towards it. It is obvious that the Weber point fulfills the other properties. Let $w := W(X)$ be the Weber point of $X = \{p_1, \ldots, p_n\}$, for $t \in [0, 1]$, define $p' := (1-t)p_1 + tw$. Due definition,

$$\sum_{i=1}^{n} |p_i - w| < \sum_{i=1}^{n} |p_i - z| \quad \forall z \in \mathbb{R}^2 \setminus \{w\}$$

holds. This and the inequality $|z - p_1| \leq |z - p'| + |p' - p_1|$ lead to

$$\begin{aligned}
\sum_{i=2}^{n} |p_i - w| + |p' - w| &< \sum_{i=2}^{n} |p_i - z| + |p_1 - z| + \underbrace{|p' - w| - |p_1 - w|}_{=-|p'-p_1|} \\
&\leq \sum_{i=2}^{n} |p_i - z| + |p' - z| + |p' - p_1| - |p' - p_1| \\
&= \sum_{i=2}^{n} |p_i - z| + |p' - z|
\end{aligned}$$

This means that w is the unique Weber point of the multi set $(X \uplus \{p'\}) \setminus \{p_1\}$. □

Theorem 4.10 leads directly to the following corollary.

Corollary 4.11 *In Theorem 4.4 it has been shown that for the cases $n \in \{3, 4\}$ there is exactly one possible contraction point. Since every Weber point is a contraction point, in Theorem 4.4 a characterization of the Weber point has been given.*

Furthermore, we obtain a new proof for the following well–known fact.

Corollary 4.12 ([KM97]) *It holds: $W(X) \in CH(X)$.*

Proof: If the Weber point is not unique then the corollary is true. If the Weber point is unique, then $w := W(X)$ is a contraction point. And since every contraction point lies in the convex hull (Theorem 4.7), w is in the convex hull, too. □

The Weber point has the following properties. A similar version for sets was given in [Wei37], we generalize this result to multi sets. The result will play an important role in our study and will be exploited extensively.

Theorem 4.13 *Let $w = W(X)$ be the unique Weber point of X, let $X(w) \in \mathbb{N}$ be the multiplicity of w in X. It holds:*

$$\left| \sum_{\substack{i=1 \\ p_i \neq w}}^{n} \frac{p_i - w}{|p_i - w|} \right| \leq X(w)$$

And w is the only point with this property. Especially, if $w \notin X$, i.e. $X(w) = 0$, we obtain

$$\sum_{i=1}^{n} \frac{p_i - w}{|p_i - w|} = 0$$

Proof: [follows [KM97]] Define $g(z) := |p - z|$ for an arbitrary point $p \in \mathbb{R}^2$. If $z \neq p$, we compute the gradient as

$$\mathrm{grad} g(z) = \frac{p_i - z}{|p_i - z|}$$

Let v be a unit vector. If $z = p$, we obtain for the directional derivative in direction v

$$g_v(z) = 1$$

If $w \notin X$, the function f is differentiable in w. Since w minimizes the sum of the distances, it holds

$$0 = \mathrm{grad} f(w) = \mathrm{grad} \left(\sum_{i=1}^{n} |p_i - w| \right) = \sum_{i=1}^{n} \frac{p_i - w}{|p_i - w|}$$

If $w \in X$, we have to be more carefully. Let $k := X(w)$. For the function

$$\tilde{f}(z) := \sum_{\substack{i=1 \\ p_i \neq w}}^{n} |p_i - z| = f(z) - k|w - z|$$

and the directional derivative in direction v we obtain

$$f_v(w) = \left\langle \mathrm{grad} \tilde{f}(w), v \right\rangle + k = \left\langle \sum_{\substack{i=1 \\ p_i \neq w}}^{n} \frac{p_i - w}{|p_i - w|}, v \right\rangle + k$$

Since f is minimized in w, $f_v(w) \geq 0$ holds. The theorem is true, if $\mathrm{grad}\tilde{f}(w) = 0$. Therefore, we can assume that $\mathrm{grad}\tilde{f}(w) \neq 0$, and we define $v := -\frac{\mathrm{grad}\tilde{f}(w)}{|\mathrm{grad}\tilde{f}(w)|}$. This leads to

$$0 \leq f_v(w) = -\left|\sum_{\substack{i=1 \\ p_i \neq w}}^{n} \frac{p_i - w}{|p_i - w|}\right| + k$$

On the other hand, assume that for $\hat{w} \in \mathbb{R}^2$,

$$\left|\sum_{\substack{i=1 \\ p_i \neq \hat{w}}}^{n} \frac{p_i - \hat{w}}{|p_i - \hat{w}|}\right| \leq X(\hat{w})$$

holds. For the function

$$\hat{f}(z) := \sum_{\substack{i=1 \\ p_i \neq \hat{w}}}^{n} |p_i - z| = f(z) - X(\hat{w})|\hat{w} - z|$$

we obtain that

$$0 \leq -\left\langle \mathrm{grad}\hat{f}(\hat{w}), \frac{\mathrm{grad}\hat{f}(\hat{w})}{|\mathrm{grad}\hat{f}(\hat{w})|} \right\rangle + X(\hat{w})$$

Since for all $v \in S^1$,

$$\left\langle \mathrm{grad}\tilde{f}(\hat{w}), \frac{\mathrm{grad}\hat{f}(\hat{w})}{|\mathrm{grad}\hat{f}(\hat{w})|} \right\rangle \geq \left\langle \mathrm{grad}\tilde{f}(\hat{w}), v \right\rangle$$

holds, it follows that $f_v(\hat{w}) \geq 0$. This means that f is minimized in $\hat{w}$, and since w is unique, $\hat{w} = w$. □

The inequality described in theorem 4.13 shows a strong connection to the median of numbers: Let $\{r_1, r_2, \ldots, r_n\}$ be a set of real numbers. If n is odd, then this set has a unique median r and

$$\left|\sum_{\substack{i=1 \\ r_i \neq r}}^{n} \frac{r_i - r}{|r_i - r|}\right| \leq 1$$

holds. This is the reason why the Weber point is sometimes called *spatial median*.

Furthermore, the theorem shows important properties of the Weber point: For a given point $q \in \mathbb{R}^2$, it can be tested if $q = W(X)$. Especially, it is easy to check whether $W(X) \in X$ or not.

Furthermore the lemma can be used to prove Theorem 4.10. For $p_i \neq w$ and $t \in [0, 1)$, $p' := (1 - t)p_i + tw$, we obtain

$$\frac{p_i - w}{|p_i - w|} = \frac{p' - w}{|p' - w|}$$

However, we have to pay attention to the cases in which the multiplicity of the Weber point changes.

4.4.1 Construction of the Weber point

Since for $n \in \{2, 3, 4\}$ there is at most one contraction function, the Weber point can be constructed very easily in these cases. It is remarkable that the Weber point is a solution which is minimal in the sense that it minimizes the path length the points have to move.

However, the Weber Point is not constructible by compass and ruler in general, this has been shown in [Baj88], [CM69]. Constructing the Weber point is equivalent to computation of the roots of a high degree polynomial. Even for $n = 5$, one can find examples for which the Weber point cannot be constructed. In [CM69] the following simple example is given, see Figure 4.8: Due to the symmetry with respect to the x–axis, the Weber point

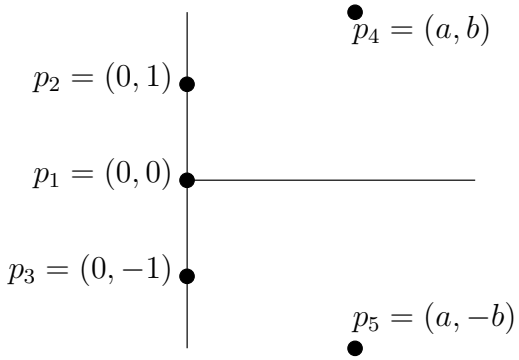

Figure 4.8: 5 points for which the Weber point is not constructible.

w of these five points has the coordinates $(x, 0)$, where $x \in [0, a]$. The value of x has to be chosen such that

$$\sum_{i=1}^{5} |p_i - w| = x + 2\sqrt{1 + x^2} + 2\sqrt{b^2 + (a - x)^2}$$

is minimized. This leads to a polynomial of degree 8, which cannot be solved for all a, b.

On the other hand, this is no problem for an analog computer, Figure 4.9 shows a so called *Varignon Frame* [Wes93, KM97]. A horizontal board is drilled with holes

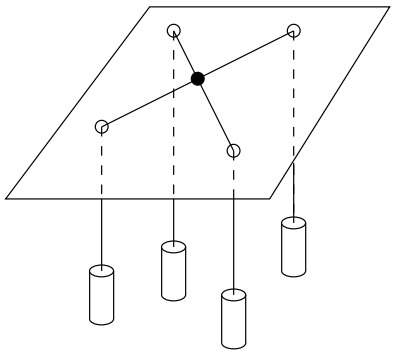

Figure 4.9: Varignon Frame, an analog computer.

corresponding to the positions of the robots, i.e., for every $p \in \text{uniq}(X)$ there is one hole h_p. Furthermore, for every $p \in \text{uniq}(X)$ there is one string s_p, and those strings are tied together in a knot at one end. The loose end of the string s_p is passed from above through the hole h_p. Then the loose end of s_p is attached to physical weights of unit mass. The number of weights attached to a string s_p is equal to $X(p)$. If we regard this as an ideal

physical system without friction, then the knot moves to a position that is the Weber point. This is due to the fact that the sum of the forces corresponds to the sum of unit vectors. It is hard to compute the Weber point algorithmically, but it is easy to do it with this frame.

The hardness of such a simple example is interesting because if we replace the intuitive notion of *distance* by the less intuitive notion of *squared distance*, it becomes easy to minimize the resulting function:

$$z \mapsto \sum_{p_i \in X} (p_i - z)^2 \text{ is minimized in } z := \frac{1}{n} \sum_{p_i \in X} p_i$$

Furthermore, this function has a unique minimum for all instances of X. But unfortunately, it is not invariant under movement.

On the other hand, it can be tested easily, whether one of the given points is the Weber point itself. Therefore, one can ask for the probability that for a random multi set X, $W(X) \in X$ holds. Does the probability grow, if n becomes bigger?

The following Lemma is an example how to compute such a probability.

Lemma 4.14 *Let $X = \{p_1, p_2, p_3\}$ a set of random points, where the points are chosen independently, and equally distributed from the one–dimensional sphere $S^1 := \{x \in \mathbb{R}^2 || x| = 1\}$. It holds that $\Pr[W(X) \in X] = \frac{1}{3}$.*

If the points are chosen from the unit disk $D := \{x \in \mathbb{R}^2 || x| \leq 1\}$, we get an upper bound for the probability $\Pr[W(X) \in X] \leq c \approx 0.391$.

Proof: In Figure 4.10, the point p_3 is the Weber point of X, iff $\alpha \geq 120^o$. Let o be the

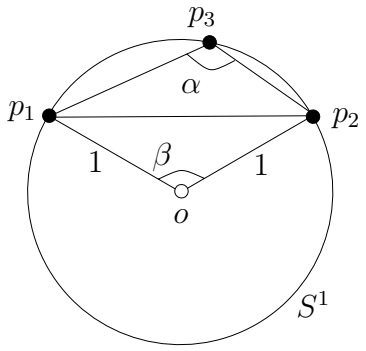

Figure 4.10: 3 Points on S^1

origin, let β be the angle between the vectors $p_1 - o, p_2 - o$. Basic geometry says that $2\alpha + \beta = 2\pi$; and therefore, $\alpha \geq 120^o$, iff $\beta \leq 120^o = \frac{2}{3}\pi =: \beta_0$. This means that one of the given points is the Weber point, iff for the spherical distance d between every pair of points holds:

$$d(p_i, p_j) \leq \beta_0 \qquad \forall i, j \in \{1, 2, 3\}$$

Assume that p_1, p_2 are fixed. Define $\beta := d(p_1, p_2)$. If $\beta \leq \beta_0$, then p_3 is the Weber point with probability $\Pr[p_3 = W(X)] = \frac{\beta}{2\pi}$ and $\Pr[p_1 = W(X)] = \Pr[p_2 = W(X)] = \frac{\beta_0 - \beta}{2\pi}$. Therefore, the following holds

$$\Pr[W(X) \in X \,|\, \beta] = \frac{\beta}{2\pi} + 2\frac{\beta_0 - \beta}{2\pi} = \frac{1}{2\pi}(2\beta_0 - \beta)$$

To compute the probability, we have to integrate this expression.

$$\begin{aligned} Pr[W(X) \in X] &= 2\frac{1}{2\pi}\int_0^{\beta_0} \frac{1}{2\pi}(2\beta_0 - \beta)d\beta = \frac{1}{2\pi^2}(2\beta_0^2 - \frac{\beta^2}{2}\Big|_0^{\beta_0}) \\ &= \frac{1}{2\pi^2}(2\beta_0^2 - \frac{1}{2}\beta_0^2) = \frac{1}{3\pi^2}\beta_0^2 = \frac{1}{3} \end{aligned}$$

As to the second claim, let $p_1, p_2, p_3 \in D$. Since $\Pr[p_i = p_j] = 0$ for $i \neq j$, $\Pr[W(X) \in X] = 3\Pr[W(X) = p_1]$ holds. Let $A(p_2, p_3) \subset \mathbb{R}^2$ be the set such that for all $p \in \mathbb{R}^2$, $p = W(p, p_2, p_3)$ iff $p \in A(p_2, p_3)$ holds. For the size of $A(p_2, p_3)$, we compute

$$|A(p_2, p_3)| = 2d^2(p_2, p_3)\left(\frac{\pi}{9} - \frac{1}{4\sqrt{3}}\right),$$

where $d(p_2, p_3)$ denotes the distance between p_2, p_3. With this, we obtain

$$\Pr[W(X) = p_1 \,|\, p_2, p_3] = \frac{|A(p_2, p_3) \cap D|}{\pi} \leq \frac{|A(p_2, p_3)|}{\pi}$$

Since $\int_D \int_D d^2(p_2, p_3)dp_3\, dp_2 = \pi^2$, we can compute the probability

$$\Pr[W(X) = p_1] \leq \frac{2}{\pi}\left(\frac{\pi}{9} - \frac{1}{4\sqrt{3}}\right) \underbrace{\frac{1}{\pi^2}\int_D \int_D d^2(p_2, p_3)dp_3\, dp_2}_{=1}$$

With this, we conclude that $\Pr[W(X) \in X] \leq c := 3\frac{2}{\pi}(\frac{\pi}{9} - \frac{1}{4\sqrt{3}}) \approx 0.391$. □

If the number of random points grows, one expects that the probability that one point is near by the Weber point of the other points grows. Therefore the next Lemma is not very surprising.

Lemma 4.15 *Let $D := \{x \in \mathbb{R}^2 \,|\, |x| \leq 1\}$ be the unit disk. Let $(p_i)_{1 \leq i \leq n}$ be a sequence of independent and uniformly distributed random variables in D. Let $r := \min\{|p_i| \,|\, i \in \{1, \ldots, n\}\}$ be the minimal distance from a point to the origin. Then it holds*

$$E[r] = \frac{4^n(n!)^2}{(2n+1)!}$$

Which implies

$$E[r] \sim \frac{\sqrt{\pi}}{2}\frac{1}{\sqrt{n}} \stackrel{n \to \infty}{\longrightarrow} 0$$

Proof: Since $\Pr[|p_i| \leq a] = a^2$ for a given $a \in [0, 1]$ we obtain

$$\begin{aligned} \Pr[r \leq a] &= 1 - \Pr[r > a] = 1 - \prod_{i=1}^{n} \Pr[|p_i| > a] \\ &= 1 - \prod_{i=1}^{n}(1 - \Pr[|p_i| \leq a]) = 1 - (1 - a^2)^n \end{aligned}$$

This leads to the density function $\psi(a) = 2an(1 - a^2)^{n-1}$.

$$E[r] = \int_0^1 a\, d\psi(a) = 2n\int_0^1 a^2(1 - a^2)^{n-1}\, da$$

$$
\begin{aligned}
&= \frac{2^2\, n(n-1)}{3} \int_0^1 a^4 (1-a^2)^{n-2}\, da \text{ (partial integration)} \\
&= \frac{2^k}{\prod_{i=1}^k (2i-1)} \frac{n!}{(n-k)!} \int_0^1 a^{2k} (1-a^2)^{n-k}\, da \text{ (iteration)} \\
&= \frac{2^n}{\prod_{i=1}^n (2i-1)}\, n!\, \frac{1}{2n+1} = \frac{2^n\, n!}{2^n\, n!} \frac{2^n\, n!}{\prod_{i=1}^{n+1}(2i-1)} \\
&= \frac{4^n (n!)^2}{(2n+1)!}
\end{aligned}
$$

The *Stirling formula* $n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$ leads to

$$
\begin{aligned}
\frac{4^n (n!)^2}{(2n+1)!} &\sim \frac{4^n\, 2\pi n \left(\frac{n}{e}\right)^{2n}}{\sqrt{2\pi}\sqrt{2n+1}\left(\frac{2n+1}{e}\right)^{2n+1}} = \frac{\sqrt{2\pi}}{\sqrt{2n+1}} \frac{n}{2n+1}\, e \underbrace{\left(\frac{2n}{2n+1}\right)^{2n}}_{\to e^{-1}} \\
&\sim \frac{\sqrt{\pi}}{2} \frac{1}{\sqrt{n}}
\end{aligned}
$$

□

Although the expected distance to the Weber point becomes smaller and smaller, the probability that one of the given points is the Weber point decreases.

Theorem 4.16 *Let $D := \{x \in \mathbb{R}^2 \,|\, |x| \leq 1\}$ be the unit disk. Let $(p_i)_{i\geq 1}$ be a sequence of independent and uniformly distributed random variables in D, let $X_n := \{p_1, \ldots, p_n\}$ be a multi set. Then the following holds:*

$$\lim_{n\to\infty} \Pr[W(X_n) \in X_n] = 0$$

Furthermore, there exists a sequence $(P_n)_n$ with $\Pr[W(X_n) \in X_n] \sim P_n$ and

$$0.62\frac{1}{n} \approx \frac{\pi^2}{16}\frac{1}{n} \leq P_n \leq \left(\frac{\pi^3}{16\pi - 32}\right)^2 \frac{1}{n} \approx 2.88\frac{1}{n}$$

Proof: The proof is done in two steps. First, we use Monte Carlo integration ([Zwi92]) to transform the sum of distances into an integral. In the second step, we derive upper and lower bounds for this integral.

Since $\Pr[p_i = p_j] = 0$ for $i \neq j$, it can be assumed that X_n is a set for all n. For a $p_i \in X_n$ we obtain

$$p_i = W(X_n) \Leftrightarrow \left| \sum_{\substack{j=1 \\ j\neq i}}^n \frac{p_j - p_i}{|p_j - p_i|} \right| \leq 1$$

Since $\Pr[p_i = W(X_n) \wedge p_j = W(X_n)] = 0$ for $i \neq j$, it follows that

$$\Pr[W(X_n) \in X_n] = \sum_{i=1}^n \Pr[p_i = W(X_n)] = n \Pr[p_1 = W(X_n)]$$

Monte Carlo integration shows that

$$\frac{\pi}{n} \sum_{j=2}^n \frac{p_j - p_1}{|p_j - p_1|} \longrightarrow \int_D \frac{z - p_1}{|z - p_1|} dz \quad \text{for } n \to \infty$$

Therefore, for n big enough,

$$\Pr\left[\frac{\pi}{n}\left|\sum_{j=2}^{n}\frac{p_j-p_1}{|p_j-p_1|}\right|\le\frac{\pi}{n}\right]\approx\Pr\left[\left|\int_D\frac{z-p_1}{|z-p_1|}dz\right|\le\frac{\pi}{n}\right]$$

holds, Let $\delta := |p_1|$ be the distance between p_1 and the origin of D. The probability on the right side depends only on δ. Let $p_\delta := (-\delta, 0)$ be a point on the x–axis, we define

$$A(\delta) := \left|\int_D\frac{z-p_\delta}{|z-p_\delta|}dz\right|$$

Due to symmetry it is easy to see that the second coordinate does not influence $A(\delta)$, i.e., for $z = (x_z, y_z)$, we obtain

$$A(\delta) = \left|\int_D\frac{x_z+\delta}{|z-p_\delta|}dz\right| = \int_D\frac{x_z+\delta}{|z-p_\delta|}dz$$

Let $D_r := \{z \in D \,|\, |z-p_\delta| \le r\}$ be the disk around p_δ with radius r. For $r \in [0, 1-\delta]$, it holds

$$\int_{D_r}\frac{x_z+\delta}{|z-p_\delta|}dz = 0$$

which implies

$$A(\delta) = \int_{D\setminus D_{1-\delta}}\frac{x_z+\delta}{|z-p_\delta|}dz$$

For $r \in [1-\delta, 1+\delta]$ fixed, let α be the angle between the x–axis and a point, where $\partial D = S^1$ and ∂D_δ intersects (seen from p_δ). We compute

$$\alpha = \arccos\left(\frac{1-\delta^2-r^2}{2\delta r}\right)$$

Using polar coordinate, we obtain

$$\begin{aligned}
A(\delta) &= \int_{1-\delta}^{1+\delta}\int_{-\alpha}^{\alpha}\frac{r\cos(\varphi)}{r}r\,d\varphi\,dr = \int_{1-\delta}^{1+\delta}\int_{-\alpha}^{\alpha}r\cos(\varphi)\,d\varphi\,dr \\
&= 2\int_{1-\delta}^{1+\delta}r\int_0^{\alpha}\cos(\varphi)\,d\varphi\,dr = 2\int_{1-\delta}^{1+\delta}r\left(\sin(\varphi)\Big|_0^{\alpha}\right)dr \\
&= 2\int_{1-\delta}^{1+\delta}r\sin\left(\arccos\left(\frac{1-\delta^2-r^2}{2\delta r}\right)\right)dr
\end{aligned}$$

Since $\sin(x) \le 1$ for all $x \in \mathbb{R}$ we obtain an upper bound for A.

$$A(\delta) \le 2\int_{1-\delta}^{1+\delta}r\,dr = 4\delta$$

To obtain a lower bound, we regard the function $f(r) := \arccos(\frac{1-\delta^2-r^2}{2\delta r})$. To approximate f we define the functions

$$\begin{aligned}
f_1(r) &:= (r-1+\delta)\frac{\pi}{2(\sqrt{1-\delta^2}-1+\delta)} \quad \text{for } r \in [1-\delta, \sqrt{1-\delta^2}] \\
f_2(r) &:= (r-\sqrt{1-\delta^2})\frac{\pi}{2}(1+\delta-\sqrt{1-\delta^2})+\frac{\pi}{2} \quad \text{for } r \in (\sqrt{1-\delta^2}, 1+\delta]
\end{aligned}$$

Since

$$\begin{aligned} 0 &\leq f_1(r) \leq f(r) \leq \frac{\pi}{2} \qquad r \in [1-\delta, \sqrt{1-\delta^2}] \\ \frac{\pi}{2} &\leq f(r) \leq f_2(r) \leq \pi \qquad r \in (\sqrt{1-\delta^2}, 1+\delta] \end{aligned}$$

we obtain that $\sin(f(r)) \geq sin(f_i(r))$. Therefore

$$\begin{aligned} A(\delta) &\geq \int_{1-\delta}^{\sqrt{1-\delta^2}} 2r\sin(f_1(r))\,dr + \int_{\sqrt{1-\delta^2}}^{1+\delta} 2r\sin(f_2(r))\,dr \\ &= \frac{8\delta}{\pi^2}(4\sqrt{1-\delta^2} - 4 + 2\pi - \sqrt{1-\delta^2}\pi) \\ &= \frac{8\delta}{\pi^2}(\underbrace{(4-\pi)}_{>0}\underbrace{\sqrt{1-\delta^2}}_{\geq 0} + 2\pi - 4) \geq \frac{8\delta}{\pi^2}(2\pi - 4) = \frac{16\pi - 32}{\pi^2}\delta \end{aligned}$$

The two bounds for A lead to bounds for $P_n := n\left(A^{-1}\left(\frac{\pi}{n}\right)\right)^2$. We compute

$$\begin{aligned} 4\delta &\geq A(\delta) \\ A^{-1}(4\delta) &\geq \delta \\ A^{-1}\left(\frac{\pi}{n}\right) &\geq \frac{\pi}{4n} \\ P_n = n\left(A^{-1}\left(\frac{\pi}{n}\right)\right)^2 &\geq \frac{\pi^2}{16n} \approx 0.62\frac{1}{n} \end{aligned}$$

In a similar way we compute an upper bound:

$$P_n \leq \left(\frac{\pi^3}{16\pi - 32}\right)^2 \frac{1}{n} \approx 2.88\frac{1}{n}$$

□

Remark: We found several better bounds for $A(\delta)$. Since they are hard to invert, we described the linear functions only. In Figure 4.11 we compare experimental results (a) for the probability with the lower (b) and upper (c) bounds derived in Theorem 4.16. The experimental values were obtained by 1000000 experiments for every $n \in \{3, \ldots, 17\}$.

To get the lower and upper bounds, we transformed the sum of unit vectors into an integral. For large n, this method gives good bounds. However, for $n = 3$, the lower bound is bigger than the experimental value. This is due to the fact, that we compare two different but related things: The sum of unit vectors is small, if the points are almost collinear; the integral is small, if one point is close to the origin. The experimental results show that for $n > 3$, the theoretical bounds are quite good.

In [DSL92], a similar result has been presented and proved by the central limit theorem. We think that our result is still interesting, because our proof uses Monte Carlo integration in a non standard way. Normally, Monte Carlo integration is used for numerical integration of a function, the integral is transformed into a sum. We did the reverse thing, a sum was transformed into an integral.

Weber–Test Algorithm
It takes not more than $O(n^2)$ time to check whether a multi set of points contains its

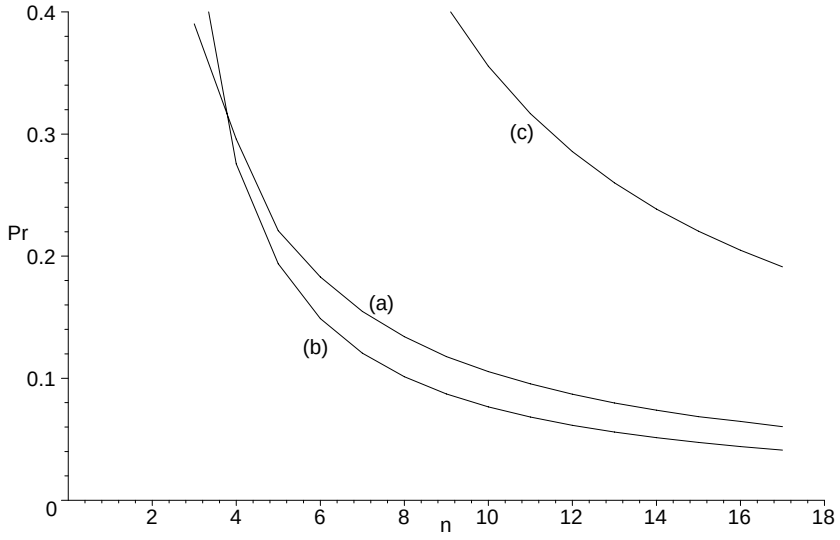

Figure 4.11: Experimental results (a) compared to theoretical bounds (b),(c)

Weber point: For every point p_i, compute the vector $v_i := \sum_{j=1, p_j \neq p_i}^{n} \frac{p_j - p_i}{|p_j - p_i|}$ and test whether $|v_i| \leq X(p_i)$ holds (where $X(p_i)$ is the multiplicity of p_i in X). We call this method the *Weber–Test algorithm*.

Theorem 4.13 showed that the Weber point can be regarded as a "spatial median". A simple method to compute the median of n numbers is to sort the numbers in $O(n \log n)$ time and pick the median. However, with an improved algorithm, the median can be computed in linear time [BFP+73]. This leads to the question, whether we can do something similar with the Weber point. In the following, we improve the Weber–Test algorithm. In

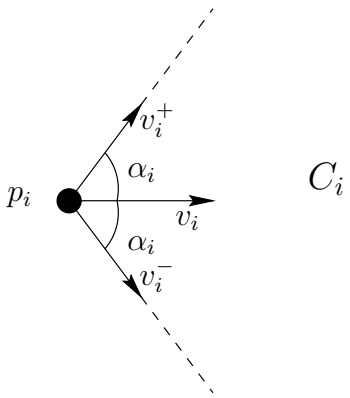

Figure 4.12: Some notation

the improved Weber–Test algorithm, we use the notation indicated in Figure 4.12. For a point $p_i \in X$ with $v_i \neq 0$, define $\alpha_i := \arccos\left(\frac{1}{|v_i|} X(p_i)\right)$. Let v_i^+, v_i^- be two different unit vectors such that the angles between v_i, v_i^+ and v_i, v_i^- are α_i. The vectors v_i^+, v_i^- span an open cone $C_i := \{x \in \mathbb{R}^2 \setminus \{p_i\} | \text{angle}(v_i, x - p_i) < \alpha_i\}$. With $\mathcal{C}$ we denote a set of possible candidates.

Fast–Weber–Test Algorithm
$\mathcal{C} := X$
While $\mathcal{C} \neq \emptyset$

Choose a $p_i \in \mathcal{C}$.
If $|v_i| \leq X(p_i)$
  return p_i.
else
  $\mathcal{C} = \mathcal{C} \cap C_i$
return FALSE.

Lemma 4.17 *The Fast–Weber–Test algorithm runs in $O(n^2)$ time. The Fast–Weber–Test algorithm returns the Weber point, iff $W(X) \in X$.*

Proof: Since $p_i \notin C_i$, after having chosen p_i, it is removed from $\mathcal{C}$. For every p_i, computing the vector v_i and updating $\mathcal{C}$ takes $O(n)$ time. Since $p_i \notin C_i$, it is removed from $\mathcal{C}$.

We have to show that the updating of $\mathcal{C}$ is not too restrictive, i.e., if $W(X) \in X$, then $W(X) \in \mathcal{C}$. Let $p_i \in X$ be a point with $v_i \neq 0$. The function $f(z) = \sum_{i=1}^{n} |p_i - z|$ is strictly convex, and therefore, the set $\{z \in \mathbb{R}^2 \mid f(z) < f(p_i)\}$ is convex. f is not differentiable in p_i, but for the directional derivative, we obtain

$$\begin{aligned} \frac{\partial}{\partial v_i^+} f(p_i) &= \frac{\partial}{\partial v_i^-} f(p_i) = X(p_i) + \left\langle -v_i, v_i^+ \right\rangle = X(p_i) - |v_i| \cos(\alpha_i) \\ &= X(p_i) - |v_i| \left(\frac{1}{|v_i|} X(p_i) \right) = 0 \end{aligned}$$

This implies $\{z \in \mathbb{R}^2 \mid f(z) < f(p_i)\} \subset C_i$, and therefore, $W(X) \in C_i$. □

If we regard the Fast–Weber–Test algorithm as randomized, we can think about the expected run time. If the points are collinear, the problem becomes finding the median of n numbers. The Fast–Weber–Test algorithm finds that median in expected time $O(n \log n)$. However, this does not hold for arbitrary multi sets. For $X_n := \{\exp(\frac{2\pi i}{n}) \mid i \in \{1, \ldots, n\}\}$, i.e., the points are uniformly placed on the sphere S^1, the Fast–Weber–Test algorithm takes $\Theta(n^2)$ time. This shows that the run time depends on the geometry of the points, not only on the order the algorithm chooses them.

The algorithm can be improved, using the fact that the distance between a point p_i and the Weber point w cannot be too big. Since $|p_i - w| \leq |p_i - p_j| + |p_j - w|$, we compute

$$|p_i - w| \leq \frac{1}{n} \sum_{j=1}^{n} |p_i - p_j| + |p_j - w| \leq \frac{2}{n} \sum_{j=1}^{n} |p_i - p_j|$$

Unfortunately, this improvement does not help in the worst case example mentioned above.

The result about the angle between v_i and $w - p_i$ can be exploited in a different context. We know that this angle is less than $\arccos\left(\frac{X(p_i)}{|v_i|}\right)$, i.e.,

$$\arccos \left\langle \frac{v_i}{|v_i|}, \frac{w - p_i}{|w - p_i|} \right\rangle < \arccos \left(\frac{X(p_i)}{|v_i|} \right)$$

This leads to

$$|v_i| \geq \left\langle v_i, \underbrace{\frac{w - p_i}{|w - p_i|}}_{\in S^1} \right\rangle > X(p_i)$$

This means that the length of the orthogonal projection of v_i onto the vector $w - p_i$ is bigger than the multiplicity of p_i in X. This improves the inequality $|v_i| > X(p_i)$ for a point $p_i \neq w$ (theorem 4.13).

4.4.2 Approximation

For $w := W(X) \notin X$, it is hard to construct w. Since the distance function is convex, it is easy to approximate the Weber point. For the sake of completeness and to show how the Weber point can be approximated, we present Weiszfeld's algorithm [Wei37]. The property $\sum_{p \in X} \frac{p-w}{|p-w|} = 0$ leads to the following approximation scheme: Choose a point $w^{(0)} \in \mathbb{R}^2$, then do the iteration

$$w^{(k+1)} := \frac{1}{\sum_{i=1}^{n} \frac{1}{|p_i - w^{(k)}|}} \quad \sum_{i=1}^{n} \frac{p_i}{|p_i - w^{(k)}|}$$

This algorithm has been discussed, criticized and improved in many articles [Kuh73, Ost78, CT89, Bri95, VZ01].

4.4.3 Things that are not true

The Weber point has a very intuitive definition. However, some of its properties are not intuitive. Therefore, intuition can lead to wrong results. In order to illustrate this and to get a better understanding of these properties, we present 4 wrong statements.

Wrong 4.18 *Let g be a line, let h_1, h_2 the two half-planes defined by g. If there are more points of X in h_1 than in h_2, then $W(X) \in h_1$.*

Figure 4.13 gives an example that this statement is wrong. For the point p_1, $p_1 =$

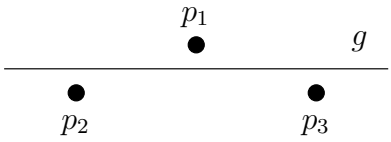

Figure 4.13: Example for Wrong 4.18

$W(\{p_1, p_2, p_3\})$ holds. But p_1 is the only point in its half-plane, and there are two other points in the other half-plane.

The first statement dealt only with points. We have seen that for a point p_i, the vector

$$v_i := \sum_{\substack{j=1 \\ p_j \neq p_i}}^{n} \frac{p_j - p_i}{|p_j - p_i|}$$

is related to the negative gradient of the sum of distances. In the Fast–Weber–Test algorithm, we used this to improve the Weber–Test algorithm. This naturally leads to the question, whether this relation can be exploited in other contexts as well.

Wrong 4.19 *Let g be the line through two points p_i, p_j, let h_1, h_2 the two half-planes defined by g. If $p_i + v_i, p_j + v_j \in h_1$, then $W(X) \in h_1$.*

The wrongness of this statement can be seen by the Example 4.14. The points p_1, p_2, p_4 are collinear, p_1, p_3, p_5 are collinear, too. This leads to:

$$\begin{aligned} v_2 &= \frac{p_3 - p_2}{|p_3 - p_2|} + \frac{p_5 - p_2}{|p_5 - p_2|} \\ v_3 &= \frac{p_2 - p_3}{|p_2 - p_3|} + \frac{p_4 - p_3}{|p_4 - p_3|} \end{aligned}$$

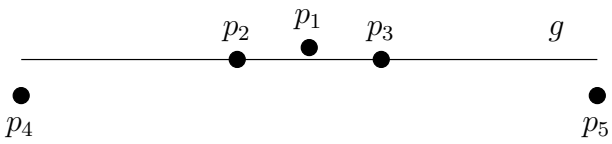

Figure 4.14: Example for Wrong 4.19

Therefore, the points $p_2 + v_2, p_3 + v_3$ are in the lower half-plane, while the Weber point p_1 is in the upper half-plane.

Wrong 4.20 *There is a constant $c < 90^o$ such that for every X and for every $p_i \in X$ with $p_i \neq W(X)$, the angle between the vectors $w - p_i$ and v_i is not bigger than c.*

We want to illustrate this with the example in Figure 4.15. For $k > 1$, consider the multi

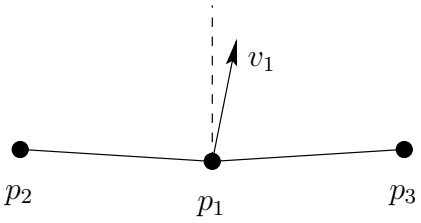

Figure 4.15: Angle between v_i and $w - p_i$ can be big.

set X with $X(p_1) = 1$, $X(p_2) = k - 1$, and $X(p_3) = k$. The angle between the vectors $p_2 - p_1, p_3 - p_1$ is close to but less than 180^o. Since the points are not collinear, p_3 is the Weber point. But for big k, the angle between v_1 and $p_3 - p_1$ is close to 90^o. If there is one additional point at p_2, then this angle becomes 0. This example shows that the Weber point is, in some sense, not a continuous function. If there is one additional point at p_2, then p_1 becomes the Weber point of the new multi set or the angle becomes 0. One additional point changes much.

Let p_1, p_2, p_3 be 3 points that form a triangle, let $p_4 \in \mathbb{R}^2$ be an arbitrary point. We know that $p_4 = W(p_1, p_2, p_3, p_4)$ holds, iff p_4 is inside this triangle. This motivates the following question: Given a multi set X, compute the set $S(X)$ such that $q \in S(X)$ iff $q = W(X \uplus \{q\})$. For a multi set X, $S(X)$ is called the *Weber set*. The Weber set is not empty, because $W(X) \in S(X)$ holds.

Wrong 4.21 *The Weber set is convex.*

Figure 4.16 gives an example for the wrongness of the statement. There are four points in convex position. The Weber point is on the intersection of the diagonals, the non convex Weber set is marked grey.

4.5 Convex Core

We mentioned that a contraction point must be in the convex hull. In this section we show that for an even number of points, this result can be improved. To be more precise, we define a so called *convex core*. The convex core is a subset of the convex hull. To our

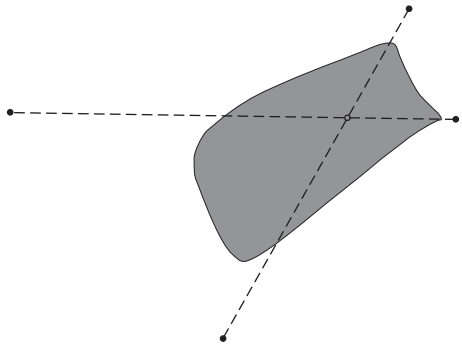

Figure 4.16: Weber set can be non convex.

knowledge, the convex core has not been investigated in the literature. We show that for an even number of points, every contraction point is inside the convex core. This is an improvement of the result, that every contraction point is in the convex hull (theorem 4.7). Since the Weber Point is a contraction point, the Weber point is in the convex core, too. This improves the result that the Weber point is in the convex hull (corollary 4.12). To our knowledge, this has not been shown before.

Definition 4.22 *For a point $p \in X$, the set $K_p \subset \mathbb{R}^2$ is defined as the convex hull of $X \setminus \{p\}$, i.e. $K_p := CH(X \setminus \{p\})$. The* Convex Core *of X is defined as*

$$CC(X) := \bigcap_{p \in X} K_p$$

Example: For $X = \{p_1, p_2\}$ we obtain for the convex core

$$\mathrm{CC}(X) = \begin{cases} \emptyset & \text{if } p_1 \neq p_2 \\ p_1 & \text{if } p_1 = p_2 \end{cases}$$

If $|X| = 4$ then $\mathrm{CC}(X)$ consists of a single point, or, if the points are collinear, of the line segment connecting the two middle points.

Figure 4.17 shows two examples. The convex cores are marked grey.

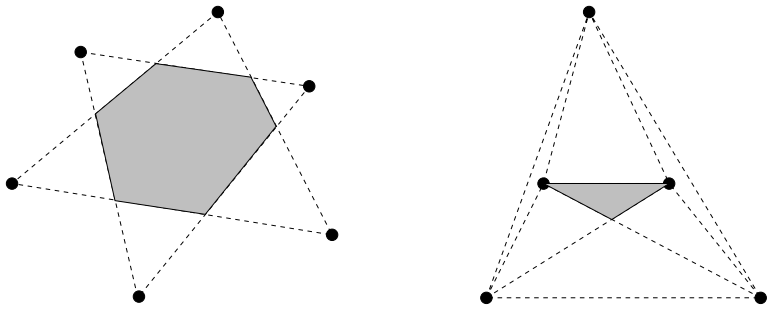

Figure 4.17: Examples for convex cores

The following lemma shows basic properties of the convex core. For a multi set X, the convex hull $\mathrm{CH}(X)$ can be described by the multi set of boundary points $\partial X \subset X$.

A point $p \in X$ belongs to the boundary ∂X, if p is on the polygon that bounds $\mathrm{CH}(X)$. Note that, in ∂X, points can occur with multiplicity bigger than 1. For instance, if X consists of collinear points, $\partial X = X$ holds. With X°, we denote the multi set of inner points, i.e., $X^\circ := X \setminus \partial X$.

Lemma 4.23 *The convex core $CC(X)$ has the following properties*

- *$CC(X)$ is a convex set.*
- *The convex core is monotone, i.e., for two multi sets X, X' with $X \subset X'$, $CC(X) \subset CC(X')$ holds.*
- *If $p \in CC(X)$, then $CC(X \uplus \{p\}) = CC(X)$.*
- *For $|X| \geq 4$, $CC(X) \neq \emptyset$ holds.*
- *The convex core $CC(X)$ is a superset of $CH(CC(\partial X) \uplus X^\circ)$.*
- *The convex core $CC(X)$ only depends on $\partial X, \partial X^\circ$, i.e., $CC(X) = CC(\partial X \uplus \partial X^\circ)$.*

Proof: The convex core is defined as the intersection of convex sets. Since the intersection of convex sets is convex, too, $\mathrm{CC}(X)$ is convex.

To show monotonicity, it is enough to show it for the case $|X'| = |X| + 1$. For $X = \{p_1, \ldots, p_n\}$ and $X' = X \uplus \{p\}$, we obtain

$$\begin{aligned} \mathrm{CC}(X') &= \bigcap_{i=1}^{n} \mathrm{CH}\left((X \uplus \{p\}) \setminus \{p_i\}\right) \cap \mathrm{CH}(X) \\ &\supset \bigcap_{i=1}^{n} \mathrm{CH}(X \setminus \{p_i\}) = \mathrm{CC}(X) \end{aligned}$$

Let $p \in \mathrm{CC}(X)$, for this we compute

$$\begin{aligned} \mathrm{CC}(X \uplus \{p\}) &= \bigcap_{i=1}^{n} \mathrm{CH}((X \uplus \{p\}) \setminus \{p_i\}) \cap \mathrm{CH}(X) \\ &\subset \bigcap_{i=1}^{n} \mathrm{CH}((X \uplus \{p\}) \setminus \{p_i\}) = \bigcap_{i=1}^{n} \mathrm{CH}(X \setminus \{p_i\}) \\ &= \mathrm{CC}(X) \end{aligned}$$

For every X with $|X| = 4$, $\mathrm{CC}(X) \neq \emptyset$ holds. Due to monotonicity, this implies that $\mathrm{CC}(X) \neq \emptyset$ for $|X| \geq 4$.

For $p_i \in X^\circ$ and for $p_j \in X$, $p_i \in K_{p_j}$ holds. Therefore, $p_i \in \mathrm{CC}(X)$ and $\mathrm{CH}(X^\circ) \subset \mathrm{CC}(X)$. Since $\partial X \subset X$, $\mathrm{CC}(\partial X) \subset \mathrm{CC}(X)$ (monotonicity). Furthermore, $\mathrm{CC}(X)$ is convex, $CH(\mathrm{CC}(\partial X) \uplus X^\circ) \subset \mathrm{CC}(X)$.

Let $p_i \in X \setminus \partial X \setminus \partial X^\circ$. For $p_j \neq p_i$, $\mathrm{CH}(X \setminus \{p_j\}) = \mathrm{CH}(X \setminus \{p_j, p_i\})$ holds. This leads to $\mathrm{CC}(X) = \mathrm{CC}(X \setminus \{p_i\})$. □

In order to prove the main result of this section, we need the following lemma.

Lemma 4.24 *Let $n \geq 4$ be even, let $X = \{p_1, \ldots, p_n\}$ such that $p_1, \ldots, p_{n-1}$ are collinear, i.e. they lie on a straight line l, and $p_n \notin l$. Then the median of $p_1, \ldots p_{n-1}$ is a contraction point and it is unique.*

Proof: To simplify notation let the p_1 be the median of $p_1, \dots p_{n-1}$, wlog. $p_1 = 0$. Let $q \in \{p_2, \dots p_{n-1}\}$ be a point with the greatest distance to p_1. Define the multi set

$$X' := \{-p_n, \underbrace{-q, \dots, -q}_{\frac{n}{2}-1}, \underbrace{q, \dots, q}_{\frac{n}{2}-1}, p_n\}$$

or

$$X'(p) = \begin{cases} 1 & \text{if } p = p_n \text{ or } p = -p_n \\ \frac{n}{2} - 1 & \text{if } p = q \text{ or } p = -q \\ 0 & \text{else} \end{cases}$$

in which the number of points on position $-q$ is equal to the number of points on q, see Figure 4.18 for an example. Due to rotation symmetry, for X' and a contraction function

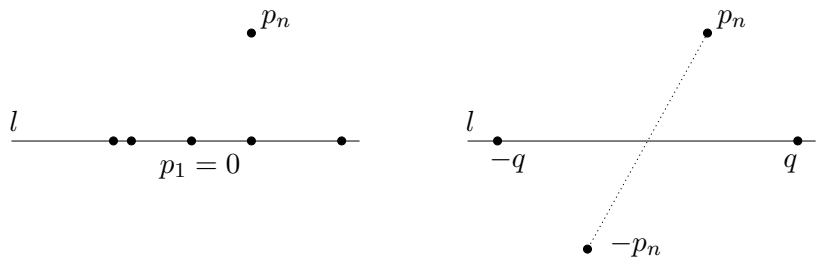

Figure 4.18: Example for multi sets X, X'

$C, p_1 = C(-p_n, -q, \dots, -q, q, \dots, q, p_n)$ holds. Then, the contraction property of C leads to

$$p_1 = C(-p_n, -q, \dots, -q, q, \dots, q, p_n) = C(p_1, p_2, \dots, p_{n-1}, p_n)$$

□

This uniqueness result helps to prove the main result of this section. The following theorem improves theorem 4.7 and corollary 4.12). Moreover, the notion of contraction functions gives a short and elegant proof for this result.

Theorem 4.25 *Let $|X|$ be even. Then every contraction point for X lies in the convex core $CC(X)$. Especially, the Weber point $W(X)$ is in the convex core as well.*

Proof: The theorem is true for $n = 2$: Either there is no contraction point or the convex core consists of the contraction point only. Therefore, it can be assumed, that $n \geq 4$.

Let c be a contraction point. Assume for contradiction that $c \in CH(X) \setminus \mathrm{CC}(X)$. Then there is a $p_i \in X$ with

$$c \notin CH(X \setminus \{p_i\})$$

The set $CH(X \setminus \{p_i\})$ is compact, convex and bounded by a polygon. Therefore, there is a straight line l between c and $CH(X \setminus \{p_i\})$. In order to move to c, $n-1$ points have to cross the line l. But this is the situation from Lemma 4.24, which says that c is on l. This contradicts the assumption and proves the theorem. □

This theorem can be used to "reprove" known results.

Lemma 4.26 (part of lemma 4.3) *If $X = \{p_1, p_2\}$ consists of two different points, then there is no contraction point.*

New Proof: For the convex core, $CC(X) = \{p_1\} \cap \{p_2\} = \emptyset$ holds. Since every contraction point must lie in the convex core, there is no contraction point. □

Lemma 4.27 (part of theorem 4.4) *Let $X = \{p_1, p_2, p_3, p_4\}$ such that these four points are not collinear. Then the possible contraction point mentioned in theorem 4.4 is unique.*

New Proof: If the points are not collinear, the convex core $CC(X)$ consists of one single point. □

It is possible that the convex core of a multi set is equal to the convex hull. If this is the case, theorem 4.25 does not help to restrict the position of possible contraction points. For a special case, the theorem 4.25 can be improved.

Theorem 4.28 *Let $|X|$ be even. For every $k \in \mathbb{N}$, $k \geq 1$, the property*

$$C(kX) \in CC(X)$$

holds.

Proof: Lemma 4.24 remains true for the weighted case. Therefore, the property holds. □

Remark: A multi set X and its convex core $CC(X)$ are objects of two different types. The multi set consists of a finite number of points, the convex core is a compact region bounded by a polygon. This is the reason why the procedure can not be iterated.

The definition of the convex core can be used to get an algorithm that computes the convex core: Let X be a multi set and let $n := |X|$. For every point $p \in X$, compute K_p in $O(n \log n)$ time (see [PS85]). Then compute the intersection of these n polygons. It takes not less than linear time to intersect two polygons, therefore, the algorithm takes not less than quadratic time. However, we can do better.

Theorem 4.29 *Let X be a multi set, and let $n := |X|$. The convex core of X can be computed in $\Theta(n \log n)$ time.*

Proof: To show the upper bound, we present an algorithm that computes the convex core in $O(n \log n)$ time. We think of $\partial X, \partial X^\circ$ as circularly ordered lists.

First, compute the convex hulls $\mathrm{CH}(X)$ and $\mathrm{CH}(X \setminus \partial X)$ in $O(n \log n)$ time. Let $\partial X = [p_1, p_2, \ldots, p_m]$ and let $\partial X^\circ = [q_1, \ldots q_r]$ the ordered lists of points. Define $K := \partial X$.

Second, for every $p_i \in \{p_1, \ldots, p_m\}$, do the following (start with $i = 1$ and go incrementally to $i = m$). Detect the points $q_i^1, \ldots q_i^\nu \in \partial X^\circ$ that are lying in the (possible degenerated) triangle spanned by p_{i-1}, p_i, p_{i+1} [1], this can be done in $O(\nu)$ time since ∂X° is a circularly ordered list. We detect the convex hull H_i of $\{p_{i-1}, q_i^1, \ldots q_i^\nu, p_{i+1}\}$ in $O(\nu)$ time in the following way, see Figure 4.19. For every point q_i^j, compute the angle between the two vectors $(q_i^j - p_{i-1}), (p_{i+1} - p_{i-1})$. Let q_i^- the first point that maximizes this angle. For every point q_i^j, compute the angle between the two vectors $(q_i^j - p_{i+1}), (p_{i-1} - p_{i+1})$. Let q_i^+ the last point that maximizes this angle. The ordered list $[p_{i-1}, q_i^-, \ldots, q_i^+, p_{i+1}]$ corresponds to the boundary polygon of H_i. Let D_i be a description of $H_i \cap K$. Then, p_i

[1] We omit the discussion of $p_{m+1} = p_1$ and $p_0 = p_n$

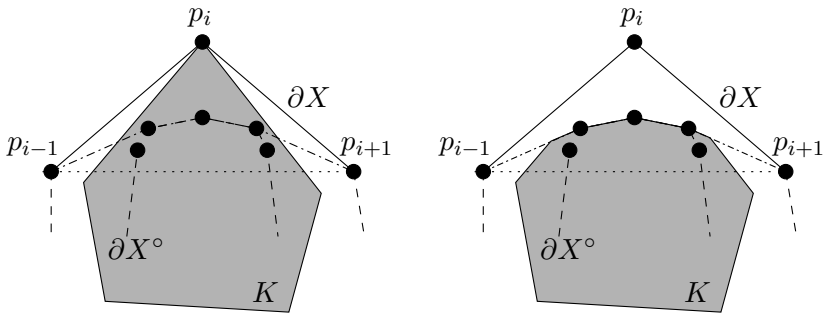

Figure 4.19: Algorithm to compute CC

is removed from K, D_i is inserted in K. After the second step is done for every point p_i, $K = CC(X)$ holds.

To get the lower bound $\Omega(n \log n)$, we consider the multi set $X' := 2X$. For X', $\mathrm{CC}(X') = \mathrm{CH}(X') = \mathrm{CH}(X)$ holds. Since we need $\Omega(n \log n)$ time to compute $\mathrm{CH}(X)$, the same holds for $\mathrm{CC}(X')$, too.

□

4.6 Is $c = w$?

In the previous sections, we showed that there are several similarities between contraction functions and Weber points.

- A contraction function exists, iff the Weber point is unique (lemmas 4.3,4.9).
- The Weber point is a contraction point (theorem 4.10).
- For $|X| = 3, 4$, the Weber point is the unique contraction point (lemma 4.4, theorem 4.10).

This leads to the question, whether the Weber point is always the only contraction point.

To be a possible contraction point, it is not enough that one point has a special position. This has been shown for the case of three points. If the three points form a triangle, and the biggest angle of this triangle is between 90^o and 120^o, none of the three points is a contraction point, although one is characterized uniquely. We consider an extended example in Figure 4.20. There are five points $\{p_1, p_2, p_3, p_4, p_5\}$, and these points are on two different half lines that start in p_1. We call this situation *A pair of scissors*. Assume for contradiction that p_1 is a possible contraction point. If the other points move towards p_1, it is possible that the set $\{p_1, p'_2, p'_3, p'_4, p'_5\}$ is obtained. But for this situation, the Weber point w of $\{p_1, p'_2, p'_3, p'_4, p'_5\}$ is the only possible contraction point. The reason for this is that the angle α between the two lines is too small; if the angle is bigger than $\frac{360^o}{5} = 72^o$, this construction is not possible. On the other hand, p_1 is the Weber point, if $\cos\alpha \leq -\frac{7}{8}$ ($\arccos(-\frac{7}{8}) \approx 150.04^o$) holds.

Such a pair of scissors can be defined for more points. Let $n = 2k+1$ ($k \geq 1$); n points are given, such that $k+1$ points are on one half line starting at p_1, and $k+1$ points are on a second half line starting at p_1, too. Let α be the angle between the two half lines. If $\alpha < \frac{360^o}{n}$ and the multiplicity of p_1 is 1, then p_1 cannot be a contraction point. In this

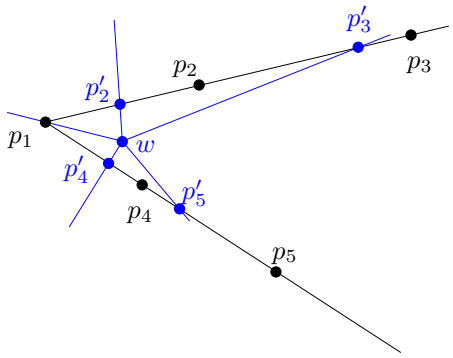

Figure 4.20: A pair of scissors.

case, a construction similar to Figure 4.20 can be done. The point p_1 is the Weber point, if $\alpha \geq \arccos(\frac{1}{2k^2} - 1)$.

For the special case $n = 3$ ($k = 1$), we get the result that p_1 is a contraction point, iff $\alpha \geq 120^o$. In this case, the bounds are tight. For $n > 3$, there is a gap between the two bounds.

A similar approach to show that a point cannot be a contraction point is shown in Figure 4.21. We are given 4 points $X = \{p_1, p_2, p_3, p_4\}$ and we assume that there is a contraction function C with $C(X) = c$. Since $c \notin X$, there are 4 rays that start at c and

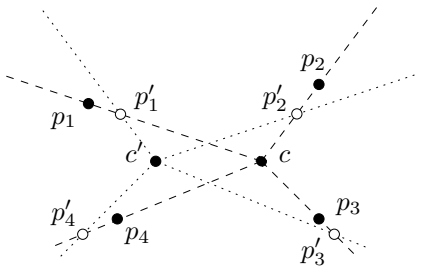

Figure 4.21: A copy of rays.

go through the 4 points. In the picture, those rays are drawn dashed. A copy of this arrangement of rays is mirrored and translated (drawn as dotted lines). For every dashed ray, there is a dotted ray that intersects the dashed one, and vice versa. This defines a matching. Define the points p'_i as intersections of matchings lines. Due to the contraction property, $C(p'_i) = c$ holds. And due to the isometric property, $C(p'_i) = c'$ holds. This is a contradiction, since $c \neq c'$.

For four points, this is not a surprising result since we know already the uniqueness in this case. However, we can generalize this case to the following theorem.

Theorem 4.30 *Let $X = \{p_1, \ldots, p_n\}$ be a multi set of points, let $c \notin X$. Let r_i be the ray starting at c and going through p_i. If there is an isometric function $o : \mathbb{R}^2 \to \mathbb{R}^2$ with the property that there is a matching between the rays $\{r_i | i\}$ and $\{o(r_i) | i\}$ and $c \neq o(c)$, then c is not a possible contraction point.*

We can use this to re-derive a well known result (part of theorem 4.4).

Lemma 4.31 *Let X be a multi set with 4 points. Let $W(X) \notin X$ and let $c \neq W(X)$ be an arbitrary point with $c \notin X$. There is a matching copy of the rays $r_i := [c, p_i]$. Therefore, the Weber point is the only possible contraction point of X.*

Proof: Let R be the rotation by 180^o about $w := W(X)$. We will show that the operation R gives a matching copy of the rays r_i. Let p_i, p_{i+2} $(i = 1, 2)$ two opposite points, i.e., w lies on the straight line going through p_i, p_{i+2}. If c lies on that line too, the transformed rays intersect. In the other case, we obtain a parallelogram, see Figure 4.22.

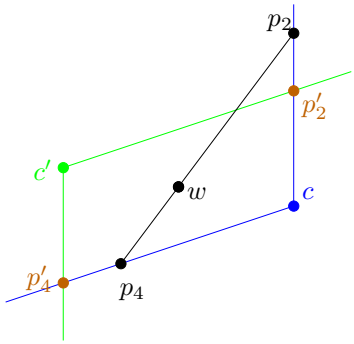

Figure 4.22: Copied rays form a parallelogram.

The points p'_i are defined as the intersections of the corresponding rays. □

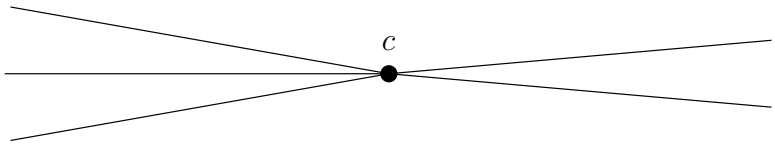

Figure 4.23: Set of rays that cannot be matched to a copy.

Again, this result cannot be generalized to five or more points.

Lemma 4.32 *For the set of rays in Figure 4.23, there is no matching copy.*

4.7 Model modifications

Thus far, we have restricted ourselves to a rather simple model. To study the limits of our results, we discuss more general models.

Higher Dimensions
Until now, the points $p \in X$ have been points in $\mathbb{R}^2$. But what about higher dimensions? If we extend the definitions from $\mathbb{R}^2$ to $\mathbb{R}^d$, we obtain the following results.

Theorem 4.33 *Let $X = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ be a multi set for a dimension $d \geq 1$, let c be a contraction point for X. Then c lies in the convex hull of X.*

Proof: The proof is done by induction. The theorem is true for dimensions $d = 1, 2$. Let $d > 2$, assume for contradiction that there is a possible contraction point c which is not in the convex hull of X. Therefore, all points have to move through a $(d-1)$–dimensional plane which does not contain c. Due to induction every contraction point must lie inside this plane. This is a contradiction. □

Unfortunately theorem 4.25 about the convex core cannot be generalized.

Example: Let $X = \{p_1, p_2, p_3, p_4\} \subset \mathbb{R}^3$ such that the four points are the vertices of a regular tetrahedron. Due to symmetries, there is only one contraction point inside the tetrahedron. But the convex core of X is empty.

Multiplicity
To this point, we have assumed that the robots can detect multiplicity, i.e., every robot can count the number of robots on a position. The point formation problem is not solvable, if the robots do not have this property. To see this, let us assume that $n-1$ out of n robots have reached the destination point; if the robots cannot detect multiplicity, this configuration is equivalent to the not solvable configuration of two different points.

General Trajectories
We restricted ourselves to straight line movement towards the destination point. One can ask, whether there are contraction functions which are invariant against other kinds of continuous movement. For some special cases, there is a positive answer. We illustrate this with the example from Figure 4.24. The point p_1 is the Weber point. In order to

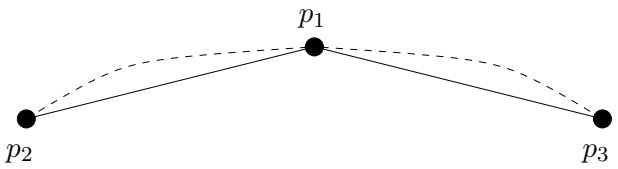

Figure 4.24: Several possibilities for movement

move towards p_1, the points p_2, p_3 can use the dashed paths. Let p_2' be a point on the left dashed line, let p_3' be a point on the right dashed line. For $X' := \{p_1, p_2', p_3'\}$, p_1 is the Weber point.

Distance Function
We defined the contraction problem for the Euclidean distance only. There are many other metrics, one important example is the so called Manhattan metric. The Manhattan metric plays an important role in facility location. To measure a distance in the Manhattan metric, one has to know two directions. Figure 4.25 shows an example that the Manhattan

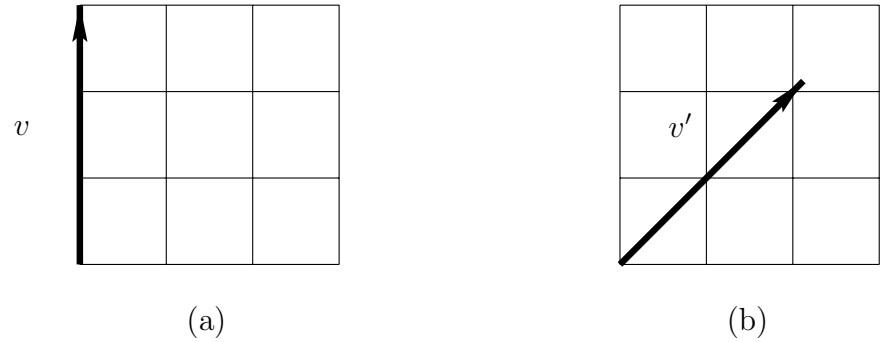

Figure 4.25: Two vectors of different lengths

distance is not independent of the orientation. In part (a), the vector v has Manhattan length 3, in part (b) v is rotated by 45^o, but the rotated vector v' has Manhattan length $2\sqrt{4.5} > 4$. [Dre83] mentions that "if a function of two points in the plane is invariant to translation and rotation of the system of coordinates, then it must be a function of the Euclidean distance between the points". Therefore, the Manhattan metric is useless in the context of contraction functions.

It is interesting that for the Manhattan metric, the Weber point for an arbitrary set of points can be easily computed. If the points have coordinates $p_i = (x_i, y_i)$, one Weber point is given by the point $w := (\text{median}\{x_i\}, \text{median}\{y_i\})$. Neither the Weber point nor the paths to the Weber point must be unique. For instance, for the points $\{(0,0),(1,0),(0,1),(1,1)\}$, every point inside this square is a Weber point.

Furthermore, it can be used to have a look at the *continuous Weber problem*, [FMW00]: For a bounded region $D \subset \mathbb{R}^2$, the function

$$z \mapsto \int_D \text{dist}(z,p)dp$$

is defined. The Weber point is the point that minimizes this integral. While [FMW00] presents some results for the Manhattan metric, it seems to be impossible to get similar results for the Euclidean metric.

Circle Formation

If the Weber point is not in the given set, it can be used to get a solution for the *circle formation problem*, mentioned in [SS90, CFK97]. In this problem, the robots should move such that all their positions are on a circle with radius bigger than 0. The center of the circle is defined as the Weber point, the radius can be defined in different ways. If the robots use all the same unit distance, an arbitrary radius can be chosen. If this is not the case, the distances from the Weber point can be used, e.g., define the radius as the smallest distance from a point p_i to w. For any fixed $k \in \{1,\ldots,n\}$, the kth smallest distance to the Weber point can be chosen as radius, too.

If the w is in the point set, then it can be used to solve a modified circle formation problem, in which all robots have to move to a circle or to the center of this circle. The robots at the Weber point remain there, the other robots form a circle around this center.

4.8 Conclusion

In order to solve the point formation problem, we defined contraction functions. This solution works in an asynchronous setting, and it is quite easy to prove and to understand this. For multi sets of cardinality smaller than five, we presented geometric constructions for contraction points. Furthermore, we showed that these contraction points are unique: First, we showed uniqueness for symmetric multi sets; second, the contraction property of contraction functions proved uniqueness for arbitrary multi sets. It turned out that contraction functions are interesting for their own sake, and we presented several interesting properties of contraction functions, e.g., that every contraction point for a multi set of points lies in the convex hull of these points.

The further discussion led to the Weber point. The Weber point plays an important role in facility location, and we showed that the Weber point is a possible contraction point. With this relation, we were able to "rederive" the result that the Weber point of a multi set of points lies in the convex hull of these points.

We proved new results as well: For an even number of points, every contraction point is in the convex core. The convex core is a subset of the convex hull. Since the Weber point is a possible contraction point, the Weber point is in the convex core, too.

It is an open question, whether the Weber point is the only possible contraction point. For multi sets with cardinality less than five, i.e., the Weber point can be constructed by compass and ruler, we proved that the Weber point is the only possible contraction point. Another open question is whether there is sub–quadratic algorithm to detect whether a given multi set of points contains its Weber point.

A challenge for future research is to look at spaces other than the Euclidean plane. For instance, it seems to be interesting to study the point formation problem on the sphere, because the sphere plays an important role in facility location (see [Dre83]). The sphere has a more complex structure than the plane, since for two antipodes, the shortest path between them is not unique.

Chapter 5

Point Formation on a line: Contraction Functions and Weber point

5.1 Introduction

In the previous chapter, we discussed the contraction functions for points. In order to get a better insight as to show how broadly the concept of contraction functions can be applied, we add a new structure to the problem: In addition to the points, a straight line $g \subset \mathbb{R}^2$ is given, and the goal is to find a contraction point on g. This line g is only a point set and has no inner structure, e.g., there are no emphasized points or directions.

The additional straight line is a natural extension of the model, since we can assume that the line has no emphasized points. For a circle, this does not hold, because the center of the circle is defined uniquely. In this chapter, we discuss contraction functions for points and a line. Therefore, we call this the P(oints)L(ine)–problem.

This chapter has the following structure. After having defined contraction functions in section 5.2, we discuss the properties and compare these with results from the previous chapter. In section 5.3, a generalized definition of Weber point is given and properties are presented. Section 5.4 shows two additional contraction functions. The chapter is concluded by section 5.5.

5.2 Contraction Functions and their properties

For this problem, we extend the definition of contraction functions in the following way.

Definition 5.1 *Let $n \in \mathbb{N}$ be fixed, let C be a function that maps a multi set $X = \{p_1, \ldots, p_n\}$ and a straight line g to a (contraction) point $c := C(X, g) \in \mathbb{R}^2$. C is called a* Contraction Function for the PL–problem *if the following properties are fulfilled for every multi set X and every straight line g.*

1. *The contraction point is on the line g, i.e., $C(p_1, \ldots, p_n, g) \in g$.*
2. *There is no ordering among the points, i.e., for every permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ holds:*
$$C(p_{\pi(1)}, \ldots, p_{\pi(n)}, g) = C(p_1, \ldots, p_n, g)$$

3. *The robots do not have a common coordinate system: For every isometric function* $o : \mathbb{R}^2 \to \mathbb{R}^2$*, it holds:*

$$C(o(p_1), \ldots, o(p_n), o(g)) = o(C(p_1, \ldots, p_n, g))$$

Furthermore, the line has no emphasized points: If o *is a translation parallel to* g*, then*

$$C(o(p_1), \ldots, o(p_n), g) = o(C(p_1, \ldots, p_n, g))$$

The line has no emphasized direction: If o *is a mirroring perpendicular to* g*, then*

$$C(o(p_1), \ldots, o(p_n), g) = o(C(p_1, \ldots, p_n, g))$$

4. *Linear movement towards the contraction point does not change it: Let* $c := C(X, g)$*, for every vector* $t = (t_1, \ldots, t_n) \in [0, 1]^n$ *holds*

$$C(\ldots, (1 - t_i)p_i + t_i c, \ldots, g) = c$$

If $C(X, g) = c$ *for a contraction function* C*, then* c *is called a* Possible Contraction Point *for* X, g*.*

Definition 5.1 is very similar to Definition 4.1, therefore, the following lemma is not very surprising.

Lemma 5.2 *If* $X = \{p_1\}$*, then the only possible contraction point is the perpendicular projection of* p_1 *onto* g*.*

Proof: Let q be the perpendicular projection of p_1 onto g. It is obvious, that q is a contraction point. Therefore, we only have to show uniqueness.

Let C be a contraction function, let $o : \mathbb{R}^2 \to \mathbb{R}^2$ be defined as the mirroring with mirror axis as the straight line through p_1 and q. For this, we obtain

$$C(p_1, g) = C(o(p_1), g) = o(C(p_1, g))$$

This can only be fulfilled for $C(p_1, g) = q$. □

But there are differences between the two definitions, too. The contraction problem was not solvable for the case of two different points. The line g as a additional geometric entity does not help to solve the problem for two different points, if both points are on g. But there is a solution, if this is not the case.

Lemma 5.3 *PL–problem for two different points is solvable with a unique solution, if not both points are on* g*.*

Proof: First, we regard the case where one of the points is on g, wlog. $p_1 \in g$. In this situation, $c := p_1$ is a contraction point. Let C be a contraction function, then

$$c' := C(p_1, p_2) = C(c, c')$$

But c, c' are points on g. If $c \neq c'$, then the problem is not solvable. Therefore, $c' = c$.

Now, we can assume that both points are not on g. The line g divides the plane in two half planes, see Figure 5.1.

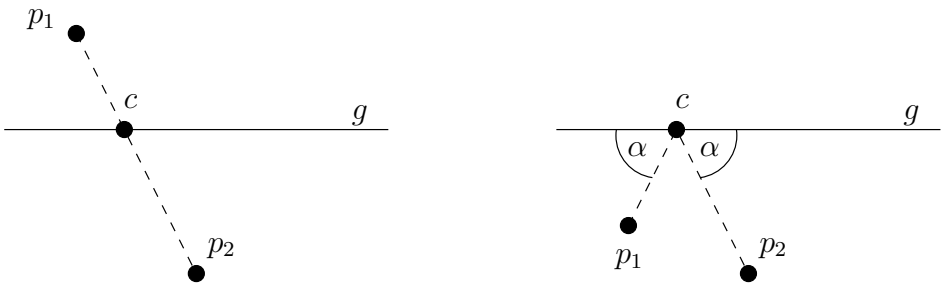

Figure 5.1: Contraction point c

If the two points are in different half planes, we define c as the intersection of g with the straight line through p_1, p_2. To show uniqueness, we assume, that both points have the same distance from g. Let R be the rotation by 180^o about c , let M be the mirroring with the straight line perpendicular to g and through c as mirror axis. For this and an arbitrary contraction function C' we obtain

$$C'(p_1, p_2, g) = C'(p_1, p_2, M(g)) = C'(p_2, p_1, M(g)) = R(C'(p_1, p_2, g))$$

This shows that $C'(p_1, p_2, g) = c$. The contraction property shows that this is true for arbitrary distances from g.

If the two points are in the same half plane, we mirror one of the two points into the other half plane and obtain c as before. To show uniqueness, we assume that both points have the same distance from g. This is a symmetric situation and allows only c as contraction point. Once again, the contraction property shows the uniqueness for arbitrary distances from g. □

For two points, c can be computed very easily. If one point p_i is on g then this point is the contraction point. Therefore, we can assume that both points are in one half plane defined by g. If we use coordinates $g(x) = (x, 0)$ and $p_i = (x_i, y_i)$ it can be assumed that $y_i > 0$ and $x_1 \leq x_2$. Then, for $c = (x_c, 0)$ we obtain

$$\tan \alpha = \frac{y_1}{x_c - x_1} = \frac{y_2}{x_2 - x_c}$$

This leads to

$$x_c = \frac{y_1 x_2 + y_2 x_1}{y_1 + y_2}$$

Remark: The problem for 3 points has no longer a unique solution, in Figure 5.2 an example is given. Given are the points $\{p_1, p_2, p_3\}$ and the straight line g. The point p_1 is the only point on g, therefore, $c_1 := p_1$ is a possible contraction point. The line g implies two possible orderings, but in both cases p_2 is the median. A second possible contraction point c_2 is the orthogonal projection of p_2 onto g.

We will show that some results from the previous chapter can be generalized. One of this results is the majority lemma 4.5.

Lemma 5.4 *If there is a $q \in g$ with $X(q) =: k > \frac{|X|}{2}$, then q is the only possible contraction point for X, g.*

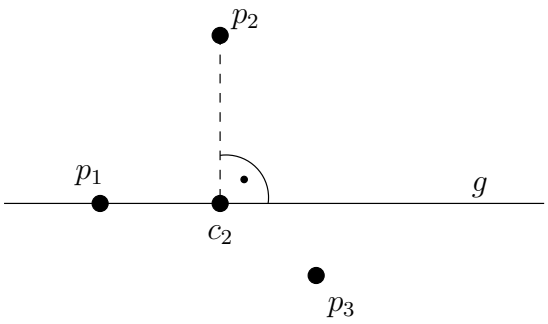

Figure 5.2: Example for a non uniqueness

Proof: The proof of lemma 4.5 still holds with the new interpretation. □

The following lemma shows that the extension property still holds.

Lemma 5.5 *The following extension property holds for the PL–problem. Let C be a contraction function and $c := C(p_1, \ldots, p_n, g)$. For a $t = (t_1, \ldots, t_n) \in [-1, \infty)^n$ define $p'_i := p_i + t_i(p_i - c)$. It holds*

$$C(p'_1, \ldots, p'_n, g) = c$$

Proof: The proof of lemma 4.6 still holds with the new interpretation. □

Theorem 4.7 stated that a contraction point must lie in the convex hull. In order to derive a similar result for the PL–problem, we need the following lemma.

Lemma 5.6 *Let $g^\perp$ be a straight line perpendicular to g. If $X \subset g^\perp$ then the intersection point $c := g \cap g^\perp$ is the unique contraction point for X, g.*

Proof: The point c is a possible contraction point because the geometric property that all points are on $g^\perp$ is invariant under movement towards c. Let $o : \mathbb{R}^2 \to \mathbb{R}^2$ be mirroring on $g^\perp$, for this we obtain

$$C(p_1, \ldots, p_n, g) = C(o(p_1), \ldots, o(p_n), g) = o(C(p_1, \ldots, p_n, g))$$

Therefore, every contraction point must be on $g^\perp$. □

This lemma is a special case of the following theorem. We presented the lemma, because we will use it to prove the theorem.

Theorem 5.7 *Every possible contraction point lies in the interval I defined as the orthogonal projection of the convex hull of X onto g.*

Proof: Assume for contradiction that there is a possible contraction point c outside of I. Let $p \in X$ be a point such that the orthogonal projection of p is the boundary point of I near c. Let $g^\perp$ the straight line through p perpendicular to g. In order to move to c all points have to cross $g^\perp$. Due to the result in Lemma 5.6 this implies that c must be the orthogonal projection of p, therefore $c \in I$. This is a contradiction. □

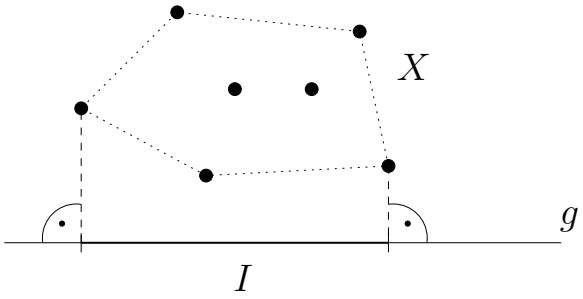

Figure 5.3: Orthogonal projection of the convex hull

5.3 Weber Point

In order to get contraction functions for more than two points, we generalize the definition of Weber points.

Definition 5.8 *Let $X = \{p_1, \ldots, p_n\}, g$ be a PL–instance. The* Weber Point $W(X, g)$ *is defined as the point $w \in g$ that minimizes*

$$z \mapsto f_{|g}(z) := \sum_{p_i \in X} |p_i - z| \quad z \in g$$

The following theorem gives an exact characterization of the uniqueness of the Weber point.

Theorem 5.9 *Let X, g be a PL–instance. The Weber point $W(X, g)$ is unique, except in the case that X consists of collinear points without median and all points are on g.*

Proof: It is well known, that the function $f(z) := \sum_{i=1}^{n} |p_i - z|$ is a convex function on $\mathbb{R}^2$, see [KM97].

Assume that the points $p_i \in X$ are not collinear. Then f is strictly convex. Therefore the restricted function $f_{|g}$ is strictly convex with a unique minimum.

Regard the collinear case. If there is a $p_i \in X, p_i \notin g$ we compute for arbitrary $q_1, q_2 \in g, q_1 \neq q_2$ and a $q := (1-t)q_1 + tq_2, t \in (0,1)$

$$\begin{aligned} |p_i - q| &= |p_i - ((1-t)q_1 + tq_2)| = |(1-t)(p_i - q_1) + t(p_i - q_2)| \\ &< (1-t)|p_i - q_1| + t|p_i - q_2| \end{aligned}$$

I.e., the function $q \mapsto |p_i - q|$ is strictly convex on g. Since the restriction $f_{|g}$ is a sum of convex functions and one of them is strictly convex, $f_{|g}$ is strictly convex and has a unique minimum.

If X is collinear with a median and all points are on g, the median is the unique Weber point for X, g.

If X is collinear without a median and all points are on g, the set of Weber points consists of a line segment. □

Similar to theorem 4.10, we show that the Weber point can be used as contraction point.

Theorem 5.10 *Let X, g be a PL-instance. If the Weber Point for X, g is unique, then it is a possible contraction point.*

Proof: Once again, we have to show, that linear movement towards the Weber point does not influence it. Let $w := W(X, g)$ be the Weber point for $X = \{p_1, \ldots, p_n\}, g$. For a $t \in [0, 1]$ define $p' := (1 - t)p_1 + tw$. Due definition, it holds

$$\sum_{i=1}^{n} |p_i - w| < \sum_{i=1}^{n} |p_i - z| \quad \forall z \in g \setminus \{w\}$$

This and the inequality $|z - p_1| \leq |z - p'| + |p' - p_1|$ lead to

$$\begin{aligned} \sum_{i=2}^{n} |p_i - w| + |p' - w| &< \sum_{i=2}^{n} |p_i - z| + |p_1 - z| + \underbrace{|p' - w| - |p_1 - w|}_{=-|p'-p_1|} \\ &\leq \sum_{i=2}^{n} |p_i - z| + |p' - z| + |p' - p_1| - |p' - p_1| \\ &= \sum_{i=2}^{n} |p_i - z| + |p' - z| \end{aligned}$$

This means that w is the unique Weber point for the multi set $(X \uplus \{p'\}) \setminus \{p_1\}$ and g. □

The methods used in the proof of theorem 4.13 can be used to give the following characterization of the Weber point. To our knowledge, this characterization has not been published before.

Theorem 5.11 *For a vector $v \in \mathbb{R}^2$ let v_g be the projection of v onto g. If the Weber point $w := W(X, g)$ is unique, then it has the following property.*

$$\left| \sum_{\substack{i=1 \\ p_i \neq w}}^{n} \frac{(p_i - w)_g}{|p_i - w|} \right| \leq X(w)$$

And w is the only point with this property. Especially, if $w \notin X$, i.e., $X(w) = 0$, we obtain

$$\sum_{i=1}^{n} \frac{(p_i - w)_g}{|p_i - w|} = 0$$

Proof: Define $g(z) := |p - z|$ for an arbitrary point $p \in \mathbb{R}^2$. If $z \neq p$ we compute the derivative as

$$g'(z) = \frac{(p_i - z)_g}{|p_i - z|}$$

If $z = p$ we obtain for the partial derivation in direction $v \in g \cap S^1$

$$g_v(z) = 1$$

For the function

$$\tilde{f}(z) := \sum_{\substack{i=1 \\ p_i \neq w}}^{n} |p_i - z| = f(z) - k|w - z|$$

and the partial derivation in direction $v \in g \cap S^1$, we obtain

$$f_v(w) = \left\langle \tilde{f}'(w), v \right\rangle + k = \left\langle \sum_{\substack{i=1 \\ p_i \neq w}}^{n} \frac{(p_i - w)_g}{|p_i - w|}, v \right\rangle + k$$

Since f is minimized in w, $f_v(w) \geq 0$ holds. The theorem is true, if $\tilde{f}'(w) = 0$. Therefore, we can assume that $\tilde{f}'(w) \neq 0$ and define $v := -\frac{\tilde{f}'(w)}{|\tilde{f}'(w)|}$. This leads to

$$0 \leq f_v(w) = -\left| \sum_{\substack{i=1 \\ p_i \neq w}}^{n} \frac{(p_i - w)_g}{|p_i - w|} \right| + k$$

On the other hand, assume that for $\hat{w} \in g$,

$$\left| \sum_{\substack{i=1 \\ p_i \neq \hat{w}}}^{n} \frac{(p_i - \hat{w})_g}{|p_i - \hat{w}|} \right| \leq X(\hat{w})$$

holds. For the function

$$\hat{f}(z) := \sum_{\substack{i=1 \\ p_i \neq \hat{w}}}^{n} |p_i - z| = f(z) - X(\hat{w})|\hat{w} - z|$$

we obtain that

$$0 \leq -\left\langle \hat{f}'(\hat{w}), \frac{\hat{f}'(\hat{w})}{|\hat{f}'(\hat{w})|} \right\rangle + X(\hat{w})$$

Since for both $v \in g \cap S^1$, holds

$$\left\langle \tilde{f}'(\hat{w}), \frac{\hat{f}'(\hat{w})}{|\hat{f}'(\hat{w})|} \right\rangle \geq \left\langle \tilde{f}'(\hat{w}), v \right\rangle$$

holds, it follows that $f_v(\hat{w}) \geq 0$. This means that f is minimized in $\hat{w}$, and since w is unique, $\hat{w} = w$. □

Remark: Since for $n = 2$ the contraction point is unique, in this case the Weber point can be constructed very easily.

The new geometric entity g cannot help to compute the Weber point for $n \geq 5$: In section 4.4.1 we cited an example from [CM69] for a not solvable instance of 5 points. Although we know that the Weber point for this set must lie on the x–axis, it cannot be constructed by compass and ruler. If we take these 5 points and define g as the x–axis, we obtain a PL–instance which is not solvable.

Furthermore, in [Baj88] it is shown that the Weber point cannot be constructed by compass and ruler for $n \geq 3$.

5.4 Additional solutions

As we have seen, the solution for the PL–problem is not unique in the general case. In this section, we present two contraction points different from the Weber point. Furthermore, we show that there are fast algorithm to compute these points, whereas the Weber point cannot be constructed.

5.4.1 Median

If n is odd, the orthogonal projection of X onto g can be used to define a *median* of X. To be more precise, the contraction point c is defined as the median of the projected points. Let $p_{i|g}$ be the projection of p_i onto g. Since $p_{i|g} - c = (p_i - c)_g$ and c is the median, we compute

$$|\{i \in \{1,\ldots,n\} | p_{i|g} = c\}| \geq \left| \sum_{\substack{i=1 \\ p_{i|g} \neq c}}^{n} \frac{p_{i|g} - c}{|p_{i|g} - c|} \right| = \left| \sum_{\substack{i=1 \\ p_{i|g} \neq c}}^{n} \frac{(p_i - c)_g}{|(p_i - c)_g|} \right|$$

Again, this shows a similarity between a contraction point, the median and the Weber point. The median can be computed for every odd n. Since this doesn't hold for the Weber point, the median is different from the Weber point. The projection of n points can be computed in time $O(n)$, the median of the projected points can be computed in time $O(n)$, too, see [BFP$^+$73]. Therefore the algorithm needs time $O(n)$.

5.4.2 Cone

We regard the situation that no point is on g or all points on g have the same position. In this situation, a contraction point different from the Weber point can be computed. In contrast to the non constructible Weber point, this solution can be computed in time $O(n \log n)$.

In a first step, it is assumed that all points $p \in X$ are in one of the two half-planes defined by g.

Definition 5.12 *Let $q \in g$ be an arbitrary point, let $\alpha \in [0^o, 90^o]$. The set*

$$C(q, \alpha) := \{x \in \mathbb{R}^2 \,|\, angle((x - q), g) \geq \alpha\}$$

is called a cone.

For $q \in g$ we define

$$\alpha_q := \max_{q \in g} \{\alpha \in [0^o, 90^o] | X \subset C(q, \alpha)\}$$

Since X is finite, this maximum exists. $q_{\max} \in g$ is defined as the point that maximizes the function $q \mapsto \alpha_q$.

To simplify notation, $C_{\max} := C(q_{\max}, \alpha_{q_{\max}})$ is called the maximum cone.

Example: In Figure 5.4 an example is given. Since $angle((p_q - q), g) = \alpha_q$, for every $\alpha > \alpha_q$, $X \not\subset C(q, \alpha)$ holds.

Theorem 5.13 *Let X be a multi set, such that there is at most one $q \in g$ with $X(q) > 0$. Then the point $q_{\max}$ exists, it is well defined, and it is a possible contraction point.*

Proof: If there is a point $q \in g$ with $X(q) > 0$, then $\alpha_q > 0$ and for every $q' \neq q$, $\alpha_{q'} = 0$ holds. Therefore, $q_{\max} = q$ is defined uniquely and it is a possible contraction point.

If no point is on g, the function $q \mapsto \alpha_q$ is continuous and $\alpha_q > 0$. For every vector $\vec{g}$ parallel to g and for every point $O \in g$, $\lim_{t \to \infty} \alpha_{(O + t\vec{v})} = 0$ holds. Therefore, at least one $q_{\max}$ exists.

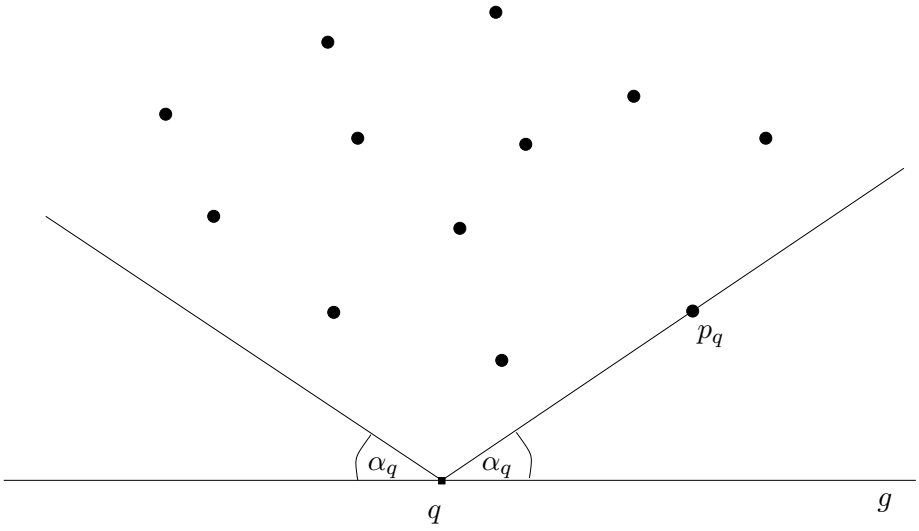

Figure 5.4: Cone for a given $q \in g$

For every $q_{\max}$, there are at least two points $p_l, p_r \in X$ such that

$$angle((p_l - q_{\max}), g) = angle((p_r - q_{\max}), g) = \alpha_{q_{\max}}$$

and $angle((p_l - q_{\max}), (p_r - q_{\max})) = 180^o - 2\alpha_{q_{\max}}$. This means that there are points on every straight line that bounds the maximum cone $C_{\max}$. Assume for contradiction that this is not true. Due to the definition of α_q there is at least one point of X on the boundary of $C(q, \alpha_q)$. Since there is no point on one straight line, there is a $q' \in g$ on the other side with $\alpha_{q'} > \alpha_{q_{\max}}$, which is a contradiction.

Since there are points on each straight line of the cone, the movement to the left or the right to a point $q' \in g$ decreases the corresponding angle $\alpha_{q'}$. Therefore, $q_{\max}$ is unique.

If a point $p \in X$ moves towards $q_{\max}$, the angle $\alpha_{q_{\max}}$ remains constant or is increased. For every other $q \in g$ the angle α_q remains constant or is decreased. This shows that it is a possible contraction point. □

Due to the fact that in the case $n = 2$ there is exact one solution, Weber point and $q_{\max}$ are equal if X consists of two points. This is not true for more than two points. This can be seen as follows. Let $X = \{p_1, p_2\}$ be a multi set with $p_1 \neq p_2$, $W(X, g) = q_{\max}(X)$ holds. But for the multi set $X' := X \uplus \{p_1\}$, $q_{\max}(X) = q_{\max}(X')$ holds, while the Weber points are different.

Furthermore, this shows that $q_{\max}$ depends only on two points $p_l, p_r \in X$, while all other points can move inside the maximum cone $C_{\max}$ without influencing $q_{\max}$. In order to compute $q_{\max}$, one can search for a pair p_l, p_r that is on both straight lines bounding $C_{\max}$. This leads to the following simple algorithm to compute $q_{\max}$, the notation is taken from Figure 5.1.

Point $q := undef$
Angle $\alpha_{\max} := 0$
For every pair (p_i, p_j)
  Compute c, α for (p_i, p_j).
  If $X \subset C(c, \alpha)$
    If $\alpha > \alpha_{\max}$

$\alpha_{\max} := \alpha$
$q := c$
$q_{\max} := q$
$\alpha_{q_{\max}} := \alpha_{\max}$

It takes $O(n)$ steps to check, whether a given cone contains all points. Since there are $O(n^2)$ many pairs of points, the algorithm needs time $O(n^3)$. The running time can be reduced by some pre–processing, which we will describe later, but this does not change the worst case analysis. Nevertheless, the search for a "good" pair (p_i, p_j) can be done in a more efficient way.

Theorem 5.14 *The point $q_{\max}$ can be computed in time $O(n \log n)$.*

To prove the theorem, we present the following algorithm.

Max–Cone Algorithm
For a pair (p_i, p_j), we can compute c, α. If one of the points is not on the boundary of the convex hull of X then $X \not\subset C(c, \alpha)$. Therefore, only points $p \in X$ have to be regarded, if they lie on the boundary of the convex hull. Computing the convex hull takes $O(n \log n)$ time, [Ede87]. In the following, it is assumed that X is a convex (multi) set.

We introduce the coordinates $g(x) := (x, 0)$ and $p_i = (x_i, y_i)$ for $i \in \{1, \ldots, n\}$. Due to the assumption that all points are in one of the two half planes, it can be assumed that $y_i \geq 0$. Then X is sorted in x–direction in $O(n \log n)$ time, such that $x_1 \leq x_2 \leq \ldots \leq x_n$. This implies an orientation, we say that x_1 is at the left side and that x_n is at the right side. Every cone is bounded by 2 straight lines, a left one and a right one. Let us look at the left line. The point p_1 can be on the left line of the maximum cone $C_{\max}$. But if $y_1 < y_2$, then p_2 can not lie on the left line of $C_{\max}$. To be more precise, let p_i, p_{i+k} be two points, if $y_i < y_{i+k}$, then p_{i+k} cannot be on the left line of the maximum cone. A similar result holds for the right line of the cone: If $y_i > y_{i+k}$, then p_i cannot be on the right line of the maximum cone.

With this, we define two lists L, R: In L are all points which are possibly on the left line, in R are all points which are possibly on the right line. The computation is done due to the following schemes.

$L := (p_1)$
$y_L := y_1$
For $k := 2$ to n
  If $y_k < Y_L$
    $L := L.append(p_k)$
    $y_L := y_k$
Next k

$R := (p_n)$
$y_R := y_n$
For $k := (n-1)$ to 1
  If $y_k < Y_R$
    $R := R.append(p_k)$
    $y_R := y_k$
Next k

After this step, it can be assumed, that $X = L \uplus R$ has a "bowl form" as indicated in Figure 5.5. It shows the pre–processed set of points from Figure 5.4. In this example $L = (p_1, p_2, p_3, p_4)$ and $R = (p_6, p_5, p_4)$.

Let p_i, p_{i+1} be two successive points from list L, let $q_{i,i+1}$ be the intersection from the straight line through p_i, p_{i+1} with g. If $q_{i,i+1} = q_{\max}$ then both points are on the left line bounding the maximum cone. If $q_{i,i+1} < q_{\max}$ then p_i can not be on the boundary, if $q_{i,i+1} > q_{\max}$ then p_{i+1} can not be on the boundary. A analogous result holds for two points $p_i, p_{i-1} \in R$.

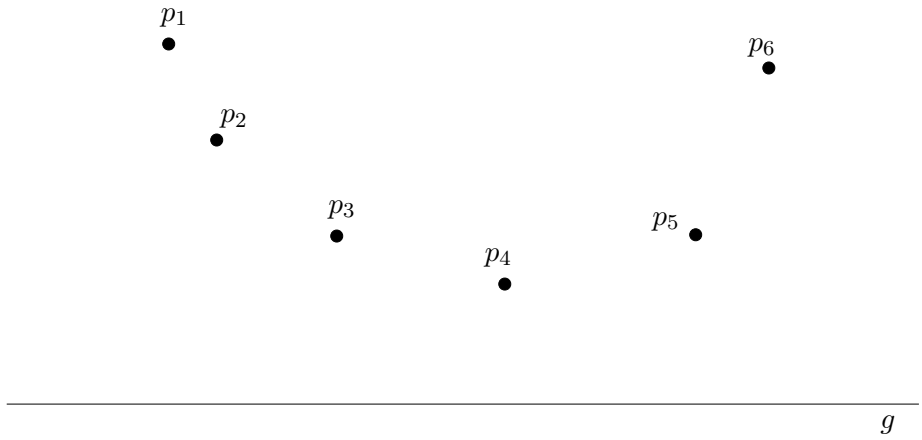

Figure 5.5: The pre–processed set from Figure 5.4

Let q_i be the perpendicular projection of p_i onto g. Generate a list I of interval boundaries as follows:

$I := (q_1)$
For every pair $p_i, p_{i+1} \in L$
  If $q_{i,i+1} \in [q_1, q_n]$
    $I := I.insert(q_{i,i+1})$
$I := I.insert(q_n)$
For every pair $p_i, p_{i-1} \in R$
  If $q_{i,i-1} \in [q_1, q_n]$
    $I := I.insert(q_{i,i-1})$
Sort I

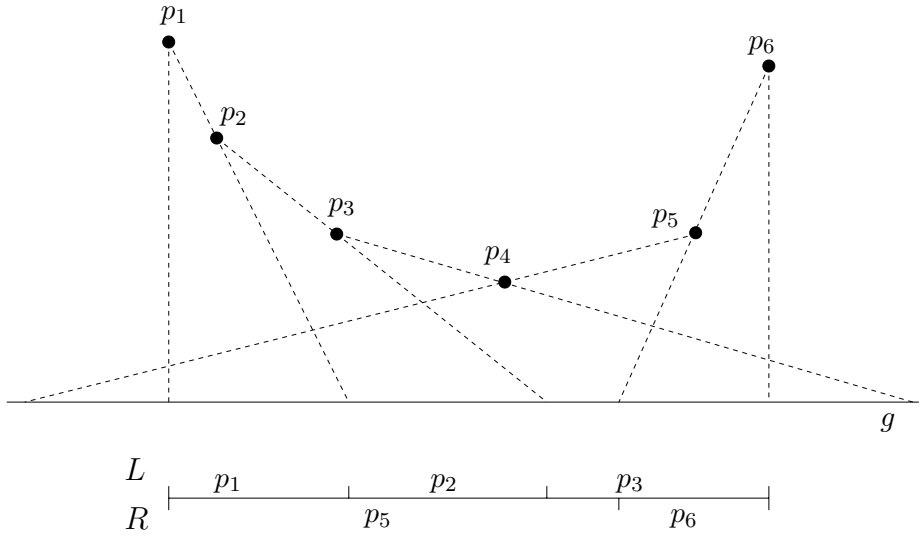

Figure 5.6: Computing the intervals

Figure 5.6 shows the computed intervals for the example from Figure 5.4. For the following part, the algorithms needs two variables p_l, p_r; $p_l \in L$ indicates the "important" point on the left side, $p_r \in R$ indicates the "important" point of the right side. For a

point $q \in g$ and a point $p \in \mathbb{R}^2$, $\angle(q,p)$ denotes the angle between $p-q$ and g. It is assumed that $0^o \leq \angle(q,p) \leq 90^o$.

$p_l := p_1$
$p_r := p_j$, where $j := \min\{\nu | q_{\nu+1,\nu} \in [q_1, q_n], q_\nu, q_{\nu+1} \in R\}$
$q := q_1$
$\alpha_l := 90^o$
$\alpha_r := \angle(q, p_r)$
While $\alpha_l > \alpha_r$
  $q := I.successor(q)$
  If $q = q_{i,i+1}$
    $p_l := p_{i+1}$
  If $q = q_{i+1,i}$
    $p_r := p_{i+1}$
  $\alpha_l := \angle(q, p_l)$
  $\alpha_r := \angle(q, p_r)$
If $q = q_{i,i+1}$
  $p_l := p_i$
If $q = q_{i+1,i}$
  $p_r := p_i$
p_l, p_r are on the boundary of the maximum cone

Example: For the example from Figure 5.6 we obtain that p_2 and p_5 are on the boundary of the maximum cone.

To finish the proof of the theorem, correctness has to be shown.

Lemma 5.15 *The Max–Cone algorithm computes* $q_{\max}$.

Proof: We have seen that the maximum cone depends on two points only. The Max–Cone algorithm identifies these points. □

Up to now, we assumed that X is in one of the two half planes defined by g. If this is not the case, the coordinates of the points in one half plane can be mirrored into the other half plane. The computation is not influenced by the choice of the half plane.

5.5 Conclusion

We presented and discussed a generalization of the contraction problem from the previous chapter: The contraction point must be on a given straight line. We have seen that the concept of contraction functions can be generalized to the PL–problem. For multi set of cardinality of size less than three, we presented geometrical constructions for contraction points. Furthermore, with the help of symmetries, we have shown that these points are unique. Several properties of contractions functions have been derived and compared to results from the previous section.

The notion of Weber point has been generalized, too. We presented several properties of this generalized Weber point. It turned out, that for multi set of cardinality of size less than three, i.e., the Weber point can be constructed by compass and ruler, the Weber point is the only possible contraction point. Interestingly, it is easy to show that the

Weber point is not the unique solution for multi sets of bigger cardinality. We presented two contraction functions different to the mapping to the Weber point. These contraction functions can be computed efficiently.

Chapter 6

Generalizations of the Weber point

6.1 Introduction

In this chapter, we do not discuss contraction, but we will use the techniques derived in the previous chapters to discuss modifications of the Weber problem. The original Weber problem has points as input, and the solution for this problem is a point: Points are mapped to a point. It is a natural generalization to think of straight lines instead of points. In the following we discuss the two problems:

- Given n straight lines in the plane, find a point that minimizes the sum of the distances to the given lines.
- Given n points in the plane, find a straight line that minimizes the sum of the distances to the given points.

The two problems can be regarded as dual problems. They can be motivated as follows: Given n railway tracks, find a good position for a service station; given the positions of n cities, build a railway track close to the cities; given n sample points, find a straight line that fits best the data.

The distance between a point and a line is measured in the usual way, i.e., as the smallest Euclidean distance between the given point and a point on the line.

There are many variants of these problems, see [Sch99], we restrict ourself to the standard cases. This chapter in organized as follows. In section 6.2, we define the *Weber point for lines* and discuss its properties. Section 6.3 does the same for *Weber line for points*. In section 6.4, we give a short summary.

6.2 Weber point for lines

In [Bar00, ALST01], the following problems has been presented.

Definition 6.1 *Let $L = \{l_1, \ldots, l_n\}$ be a multi set of straight lines in the plane $\mathbb{R}^2$. A point $p_w \in \mathbb{R}^2$ is called* Weber point *for L, if p_w minimizes the following function:*

$$p \mapsto \sum_{l \in L} dist(l, p)$$

Example: Let $L = \{l_1, l_2\}$. If l_1 intersects l_2, then the intersection point is a Weber point. If l_1 is parallel to l_2, then every point between the two lines and every point on

one of the lines is a Weber point. In the special case that $l_1 = l_2$, every point on the line is the Weber point.

In Figure 6.1, three lines bound an equilateral triangle. Every point inside this triangle

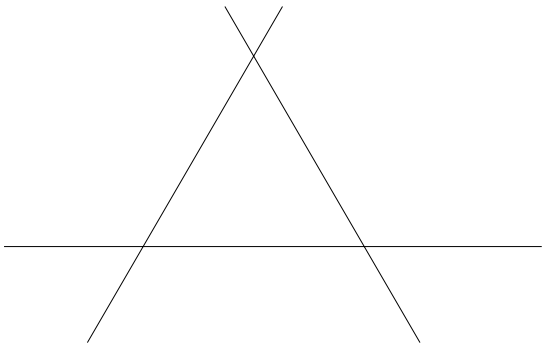

Figure 6.1: Three lines.

is a Weber point for the three lines. The same holds for every point on the boundary of the triangle.

The next theorem shows that there is a Weber point for every multi set of lines.

Theorem 6.2 ([Bar00]) *For every multi set L of straight lines, there is a Weber point p_w for L. Furthermore, there is a Weber point $p'_w \in L$, i.e. there is a line $l \in L$ with $p'_w \in l$. If two lines intersect, then there is a Weber point p''_w on an intersection of two lines.*

If there is a Weber point $p_w \notin L$, then for every line $l \in L$, there is a unit vector $\vec{l}$ parallel to l, such that $\sum_{l \in L} \vec{l} = 0$.

Proof: If L consists of parallel lines, there is a line $l \in L$ with $p_w \in l$. If there are at least two intersecting lines, for every vector $v \in S^1$ and for an arbitrary origin O, $\lim_{t \to \infty} \sum_{l \in L} \text{dist}(l, O + tv) = \infty$ holds. Since $\text{dist}(l, .)$ is continuous, there is a point p_w that minimizes the sum of distances.

As to the last claim, let p_w be a Weber point with $p_w \notin L$. For a line $l_i \in L$, we consider the function $p \mapsto \text{dist}(p, l_i)$. For $p \notin l_i$, this function is differentiable in p with gradient g_i, where g_i is the unit vector perpendicular to l_i. Since the sum of the distances is minimized in p_w, $\sum_{i=1}^{n} g_i = 0$ holds. Let R be a rotation by 90^o. For every line l_i, the unit vector $\vec{l_i} := R(g_i)$ is parallel to l_i. The following holds:

$$\sum_{l_i \in L} \vec{l_i} = \sum_{l_i \in L} R(g_i) = R(\sum_{l_i \in L} g_i) = R(0) = 0$$

Consider a Weber point $p_w \notin L$. The property of the vanishing gradient is locally constant, i.e., a point p is a Weber point, if there is no line $l \in L$ between p and p_w. The Weber point p can be arbitrarily close to a line $l \in L$. Since the distance function is continuous, there is a Weber point $p'_w \in l$.

If there is one intersection, every line $l_i \in L$ has an intersection with a line $l_j \in L \setminus \{l_i\}$. Let $p'_w \in l_i$ be a Weber point. Assume that p'_w is only on l_i. The function

$$p \mapsto \sum_{l \in L \setminus \{l_i\}} \text{dist}(l, p)$$

is differentiable in p'_w, and the gradient of this function vanishes in p'_w. This implies that the sum of distances to the other lines is constant on a line segment of l_i. This line segment is bounded by intersections with other lines. □

The theorem leads to the following corollary.

Corollary 6.3 ([Bar00]) *The Weber point for n lines can be computed in $O(n^3)$ time.*

Proof: If there is no intersection, a line l with $p_w \in l$ can be found in $O(n^2)$ time. If there are intersection, for every pair of lines l_i, l_j, it can be checked in linear time whether the intersection is the Weber point. □

6.3 Weber line for points

The following problem has been discussed in many publications, see for example [Sch99].

Definition 6.4 *Let $X = \{p_1, \ldots, p_n\}$ be a multi set of points in the plane $\mathbb{R}^2$. A straight line l_w is called* Weber line *for X, if l_w minimizes the following function:*

$$l \mapsto \sum_{p \in X} dist(l, p)$$

Example: For $X = \{p_1\}$, every line through p_1 is a Weber line. For the multi set $X' = \{p_1, p_2\}$ with $p_1 \neq p_2$, the unique line through p_1, p_2 is the unique Weber line.

In order to discuss the properties, we parameterize the lines in the plane in the following way. Choose an origin $O \in R^2$ and a vector $\vec{x} \neq 0$. For a line l, we define $r_l := \text{dist}(l, O)$ and φ_l as the angle between $\vec{x}$ and the intersection of l with the circle with radius r_l around O. Furthermore, we compute the unit vector $v_l := (\cos\varphi_l, \sin\varphi_l)$, see Figure 6.2. It is obvious that v_l is perpendicular to l. In a similar way, we define $v_\varphi := (\cos\varphi, \sin\varphi)$

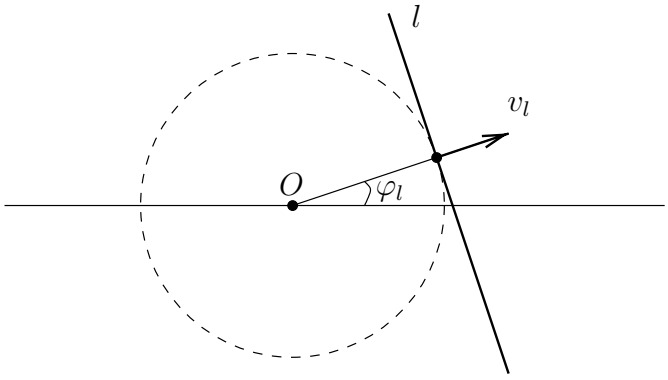

Figure 6.2: Line parameterization.

for a given φ. For a point p, $\text{dist}(l, p) = |\langle v_l, p\rangle - r_l|$ holds. It is a well known fact, that $\langle v_l, p\rangle - r_l < 0$ iff p and O are in the same half-plane. If $r_l > 0$, i.e., $O \notin l$, there is exact one φ_l, for $r_l = 0$, there are two possible values for φ_l. On the other hand, for every $r \geq 0$ and for every $\varphi \in [0, 2\pi]$, there is a line l such that $r_l = r$ and $\varphi_l = \varphi$.

With this preparation, we get the following results. Most of these results can be found in [Sch99].

Theorem 6.5 *Let X be a multi set of points. The following properties hold:*

1. *There is a Weber line l for X.*
2. *There is a Weber line l' and a point $p \in X$ such that $p \in l'$.*
3. *If there is a Weber line that does not intersect X, then $|X|$ is even.*

Proof: Every Weber line must pass the convex hull. This is due to the fact that otherwise the sum of the distances can be minimized by moving the line in direction to the convex hull. Let $R \in \mathbb{R}$ with $R > \text{dist}(O,p)$ for all $p \in X$. Since $[0,R] \times [0,2\pi]$ is a compact set and the the function $(r,\varphi) \mapsto \sum_{p\in X} \text{dist}(l,p)$ is continuous, there is a minimum for the restricted function in $[0,R] \times [0,2\pi]$. Due to definition of R, such a minimum (r_m, φ_m) cannot be obtained with $r_m = R$. Since for $r > R$, the function $(r,\varphi) \mapsto \sum_{p\in X} \text{dist}(l,p)$ is increasing in r, the (r_m,φ_m) minimizes the non-restricted function. Therefore, l with $r_l = r_m$ and $\varphi_l = \varphi_m$ is a Weber line.

As to the second claim, let l be a Weber line disjoint to X, i.e., $\langle v_{\varphi_l}, p\rangle - r_l \neq 0$ for all $p \in X$. Therefore, the function

$$(r,\varphi) \mapsto f(r,\varphi) := \sum_{p\in X} |\langle v_\varphi, p\rangle - r|$$

is differentiable in (r_l,φ_l). Let $POS := \{p \in X | \langle v_{\varphi_l}, p\rangle - r_l > 0\}$, and let $NEG := \{p \in X | \langle v_{\varphi_l}, p\rangle - r_l < 0\}$. For the derivative in direction $\frac{\partial}{\partial r}$, we obtain:

$$\frac{\partial}{\partial r} f(r,\varphi) = \sum_{p\in POS} -1 + \sum_{p\in NEG} 1$$

In (r_l,φ_l), this expression vanishes, i.e., $|POS| = |NEG|$. This is not possible for an odd number of points and proves the third claim.

Let $v_\varphi^\perp := v_{\varphi+90^o}$, for the derivative in direction $\frac{\partial}{\partial\varphi}$, we compute:

$$\begin{aligned}
\frac{\partial}{\partial\varphi} f(r,\varphi) &= \frac{\partial}{\partial\varphi}\left(\sum_{p\in POS}(\langle v_\varphi, p\rangle - r) - \sum_{p\in NEG}(\langle v_\varphi, p\rangle - r)\right) \\
&= \sum_{p\in POS}\langle v_\varphi^\perp, p\rangle - \sum_{p\in NEG}\langle v_\varphi^\perp, p\rangle \\
&= \left\langle v_\varphi^\perp, \sum_{p\in POS} p\right\rangle - \left\langle v_\varphi^\perp, \sum_{p\in NEG} p\right\rangle = \left\langle v_\varphi^\perp, \sum_{p\in POS} p - \sum_{p\in NEG} p\right\rangle
\end{aligned}$$

Again, for (r_l,φ_l), this expression vanishes. For all $\epsilon \geq 0$ small enough, $\frac{\partial}{\partial r} f(r_l \pm \epsilon, \varphi) = 0$ holds, i.e., $\epsilon \mapsto f(r_l \pm \epsilon, \varphi_l)$ is locally constant. This is true for all $\epsilon \geq 0$ until the corresponding line l' intersects X. Since f is continuous, l' is a Weber line, too. □

Theorem 6.6 *Let l be a straight line, l is a Weber line for X, iff the following conditions are fulfilled:*

1. $|l \cap X| \geq ||NEG| - |POS||$.
2. $$\sum_{p\in l\cap X} \left|\langle v_\varphi^\perp, p\rangle\right| \geq \left|\left\langle v_\varphi^\perp, \sum_{p\in POS} p - \sum_{p\in NEG} p\right\rangle\right|$$

Proof: We have seen already that for $l \cap X = \emptyset$, the Theorem is true. For $l \cap X \neq \emptyset$, the function f is not differentiable, but we can compute the derivatives in direction $\pm\frac{\partial}{\partial r}, \pm\frac{\partial}{\partial \varphi}$. □

Example: Let $X = \{p_1, p_2, p_3\}$ be a set of points, such that the 3 points form an equilateral triangle. Every straight line through two of the 3 points is a Weber line.

6.4 Conclusion

We discussed two natural generalizations of the classical Weber problem. Instead of points, we considered points and lines. We have seen that the methods for contraction we derived in the previous chapters can be used to derive results for the generalized Weber problem as well. Especially, using directional derivatives, we got characterizations for the generalized problems. This gave short and elegant proofs for the results, whereas, for instance, [Bar00] had to consider many cases to prove his results.

Chapter 7

Conclusion and Outlook

In this thesis, we highlighted three topics of distributed systems and discussed several aspects of them.

Distributing Rare Resources was the first topic. We discussed Roman Domination, Win–Win, and several derived problems. These are typical graph problems and we showed that they are NP–hard. Furthermore, a PTAS for Planar Roman Domination was given. It is an open question, whether there is a PTAS for Planar Win–Win, too. We discussed the relations between these problems: For some problems, less resources are needed than for Roman Domination. However, these problems need more communication. Especially, for Roman Domination, the decision which server should service a request can be made locally; whereas for Online Dynamic Win-Win, this is a NP–complete problem.

Second, we presented a highly available data structure, called HAT. In contrast to other proposed data structures, the HAT structure works in a weak environmental setting. The HAT structure has the ability to recover, i.e., a server that has suffered a crash failure can reinvolve itself into the data structure. To do this, no central instance is needed, communication between buddies is enough.

The last topic was the point formation problem. We defined contraction functions and showed that they are a solution for the point formation problem and thus, the point formation problem can be solved in a very simple robot model. It turned out that the mapping to the Weber point is a contraction function. With this result, we could show a new aspect of the Weber point: The Weber point is in the convex core, if the number of points is even. Some questions remain open: Is the Weber point the only contraction point? If an additional line is given, it is very easy to present contraction points different from the Weber point. However, if the line is not given, we do not know the right answer.

We looked at the three topics individually, but there are similarities between the solutions we presented. For instance, the HAT structure and the contraction functions solve problems in weak environmental and asynchronous settings. Local decisions are enough to coordinate the nodes in a HAT and the servers in Roman Domination. Since Roman Domination has a simple characterization, we were able to derive a PTAS for this problem; and since contraction functions have a simple structure, we were able to get new insights into the point formation problem and to derive easy to understand solutions. Interestingly, the contraction functions show that methods from facility location can be exploited to solve the point formation problem.

Distributed systems remain an object of intense research efforts, because they have great potential. They are also more complicated due to "friction" issues (such as incomplete knowledge or message delays), but as we showed in three examples, some of these problems can be handled quite well in weak and realistic models.

Bibliography

[ABFN00] J. Alber, H.L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed Parameter Algorithms for Planar Dominating Set and Related Problems. In *Proc. 7th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 1851 of *LNCS*, pages 97–110, 2000.

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation – Combinatorial optimization problems and their approximability properties*. Springer Verlag, 1999.

[AF95] J. Arquilla and H. Fredricksen. Graphing an Optimal Grand Strategy. *Military Operations Research*, pages 3–17, Winter 1995.

[ALST01] G. Aloupis, S. Langerman, M. Soss, and G. Toussaint. Algorithms for Bivariate Medians and a Fermat–Torricelli Problem for Lines. In *Thirteenth Canadian Conference on Computational Geometry (CCCG)* , pages 21–24, 2001.

[Baj88] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3:177–191, 1988.

[Bak94] B. Baker. Approximation Algorithms for NP–Complete Problems on Planar Graphs. *J. ACM*, 41(1):153–180, 1994.

[Bar00] R. Barbara. The Fermat–Torricelli points of n lines. *Math. Gazette*, 84(499):24–29, 2000.

[BDSW99] Y. Breitbart, S. Das, N. Santoro, and P. Widmayer, editors. *Workshop on Distributed Data and Structures 2*. DIMACS Workshop, Princeton, Carleton Scientific, 1999.

[BFP+73] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[Bri95] J. Brimberg. The Fermat–Weber location problem revisited. *Math. Prog.*, 71:71–76, 1995.

[CFK97] Y. Uny Cao, A. Fukunaga, and A. Kahng. Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots*, 4:1–23, 1997.

[CM69] E. Cockayne and Z. Melzak. Euclidean constructibility in graph minimization problems. *Math. Mag.*, 42:206–208, 1969.

[CT89] R. Chandrasekaran and A. Tamir. Open questions concerning Weiszfeld's algorithm for the Fermat–Weber location problem. *Math. Prog.*, 44:293–295, 1989.

[DH02] Z. Drezner and H. Hamacher, editors. *Facility Location: Applications and Theory*. Springer–Verlag, 2002.

[dJ62] C.F. de Jaenish. Trait des Applications de l'Analyse Mathematique au Jeau des Echecs. Petrograd, 1862.

[Dör58] H. Dörrie. *Triumph der Mathematik, Hundert Probleme aus zwei Jahrtausenden mathematischer Kultur*. Physica–Verlag, Würzburg, fifth edition, 1958.

[Dre83] Z. Drezner. Constrained Location Problems in the Plane and on a Sphere. *IIE Transactions*, 15(4):300–304, 1983.

[Dre95] Z. Drezner, editor. *Facility Location: A Survey of Applications and Methods*. Springer–Verlag, 1995.

[Dre00] P.A. Dreyer. *Applications and Variations of Domination in Graphs*. PhD thesis, Rutgers University, New Jersey, 2000.

[DSL92] Z. Drezner and D. Simchi-Levi. Asymptotic Behavior of the Weber Location Problem on the Plane. *Annals of Operations Research*, 40:163–172, 1992.

[Ede87] H. Edelsbrunner. Algorithms in combinatorial geometry. In Brauer, W., Rozenberg, G., and Salomaa, A., editors, *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, 1985.

[FMW00] S.P. Fekete, J.S.B. Mitchell, and K. Weinbrecht. On the continuous Weber and k-median problems. In *Proc. 16th Symp. Computational Geometry*, pages 70–79, 2000.

[FPSW99] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *ISAAC'99, LNCS 1741*, pages 93–102, 1999.

[FPSW00] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed coordination of a set of autonomous mobile robots. In *IEEE Intelligent Vehicle Symposium (IV 2000)*, pages 480–485, 2000.

[Gib95] E. Gibbon. *Decline and Fall of the Roman Empire*. Allen Lane, 1995.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability / A Guide to the Theory of NP–Completeness*. Freeman and Company, 1979.

[GP01] V. Gervasi and G. Prencipe. Need a Fleet? Use The Force! In *Fun With Algorithms 2 (FUN 2001)*, pages 149–164, 2001.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, 1993.

[Hed00] S.T. Hedetniemi. Roman domination in graphs ii. Slides and notes from presentation at 9th Quadrienn. Aint. Conf. on Graph Theor., Combinatorics, Algorithms, and Applications, June 2000.

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci*, 9:256–278, 1974.

[KM97] Y. Kupitz and H. Martini. Geometric aspects of the generalized Fermat-Torricelli problem. In *Intuitive Geometry, Bolyai Society Math Studies, 6*, pages 55–127, 1997.

[KP99] E. Koutsoupias and C. Papadimitriou. Worst–case equilibria. *Lecture Notes in Computer Science*, 1563:404–413, 1999.

[Krö97] B. Kröll. *Dynamisch verteilte Wörterbücher*. PhD thesis, ETH Zürich, 1997.

[Kuh73] H. Kuhn. A note on Fermat's problem. *Math. Prog.*, 4:99 – 107, 1973.

[KW94] B. Kröll and P. Widmayer. Distributing a search tree among a growing number of processors. In *Proc. ACM SIGMOD Conference on the Management of Data*, pages 265–276, 1994.

[Lam96] B.W. Lampson. How to build a highly available system using consensus. In Babaoglu and Marzullo, editors, *10th International Workshop on Distributed Algorithms (WDAG 96)*, volume 1151, pages 1–17. Springer-Verlag, Berlin Germany, 1996.

[Lav01] C. Lavers. *Why Elephants Have Big Ears : Understanding Patterns of Life on Earth*. St. Martin's Press, 2001.

[Lic82] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[LMRS99] W. Litwin, J. Menon, T. Risch, and Th. Schwarz. Design issues for scalable availability LH* schemes with record grouping. In *Workshop on Distributed Data and Structures 2 [BDSW99]*, pages 38–55, 1999.

[LN96] W. Litwin and M.-A. Neimat. High-availability LH* schemes with mirroring. In *First Int. Conference on Cooperative Information Systems (CoopIS)*, pages 196–205, 1996.

[LNS93] W. Litwin, M.-A. Neimat, and D.A. Schneider. LH*—A scalable, distributed data structure. *ACM Trans. on Database Systems (TODS)*, 21(4):480–525, 1996. A preliminary version appeared as: LH*—Linear hashing for distributed files. *Proc. ACM SIGMOD Conference on the Management of Data*, 327–336, 1993.

[Lom96] D. Lomet. Replicated indexes for distributed data. In *PDIS: International Conference on Parallel and Distributed Information Systems*, pages 108–119, 1996.

[LS00a] H.B. Lillywhite and R.S. Seymour. Hearts, neck posture and metabolic intensity of sauropod dinosaurs. *Proc. of The Royal Soc. of London, Ser. B*, 267:1883–1887, 2000.

[LS00b] W. Litwin and Th. Schwarz. LH$^*_{RS}$: A high-availability scalable distributed data structure using reed solomon codes. In *SIGMOD Conference on the Management of Data*, pages 237–248, 2000.

[NR95] M. Naor and R.M. Roth. Optimal file sharing in distributed networks. *SIAM J. Comput.*, 24(1):158–183, 1995.

[Och96] D. Ochmanek. Time to Restructure U.S. Defense Forces. *ISSUES in Science and Technology*, Winter 1996.

[O'N83] B. O'Neill. *Semi–Riemannian geometry.* Academic Press, 1983.

[Ost78] L. Ostresh. On the convergence of a class of iterative methods for solving the Weber location problem. *Oper. Res.*, 26:597–609, 1978.

[Par96] L. Parker. On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics*, 10(6), 1996.

[PLN00] A. Di Pasquale, F. Luccio, and E. Nardelli, editors. *Workshop on Distributed Data and Structures 3.* SIROCCO 2000, Carleton Scientific, 2000.

[Pol80] Polybius. *The Rise of the Roman Empire.* Penguin, 1980.

[PPS+01] A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D.S. Taylor, and P. Widmayer. Server Placements, Roman Domination and Other Dominating Set Variants. Technical Report 365, Department of Computer Science, ETH Zürich, 2001.

[PPS+02] A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D.S. Taylor, and P. Widmayer. Server Placements, Roman Domination and Other Dominating Set Variants. In *Proc. TCS 2002.* Kluwer, 2002.

[Pre00] G. Prencipe. Achievable patterns by an even number of mobile robots. Technical Report TR-00-11, Univ. di Pisa, 2000.

[Pre01] G. Prencipe. Corda: Distributed coordination of a set of autonomous mobile robots. In *Fourth European Research Seminar on Advances in Distributed Systems (ERSADS 2001)*, pages 185–190, 2001.

[PS85] F. Preparata and M. Shamos. *Computational Geometry: An Introdution.* Springer–Verlag, New York, 1985.

[PS98] R. Pfeifer and C. Scheier. *Understanding Intelligence.* The MIT Press, Cambridge, MA., 1998.

[RR00] C.S. ReVelle and K.E. Rosing. Defendens Imperium Romanum: A Classical Problem in Military Strategy. *American Mathematical Monthly*, pages 585–594, 2000.

[RS97] R. Raz and S. Safra. A Sub–Constant Error–Probability Low–Degree Test, and a Sub–Constant Error–Probability PCP Characterization of NP. In *Proc. 29th ACM STOC*, pages 475–484, 1997.

[Sch99] A. Schöbel. *Locating Lines and Hyperplanes.* Kluwer Academic Publishers, 1999.

[Smi98] J.R. Smith. *The Design and Analysis of Parallel Algorithms*. Oxford University Press, 1998.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[SS90] K. Sugihara and I. Suzuki. Distributed Motion Coordination of Multiple Robots. In *IEEE Internatiol Symposium on Intelligent Control*, pages 138–143, 1990.

[SS96] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotics Systems*, 13(3):127–139, 1996.

[SSSW01] K. Schlude, E. Soisalon-Soininen, and P. Widmayer. Distributed highly available search trees. Technical Report 364, Department of Computer Sciene, ETH Zurich, 2001.

[SSSW02] K. Schlude, E. Soisalon-Soininen, and P. Widmayer. Distributed Highly Available Search Trees. In *Sirocco 9*, pages 259–274, Andros, Greece, June 2002. Carleton Scientific.

[Ste99] I. Stewart. Defend the Roman Empire! *Scientific American*, pages 94–95, December 1999.

[Sto01] J.A. Storer. *An Introduction to Data Structures and Algorithms*. Springer, 2001.

[SW98] N. Santoro and P. Widmayer, editors. *Workshop on Distributed Data and Structures 1*. IPPS Workshop, Orlando, Florida, Carleton Scientific, 1998.

[SY93] I. Suzuki and M. Yamashita. Formation and agreement problems for anonymous mobile robots. In *31th Annual Allerton Conf. on Communication, Control, and Computing*, pages 93–102, 1993.

[SY99] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[VBW98] R. Vingralek, Y. Breitbart, and G. Weikum. Snowball: Scalable storage on networks of workstations with balanced load. *Distributed and Parallel Databases*, 6(2):117–156, 1998.

[VZ01] Y. Vardi and C.-H. Zhang. A modified Weiszfeld algorithm for the Fermat-Weber location problem. *Math. Prog.*, 90:559–566, 2001.

[Wat98] R. Wattenhofer. *Distributed Counting: How to Bypass Bottlenecks*. PhD thesis, ETH Zürich, 1998.

[Web09] A. Weber. *Über den Standort der Industrien, erster Teil: Reine Theorie des Standorts*. Verlag von J.C.B. Mohr, 1909.

[Wei37] E. Weiszfeld. Sur le point par lequel la somme des distances de n points donnés est minimum. *Tôhoku Mathematics Journal*, 43:355–386, 1937.

[Wes93] G. Wesolowsky. The Weber problem: History and perspectives. *Location Science*, 1(1):5–23, 1993.

[Zwi92] D. Zwillinger. *Handbook of Integration*. Jones and Bartlett, Boston, 1992.

Curriculum Vitae

Konrad August Schlude

date of birth: May 24, 1968
place of birth: Waldshut, Germany
citizenship: Germany

Education
1974–1978: Grundschule Jestetten
1978–1979: 5. Klasse Hauptschule Jestetten
1979–1985: Realschule Jestetten
1985–1988: Technisches Gymnasium Waldshut

Military Service
1988–1989: Obligatory German Military Service.

Studies
1989–1994: Studies at the Albert–Ludwigs–Universität Freiburg.
major: Mathematics
minor: Computer Science
academic title: Diplom–Mathematiker.
1996–2002: PhD student at the Institute of Theoretical Computer Science ETH Zürich.

Work Experience
1995–1996: Collaborator at the Chair of Building Physics, ETH Zürich.
1996–2002: Research and Teaching Assistent at the Computer Science Department, ETH Zürich.

Publications

- E. Kranakis, P. Penna, K. Schlude, D. Taylor, P. Widmayer.:
  Improving Customer Proximity to Railway Stations
  Technical Report 371, CS, ETHZ 2002
- M. Cieliebak, St. Eidenbenz, A. Pagourtzis, K. Schlude:
  Equal Sum Subsets: Complexity of Variations
  Technical Report 370, CS, ETHZ 2002
- A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D. Taylor, P. Widmayer.:
  Server Placements, Roman Domination and Other Dominating Set Variants
  Teoretical Computer Science TCS 2002
  a preliminary version appeared as: Technical Report 365, CS, ETHZ 2001
- K. Schlude, E. Soisalon–Soininen, P. Widmayer:
  Distributed Highly Available Search Trees
  SIROCCO 9, Proceedings in Informatics 13, 259-274, 2002
  a preliminary version appeared as: Technical Report 364, CS, ETHZ 2001

- T. Erlebach, M. Gantenbein, D. Hürlimann, G. Neyer, A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D. Taylor, P. Widmayer:
  On the complexity of train assignment problems
  ISAAC 2001, LNCS 2223, 390-402, 2001
  a preliminary version appeared as: Technical Report 363, CS, ETHZ 2001

- K. Schlude:
  Bemerkung zu beschränkt homogenen Funktionen
  Elem. Math. 54, 1, 1999, 30-31