On Locality and Related Problems
Communicating, Computing, Exploring

A dissertation submitted to

ETH Zurich
Swiss Federal Institute of Technology

for the degree of Doctor of Sciences

presented by

Elias Vicari
Dipl. Math. ETH
born April 14, 1980,
citizen of Rovio, Switzerland

accepted on the recommendation of

Prof. Dr. Peter Widmayer, ETH Zurich
examiner

Prof. Dr. Markus Bläser, Saarland University
co-examiner

Prof. Dr. Subhash Suri, University of California at Santa Barbara
co-examiner

2008
Abstract

Locality is a peculiarity which is exhibited by many problems appearing in the most disparate domains of theoretical computer science and mathematics. In this thesis, we demonstrate this assertion by studying three classes of problems that are related to the task of communicating, computing and exploring, respectively. By a local problem, we understand a problem which is easily solved if the problem solver(s) would have access to the whole instance at once. However, the difficulty is that the problem solver is allowed to initially look only at a small portion of the instance, and at every point of the resolution process she gets a small piece of new information that depends on her choices and on the model she is operating with. The problem solver incurs a unit cost when she gets a new information, but can work with the information she already has for free.

In the first part of this thesis we focus on the process of communication. We introduce and study the so called algebraic communication complexity of a multivariate polynomial or rational function. In this model two or more players aim to evaluate a rational function according to some input. However the input is distributed among the players. Hence, they exchange messages in such a way that eventually a player is able to correctly determine the value of the function at the given input. Our goal is to determine the least number of messages that are required to solve the task for every given input. We propose different techniques that establish lower bounds for this complexity measure. We apply these results to show that a broad class of multicast cost sharing mechanisms – namely truthful budget-balanced mechanisms – need to send many messages in a network in the worst case during the resolution process.

In the second part we deal with the realm of computation. We focus on distributed algorithms for wireless sensor networks. In such a network, every node is a computational unit that performs some actions. However, its view is limited to the set of nodes that can be reached through direct communication with its wireless equipment. As an algorithm progresses, each node gathers more information about the network topology. We start with the observation that many such algorithms require to solve some graph-theoretical problem during their execution. Hence a good understanding of the locality of such problems is much needed. In this thesis we design
and analyze distributed algorithms for the Maximal Independent Set Problem, the Minimum Connected Dominating Set Problem and the Small Connected Spanning Subgraph Problem.

Lastly, we consider robot systems of minimalistic capabilities that operate in a polygon. A robot is initially placed inside a polygon of unknown topology, hence its view of the world is local. The robot gathers further information through an exploration process. In this setting, we consider devices of different sophistication and we study how robots equipped with a different set of these devices compare to each other in terms of the set of problems that they can solve or cannot solve. In particular, we first investigate what representation of the polygon a robot can devise when equipped with certain devices. Further, we study other problems typical in the domain of computational geometry, such as recognizing the simplicity of a polygon or whether a given angle is convex or reflex. We also study the so called Art Gallery Problem and show how a group of ⌊n/3⌋ such minimalistic robots autonomously deploy to vertices of a polygon of n vertices in such a way that every point of the polygon is visible to one of them. Finally we investigate the problem of counting the number of targets which are a set of indistinguishable points spread in the polygon. We consider different models and we show that in some cases the robots must be endowed with some powerful devices to correctly accomplish the task.
Riassunto

La località è una peculiarità esibita da diversi problemi che nascono nei campi più disparati dell’informatica teorica e della matematica. In questa tesi, dimostreremo questa affermazione studiando tre classi di problemi che sono legati alle operazioni della comunicazione, della computazione e dell’esplorazione. Quando parliamo di un problema locale, ci riferiamo a un problema che può essere risolto facilmente se il (i) risolutore(i) ha un accesso globale all’istanza. Tuttavia la difficoltà risiede nel fatto che inizialmente il risolutore può guardare solo a una piccola porzione dell’istanza e in ogni momento del processo di risoluzione riceve nuove informazioni in un modo che dipende dalle proprie scelte e dal modello con cui sta operando. Il risolutore paga un costo unitario quando riceve nuove informazioni, mentre può operare liberamente con le informazioni di cui già dispone.

Nella prima parte di questa tesi ci concentriamo sul processo della comunicazione. Introduciamo e studiamo la complessità di comunicazione algebrica di un polinomio o di una funzione razionale multivariata. In questo modello due o più giocatori intendono calcolare il valore di una funzione razionale rispetto al loro argomento. Tuttavia l’argomento è distribuito tra i giocatori. Per questo motivo i giocatori si scambiano dei messaggi in modo tale che un giocatore sia infine in grado di calcolare il valore della funzione rispetto all’argomento dato. Il nostro obiettivo è di determinare il numero minimo di messaggi necessari per portare a termine l’operazione per ogni argomento. Proponiamo diverse tecniche che determinano dei limiti inferiori per questa misura di complessità. In seguito applichiamo questi risultati per dimostrare come importanti meccanismi per la suddivisione dei costi, ossia meccanismi veritieri e bilanciati, necessitino di spedire molti messaggi nella rete durante il processo di risoluzione nel caso peggiore.

Nella seconda parte ci occupiamo del processo della computazione. Ci concentriamo su algoritmi distribuiti per reti di sensori senza fili. In questo tipo di reti ogni nodo è un’unità computazionale che esegue alcune operazioni. La conoscenza iniziale di un nodo della rete è limitata all’insieme dei nodi che sono direttamente raggiungibili mediante la capacità di comunicazione senza fili. Durante l’esecuzione di un algoritmo ogni nodo raccoglie informazioni sulla topologia della rete. Si osserva che spesso tali algoritmi necessitano
di risolvere problemi classici della teoria dei grafi durante la loro esecuzione, per cui la comprensione della località di tali problemi è di determinante importanza. In questa tesi ideiamo e analizziamo algoritmi distribuiti per la risoluzione del Problema del Massimo Insieme Indipendente, del Problema del Minimo Insieme Dominante e Connesso e del Problema del Piccolo Sottografo Connesso e Ricoprente.

Infine consideriamo sistemi di robot con capacità minimalistiche che operano in un poligono. Un robot è posto inizialmente all’interno di un poligono di cui ignora la topologia, perciò la sua visione del mondo è locale. Il robot raccoglie nuove informazioni attraverso un processo di esplorazione. In questo modello consideriamo dispositivi di diversa sofisticazione e confrontiamo tra loro i robot che sono equipaggiati con diversi dispositivi, identificando i problemi che sono in grado o meno di risolvere. In particolare studiamo quale rappresentazione del poligono è in grado di costruire un certo robot se equipaggiato con un determinato insieme di dispositivi. Inoltre consideriamo altri problemi tipici del campo della geometria computazionale, quali il riconoscere se un poligono è connesso in modo semplice o multiplo, o se un angolo è convesso o concavo. Successivamente prendiamo in esame il Problema della Galleria d’Arte e dimostriamo come un gruppo di ⌊n/3⌋ robot sono in grado di disporsi autonomamente su dei vertici di un poligono di n vertici in modo tale che ogni punto del poligono sia visibile ad almeno uno di loro. Infine ci occupiamo del problema di contare il numero di obiettivi, i quali sono un insieme di punti indistinguibili sparsi nel poligono. Anche in questo caso consideriamo diversi modelli e mostriamo che in alcuni casi i robot devono essere equipaggiati con dispositivi importanti per portare a termine il compito correttamente.
Acknowledgments

These days I am having interviews with companies to make the first steps in the “post-PhD time”. A very frequent question I get is “how does your dream job look like?”. Well, if I simply look back at the last few years of my life, my job as a PhD student was pretty much close to that. For instance ETH as a university and employer is just great: it provided me with far more than what I needed, including an incomparable research environment, trips around the world, an “aquarium-office”, and two essential mental-health-preserving-and-coffee-substitute accessories like the legendary kicker table and the pool table – only to mention a few things. Even more importantly I had the utmost pleasure to deal with a group of outstanding and talented persons every day. Regardless whether I needed advice or support for research matters (read, Lemma 2...), I was looking for two or three more players to play kicker with, or I had troubles with a shell script to automate an operation I executed twice in my life, I could always find a colleague and friend with the respective skills eager to help me (i.e., a counterexample-prone researcher, a skillful middle-line-lifting legend, and a 100-point geek, respectively).

During my PhD I was member of two research groups, so I am happy to have the opportunity to thank twice as many persons. First of all, my most profound gratitude goes to Markus Bläser and Peter Widmayer, my advisors. You made this dissertation possible in the first place, but not only. The problems you have come up with and the discussions we had to (try to) solve them contributed in a decisive way to shape this thesis and my way of being a researcher. Another very special thank you goes to Subhash Suri: the problem of mobile robots that you brought to Zurich a couple of years ago eventually became the larger part of this thesis. I am also deeply indebted to you for having accepted to be a co-examiner for my defense and for being such a great host in Santa Barbara.

To the reader that might have spotted typos in this dissertation: I cowardly decline any responsibility. Any complaints can be directed to Matúš “ultimate” Mihalák who admirably proofread the text with zealous dedication. Thank you for that and for sharing crazy ideas that led to last minute papers.

A big thank you for the great time at ETH to the ladies and gentlemen sitting “upstairs”...
To end the acknowledgments “on-the-job” I am grateful to SNF and MICS for the grants that supported this thesis.

My life around the PhD has been undoubtedly influenced by many friends. We shared many inspiring experiences like the Nankatsu epic, the long battles in tennis courts, and the enthusiastic complaining in newspapers. But more than that, you were irreplaceable for recalling me that eating breaded bananas in curry sauce would ban me for crossing the Gotthard tunnel again. Thank you for keeping me on track.

This thesis has been definitely enriched by the picture on the cover by Tita Carloni. Thank you Tita for the marvelous piece of art.

Per ultimo, ma non per questo meno importante, ringrazio di cuore la mia famiglia – in special modo mio padre Danilo, mia madre Sonia, mio fratello Alessio e mia nonna Maria. Mi avete permesso di seguire il mio sogno nato forse prima delle elementari e che mi ha portato a essere ció che sono ora, con i miei pregi e i miei difetti.

Un’altra persona è stata fondamentale per me nel mio percorso. Grazie, Didi, per essermi sempre stata accanto e aver contribuito nei modi più diversi e incredibili a questa tesi.
Contents

1 Preface 1

II Communicating 7

2 Algebraic Communication Complexity 13
   2.1 Introduction and Motivation 13
      2.1.1 Preliminaries 15
      2.1.2 The Multicast Cost Sharing Problem 22
   2.2 Related Work 25
   2.3 Summary of Results 26
   2.4 Definitions and Model 27
   2.5 One-way Communication 30
   2.6 Two-way Communication 33
      2.6.1 Rank Bound 33
      2.6.2 Substitution Method 36
      2.6.3 Dimension Bound 37
   2.7 Decision Problems 39
      2.7.1 Equality Tests 41
      2.7.2 Comparisons 42
   2.8 Nondeterministic Communication Complexity 43
   2.9 Multi-party Communication Complexity 46
   2.10 Applications 49
## Contents

8.3.1 The Case of Simply-Connected Polygons . . 135  
8.3.2 The Case of Multiply-Connected Polygons . 143  
8.4 Comparing the Robot Models . . . . . . . . . . . 148  
8.5 Equivalent Robots: An Example . . . . . . . . . 153  
8.6 Open Problems . . . . . . . . . . . . . . . . . . . 157

9 The Art Gallery Problem 159  
9.1 Introduction and Motivation . . . . . . . . . . . . 159  
9.2 Summary of Results . . . . . . . . . . . . . . . . . 161  
9.3 Polygon Triangulation . . . . . . . . . . . . . . . . 161  
9.4 Guarding an Art Gallery . . . . . . . . . . . . . . . 167  
9.5 Open Problems . . . . . . . . . . . . . . . . . . . 173

10 The Target Counting Problem 175  
10.1 Introduction and Motivation . . . . . . . . . . . . 175  
10.2 Summary of Results . . . . . . . . . . . . . . . . . 178  
10.3 The Friendly Environment . . . . . . . . . . . . . 179  
10.4 The Hostile Environment . . . . . . . . . . . . . . 180  
10.4.1 Partition of the Polygon and Counting . . . . 180  
10.4.2 A Tight Approximation . . . . . . . . . . . . . 182  
10.4.3 More Power to the Robots . . . . . . . . . . . 189  
10.5 Open Problems . . . . . . . . . . . . . . . . . . . 195
Chapter 1

Preface

*Be amusing: never tell unkind stories; above all, never tell long ones.* – Benjamin Disraeli (February 20, 1874 - April 21, 1980), British Prime Minister and literary figure.

The scope of this chapter is to introduce the reader to the subject of this thesis. To this end, let us consider the following three toy problems. In the first problem, two kids are preparing for a math test about equations taking place at the following day. To train the subject, they are solving a long exercise sheet at home. They are dedicated students, so they agreed to compare the solutions to assess their degree of preparation. Unfortunately, they live in different towns and they can communicate only with short messages on their mobile phones. Is there a smarter way, in terms of number of required messages, to compare the results than sending independently the solution to each equation as a distinct number?

For the second problem, suppose that you are responsible for the choreography at the stadium for the match of your favorite team. Because of the team colors, you decide that all the supporters in the stadium must wear either a white or a red T-shirt. Every supporter comes to the stadium with a T-shirt of each color and decides which one to wear according to the color of the T-shirt of the supporters that sit near to her. The choreography demands that no two supporters that sit nearby both wear the white T-shirt. However if a supporter has no neighbors with the white T-shirt, then she has to wear the white one. Clearly, every supporter can talk to the supporters seated close to her
to take a decision, but not to those seated afar. How fast can each supporter figure out which T-shirt to wear?

The last problem we want to mention is the following. Imagine to be Theseus that wants to leave Daedalus’ labyrinth after having killed the Minotaur. According to the legend, the problem was solved by Theseus with the help of Ariadne that provided him with a thread to retrace his steps. If Ariadne would have not thought about a thread, would it be still possible for Theseus to leave the labyrinth? If he should be asked to enter the labyrinth again, can he find the shortest path between any two points of it without using a ruler?

Admittedly, the described problems are a loose interpretation and adaptation of established theoretical models. Respectively, the models we are referring to are the algebraic communication model, the message-passing model, and the polygonal visibility model in computational geometry.

Even though the three problems are seemingly unrelated, they are in fact linked by a fundamental feature – the locality. The locality is the main topic of this dissertation. The noun locality is overloaded and has undoubtedly different meanings within the domain of computer science. Hence the first task is to make clear what we mean when we talk about local problems or a local way to solve a problem in its broad sense. In the aforementioned problems there are some agents – students, supporters, and Theseus – that act as problem solvers. Later in this thesis, we are going to talk about players (Part I), sensors or nodes (Part II) and robots (Part III), respectively. The instance of a problem that a given agent solves admits a global solution, i.e., a solution that typically can be easily computed if the agent could see the whole instance at once. However, the agent has access to a local portion of the instance at a time. For instance, one can think of the problem of recognizing a painting when one is allowed to look only at a small area of it at a time. What makes our setting special is the way how an agent can move inside an instance, i.e., which new information she can get at every new step of the execution of the solution process. In fact, an agent is an active entity that somehow interacts with the given instance: the agent has a set of feasible actions that are dictated by the model. The effect of those actions is to modify the current view that the agent has of the instance, i.e., the portion of the instance she is looking at. In other words, if an agent requires new information to complete the task, only certain new
portions of the input can potentially be accessed and this depends on the type of actions that the agent is allowed to execute. We illustrate this principle with the example of Theseus. Obviously, the shape of the labyrinth is initially unknown to Theseus and this is his input. At any point of time he is in a corridor (his local view of the instance). His feasible actions are to walk along corridors and to pick new paths at branches. With these actions he learns a portion of the labyrinth. The sequence of corridors that Theseus visits in his attempt to find his way to the exit is not arbitrary: it depends on his choices, on the topology of the labyrinth, and the fact that he is only allowed to walk along corridors (as opposed to, say, drill holes in the walls).

The set of problems that we consider in this thesis also share another characteristic. Every agent can use the information she already has for free. The costly aspect is to access new information – for instance walking along a corridor. We study the hardness of the problems and we analyze the proposed algorithms according to this complexity measure: every time that an agent gathers a new piece information according to the model, she incurs a cost of one. In this way we assess the locality of each problem.

The reader should easily recognize the characteristics of locality described above in the models introduced in each part. To lighten the presentation of this thesis, we allow ourselves not to point out formally and in detail what are the local view and the set of feasible actions in each model.

The three toy problems mentioned above have not been chosen arbitrarily. In the first problem, the issue is to find an efficient communication protocol between the kids. In the problem of the supporters, we care about a fast computation of a solution. In turn, Theseus solves an exploration problem to locate the exit of the labyrinth. The operations of communicating, computing and exploring are of fundamental interest in computer science. We devote a part of this dissertation to each of these operations. In doing so, we select a family of intriguing problems that exhibit some different facets of locality in the domains of communication, computation and exploration. The problems are picked from the realms of distributed algorithmic mechanism design, distributed algorithms for wireless sensor networks, and algorithms for robotic systems, respectively – three extremely active areas in computer science.
Overview and Results

In this section we give a high-level overview of the results of this dissertation.

Part I

In Part I we deal with the model of algebraic communication complexity. In this model, two players hold a portion of the input of a multivariate function. They aim to evaluate the function value according to their input. To do so they exchange messages to ensure that eventually a player is able to evaluate the function. We are interested in establishing the least number of required messages. We develop algebraic techniques to analyze a given function in this respect.

Further, we establish the link between the algebraic communication complexity of a function and problems of distributed algorithmic mechanism design. This field deals with the problem of implementing a mechanism in a network. In every node of the network sit some rational agents, which selfishly care about their personal happiness. The purpose of a mechanism is to incentive the agents to act in such a way that leads to the social happiness. The mechanism requires to collect and deliver information to the agents by using the network infrastructure. We are interested in studying the overhead that the network incurs in doing this operation. We show that a large class of mechanisms for the multicast cost sharing problem, namely budget-balanced mechanisms, require to send unacceptably many messages in the worst case.

Part II

The topic of Part II is the design of distributed algorithms for problems related to wireless sensor networks. The technological innovation is pushing the growth of wireless sensor networks at a considerable pace. The distributed nature and resource constraints of such systems offer a source of interesting theoretical problems.

Many algorithms for wireless sensor networks call some basic subroutines that solve classical graph-theoretical problems. In Part II we study three such problems. We describe distributed algorithms that efficiently compute a Maximal Independent Set, an approximation scheme for a Minimum Connected Dominating Set, and an ap-
proximation scheme for a Small Connected Spanning Subgraph, respectively. We prove that the algorithms that solve the first two problems are efficient when the input graph is a Growth-Bounded graph. Growth-Bounded graphs are a graph model that is widely accepted to model wireless sensor networks. The algorithm for the latter problem is asymptotically optimal for general graphs.

**Part III**

In part Part III we study robot systems. We define and study a very simple robot model that operates in a polygonal environment. We successively endow the robot with sensors of different sophistication. We try to pinpoint the set of problems that a robot with a given device can solve, but which cannot otherwise. The problems we look at include many classical problems of computational geometry. A first fundamental issue that we study is related with the representation that a robot can build of its workspace. Intuitively, devices like a length-measuring device enable the robot to produce a precise representation of a polygon, while a robot that cannot distinguish a vertex from the other produces a poor representation. We try to understand whether it is true that the higher is the level of the sophistication of the device, the better is the representation. While we give hints that the statement is mostly correct, we identify a set of seemingly difficult problems that can be equally solved by robots of different sophistication. For instance, we show that for determining the Euclidean shortest path between two vertices of a polygon it is not necessary to be able to measure lengths.

Further we show how a group of \( \lfloor n/3 \rfloor \) robots can autonomously learn and successively self-deploy to selected vertices of a polygon with \( n \) vertices in such a way that every point of the polygon is visible to at least a robot. In the computational geometry jargon, we say that they solve the Art Gallery Problem.

Finally, we consider the problem of counting a set of indistinguishable points that are spread inside the polygon (targets). In this context, we distinguish two cases: the friendly and the hostile case. In the friendly case, the robot is allowed to move on the targets. Under this assumption, the problem is easy for a robot. In the hostile scenario a robot cannot approach the targets. Unless we consider robots with additional devices, we show that the number of targets can only be approximated in general.
Part I

Communicating

Algebraic Communication Complexity
Prologue

Florence, 1503 – Leonardo da Vinci is painting Mona Lisa, with whom he often has interesting chats on various subjects.

Mona Lisa: “You have never told me why your urban renovation plans for Milan have never been realised. Didn’t Ludovico Sforza urge upon you?”

Leonardo da Vinci: “Yes, he did. 1485 was a very difficult period indeed. The Black Death was devastating Milan. Novel ideas to ameliorate the hygienic conditions and water supply of the city were really pressing.”

Mona Lisa: “I’m absolutely certain that you had brilliant ideas with the aim of alleviating the aftermaths of the plague. Hadn’t you?”

Leonardo: “I thought quite long about this. In my opinion, one of the decisive issues of the present city plan is the extreme density of the buildings. To possibly do better, I envision a city organised on more floors, each almost independent to the others and connected by staircases. Roads are less winding and decisively wider. Floodgates along a river and artificial channels have to enable a capillary ship navigation at the underground level and allow the elimination of garbage.”

Mona Lisa: “The idea sounds exciting. Unfortunately I guess that the construction time and costs of such a city are overwhelming and this is why it hasn’t been realized yet. Am I right?”

Leonardo: “Yes, you are. There are many issues, among others the ones that you mention. There are additional, very subtle and curious issues of theoretical nature as well.”

Mona Lisa: “What do you mean?”

Leonardo: “Along with the actual city plan, I was also thinking about the laws and rules that are supposed to govern the new city. For instance, the underground canals are expensive to maintain. However, I think that they are of immense interest, above all to wealthy people, because in this way they can supply their
buildings in a rational and quick manner. I need to find a reasonable way to make them split the maintenance costs.”

**Mona Lisa:** “Uhm, I don’t really see the problem.”

**Leonardo:** “There are many problems indeed. Let me demonstrate this to you. If you own a big building, you should be responsible of the maintenance of the portion of the canal that branches from the river until your place. However other people might be using some parts of your canal. How do you share the costs with them?”

**Mona Lisa:** “Let me think for a minute. What about subdividing the canals into small pieces and charge equally every user of that piece.”

**Leonardo:** “Yes, I agree. This is a fair way of splitting the costs and has many advantages. You can extend it in such a way, that every person can declare a cost she is willing to pay to use the canal. You then compute the prices that you need to raise to cover the costs according to every person’s wealth. Eventually she will use the canal only if it turns out to be profitable for her and you. I’m pretty sure that it is not possible that rich people might benefit by pretending to be less wealthy.”

**Mona Lisa:** “This sounds like a great idea that even takes into account social issues. What it is more, I don’t see the problem you were talking about. Why are you so pessimistic?”

**Leonardo:** “Think of a practical implementation. You can either ask everybody to come to a public place and disclose the value that they assign to the usage of the canal or you can send functionaries to every building to collect these values.”

**Mona Lisa:** “Still looks fine to me.”

**Leonardo:** “In the first case, imagine to be the accountant that evaluates the thousands of necessary mathematical operations with the abacus, which is slow and limited to additions, subtractions, multiplications and divisions. This doesn’t sound like a conceivable scenario to me.”

**Mona Lisa:** “Ok, let’s do it according to the second scenario that you mentioned.”
**Leonardo:** “In order to be any better, the functionaries should work in a hierarchical way. A team of functionaries are responsible for the districts. The districts are combined in larger areas and every area has a chief functionary responsible for it, and so on, up in the hierarchy. Of course, in every level of the hierarchy the data has to be aggregated. Otherwise we are again left with a list of numbers and operations which is infinite in size for every practical purpose. As before, the aggregation has to be accomplished solely with the operations that an abacus can perform. Unfortunately, I couldn’t find any smart way to do this so far. I wonder if it is at all possible to save time with respect to the first scenario.”

**Mona Lisa:** “I’m sure that if a way would exist, you would have already found it. I’m also feeling quite pessimistic now.”

*Note from the author.* Leonardo’s urban plans for Milan have never been realized. However, many of his ideas were retrieved and accomplished in different historical moments. For further information, see [114].
Chapter 2

Algebraic Communication Complexity

The Internet is the most important single development in the history of human communication since the invention of call waiting. – David “Dave” Barry (born July 3, 1947), bestselling American author and Pulitzer Prize-winning humorist.

2.1 Introduction and Motivation

Locality as we intend it in this thesis certainly models the domain of communication in its broad sense. Imagine for instance the situation of a group of chatting people. The topic of the discussion is the work of Leonardo da Vinci during his career. Barbara was in Paris last month and is competent about the Mona Lisa, Marianna is from Milan and knows everything about the The Last Supper, and so on. After half an hour of an interesting exchange of information, everybody has a chance to learn something new and exciting from the others. The knowledge that every member of the group has gathered depends on what the other members have disclosed. Of course, in a real-life scenario a person might ask explicitly for some piece of
information; nevertheless we feel that the process of communicating exhibits characteristics of locality.

In this chapter we introduce a communication model and we study the cost of exchanging information in this model. In this setting, two (or more) players share a multivariate rational function and each player holds a disjoint portion of the input. A communication step consists of a player sending a message to another player. We restrict messages to be rational functions of the variables that the player owns and of the messages she has received so far. Mainly, our goal is to investigate the least amount of messages that are necessary to enable a player to compute the function value according to the input. Ideally, we would like to be able to look at a function and determine with a good precision how hard it is to evaluate it in the described distributed scenario. We refer to this aspect of a function as its algebraic communication complexity.

The reason to investigate the algebraic communication complexity is of independent interest. However, in this chapter we describe a motivating application. We study a problem of Mechanism Design: the Multicast Cost Sharing Problem. We analyse the problem in a distributed scenario, where communication plays a fundamental role. By applying the bounds established in the algebraic communication model, we will show that the communication burden of any implementation makes impractical any solution to the Multicast Cost Sharing Problem that fulfills some properties. The questions about the feasibility of mechanisms in a distributed model are addressed by a branch of Mechanism Design called Distributed Algorithmic Mechanism Design. Distributed Algorithmic Mechanism Design is the art of designing, analyzing, and implementing mechanisms in a distributed setting. Many mechanisms, like marginal cost and Shapley value [89], can be computed by arithmetic operations ("+", ".", and "/"), equality tests ("="), and comparisons ("<"). These are exactly the operations that are allowed by the players to compose messages in our setting. Furthermore, algebraic operations are the object of the study of algebraic complexity theory, a well-developed area of complexity theory. The success story of algebraic complexity theory is promising for our objectives, since many ideas and tools apply to our case as well.

To pursue our goal, we develop a model upon an algebraic structure. This approach is profoundly different to other approaches in the
2.1. Introduction and Motivation

area that assume analytical properties of the involved functions, like
continuousness and differentiability. Even though our model restricts
the power of the messages that the players exchange and the family
of the objective functions, we believe that this model of computation
fits better in the context of possible applications as the power of the
involved messages is realistically bounded. An additional advantage
of this approach is that we can introduce new features in a natural
way. For instance by allowing equality tests and comparisons, we are
able to deal with non-continuous functions. We can investigate decision problems too, where the goal is not to compute the function, but
to decide whether a given input lies in its zero-set. Or we can intro-
duce nondeterminism in a flavor recalling the usual nondeterminism
of computational complexity.

As mentioned above, the key observation is that natural mecha-
nisms solving the Multicast Cost Sharing Problem can be expressed as
rational functions of the input given to the mechanism. Moreover, the
input is distributed among players that reside in nodes of a network.
We demonstrate how to apply the machinery developed in the alge-
braic communication model in this case – in particular lower bounds.
As a consequence we show that it is not possible to implement a dis-
tributed mechanism for the Multicast Cost Sharing Problem in such a
way that (1) it fulfills some well-defined and desirable properties and
(2) the number of messages sent in the network during its execution
is moderate.

2.1.1 Preliminaries

The aim of this section is to briefly review some important concepts
that are used in the first chapter of this thesis.

Notions from Algebra

Algebra is considered one of the main branches of pure mathematics.
It attracts mathematicians as early as the 9th century AD. It studies
mathematical structures which are defined by axioms and the rela-
tionships among them. In the rest of this section, we recall the basic
definitions from algebra that are necessary to the non-familiar reader
to be able to understand the first chapter. Since we employ algebra
only in a part of the first chapter, we feel that a rigid and formal in-
Fields and Rings An important algebraic structure is a field – examples are the field of real numbers $\mathbb{R}$ and the field of complex numbers $\mathbb{C}$. Roughly speaking, a field $k$ is a set equipped with two operations, usually denoted by “$+$” (addition) and “$\cdot$” (multiplication), that fulfill some basic requirements, including commutativity, associativity and distributivity.

We review some notions that are linked with the concept of field. Let $0$ denote the neutral element of the field $k$ with respect to the addition and $1$ the neutral element with respect to the multiplication. We call the smallest non-zero $n \in \mathbb{N}$ such that

$$n \cdot 1 = 1 + 1 + \ldots + 1 = 0$$

the characteristic of the field $k$. If such a $n$ does not exist, we say that $k$ has characteristic zero. In this thesis we work only with fields of characteristic zero, like $\mathbb{R}$. To get the right intuition about the results of this chapter, it is safe in general to assume without loss of generality that the field $k$ is $\mathbb{R}$, or $\mathbb{C}$ if the field is required to be algebraically closed – see below.

Rings are similar to fields, but the elements do not necessarily have an inverse element with respect to the multiplication operation “$\cdot$”. Under the term ring we tacitly understand a commutative ring, where both operations are commutative. An example of a ring is the set of integer numbers $\mathbb{Z}$ with the natural operations.

Polynomials and Rational Functions The ring of polynomials in the variable (or indeterminate) $X$ over the field (or ring) $k$ is denoted by $k[X]$ and is the set of abstract functions (called polynomials) that can be written in the form

$$\sum_{i \in \mathbb{N}} a_i X^i$$

where all, but finitely many, coefficients $a_i \in k$ are zero. As customary, we define $X^0 := 1$ and $X^i := X \cdot X^{i-1}, i > 0$. The operations
defined on \( k[X] \) are the classical polynomial addition and multiplication. The degree of a polynomial \( f \in k[X] \) is the maximal index \( i \) with \( a_i \neq 0 \).

The definition extends naturally to more variables. If the variables are \( X_1, \ldots, X_n \), we define the ring of polynomials in \( n \) variables as \( k[X_1, \ldots, X_n] = \ldots((k[X_1])[X_2]) \ldots[X_n] \). We denote it by \( k[X] \), if we set \( X = (X_1, \ldots, X_n) \). The context will avoid confusion with the ring of polynomials in one variable. A polynomial \( f \in k[X_1, \ldots, X_n] \) is written as

\[
f = \sum_{\sigma \in \mathbb{N}^n} a_\sigma X^\sigma
\]

where all, but finitely many, coefficient \( a_\sigma \) are zero and we write \( X^\sigma := X_1^{\sigma_1} \cdot \ldots \cdot X_n^{\sigma_n} \), for \( \sigma = (\sigma_1, \ldots, \sigma_n) \). Every summand of the above sum is called a monomial. In this case, the degree of \( f \) is the maximum of \( \sigma_1 + \ldots + \sigma_n \) among the \( \sigma \in \mathbb{N}^n \) with non-zero \( a_\sigma \). A polynomial \( f \) is said to be irreducible, if from the equation \( f = g \cdot h \), \( g, h \in k[X] \) it follows that either \( g \in k \) or \( h \in k \). A field \( k \) is said to be algebraically closed, if for every non-constant polynomial \( f \in k[X] \) it exists an element \( x \in k \) such that \( f(x) = 0 \).

The field of rational functions in the variable(s) \( X \) is denoted by \( k(X) \). Note the difference in the notation with the ring of polynomials \( k[X] \). Loosely speaking, a rational function is the quotient \( p/q \) between two polynomials \( p, q \in k[X], q \neq 0 \).

**Field Extensions and Transcendence Degree** Let \( k_1, k_2 \) be two fields with the same operations for which \( k_1 \subset k_2 \) holds.

Then we call \( k_1 \subset k_2 \) a field extension. An example is the extension \( \mathbb{Q} \subset \mathbb{R} \).

Let \( k_1 \subset k_2 \) be a field extension and let \( \lambda \in k_2 \). The element \( \lambda \) induces a field extension \( k_1 \subset k_1(\lambda) \), where \( k_1(\lambda) = \{ f(\lambda) \mid f \in k_1(X) \} \). Notice that indeed \( k_1(\lambda) \) is again a field. Further, \( \lambda \) is said to be algebraic over \( k_1 \), if there is a polynomial \( f \in k_1[X] \) such that \( f(\lambda) = 0 \), or, in other words, if \( \lambda \) fulfills an algebraic equation

\[
a_0 \lambda^n + a_1 \lambda^{n-1} + \ldots + a_n = 0,
\]

for some coefficients \( a_0, \ldots, a_n \in k_1 \). For instance, \( \sqrt{2} \in \mathbb{R} \) is algebraic over \( \mathbb{Q} \) because \( \sqrt{2} \) belongs to the kernel of the polynomial

\[1\]Throughout this thesis, we do not understand the symbol “\( \subset \)” with its strict sense, i.e., “\( \subset \)” and “\( \subseteq \)” have the same meaning.
$X^2 - 2 \in \mathbb{Q}[X]$. $k_1 \subset k_2$ is an algebraic field extension if every element of $k_2$ is algebraic over $k_1$.

Conversely, a system $\Phi = (\lambda_1, \ldots, \lambda_r) \in k_2^r$ is said to be algebraically independent or transcendental over $k_1$ if from an equation $f(\lambda_1, \ldots, \lambda_r) = 0$, $f \in k_1[X_1, \ldots, X_r]$, it follows that $f = 0$. Otherwise $\Phi$ is a system of algebraically dependent elements.

Let $\Phi \in k_2^r$ be algebraically independent over $k_1$. $\Phi$ is said to be a transcendence basis if $k_2$ is algebraic over $k_1(\Phi)$. Take a maximal system of algebraically independent elements of $k_2$, i.e., a system $(\lambda_1, \ldots, \lambda_r) \in k_2^r$ of algebraically independent elements with the property that $(\lambda_1, \ldots, \lambda_r, x)$ is a system of algebraically dependent elements for every $x \in k_2$. It is well-established that every such system forms a transcendence basis of $k_1 \subset k_2$. Consequently, every field extension admits a transcendence basis. Moreover, it can be shown that every transcendence basis has the same cardinality, so we define the transcendence degree of a field extension $k_1 \subset k_2$, denoted by $\text{tr} \deg_{k_1} k_2$, to be the size of a transcendence basis for $k_2$ over $k_1$. For instance, if $X$ is a variable, then $\text{tr} \deg_k k(X) = 1$ for every field $k$ for which $X$ is algebraic independent.

To a reader not familiar with these concepts, it is useful to think of a field extension in terms of a vector field $V$, to which we add new independent vectors to get a new vector field $V'$. $V$ is a subspace of $V'$ and we express its “relative dimension” by the number of linearly independent vectors that belong to $V'$ but not to $V$. This is of course an invariant characterizing the dimension of $V$ over $V'$ and corresponds to the concept of transcendence degree. Other interesting properties that one can guess with this analogy with linear algebra indeed hold. For instance, if $k_1 \subset k_2 \subset k_3$ then $\text{tr} \deg_{k_1} k_3 = \text{tr} \deg_{k_1} k_2 + \text{tr} \deg_{k_2} k_3$.

**Affine and Projective Varieties** Varieties are the objects that are investigated by the classical algebraic geometry. Let $k$ be an algebraically closed field, $X = (X_1, \ldots, X_n)$ and $f_1, \ldots, f_r \in k[X]$. The set of points of $k^n$ for which all the polynomials $f_1, \ldots, f_r$ vanish specifies a manifold. This manifold is called an affine algebraic set or affine variety and is denoted by $V = V(f_1, \ldots, f_r) \subset k^n$. A nonempty affine algebraic set $V$ is called irreducible if it cannot be written as the union of two proper algebraic subsets. In the literature it is often the case that only irreducible algebraic sets are called va-
2.1. Introduction and Motivation

Given an arbitrary closed field $k$, the affine algebraic sets are precisely the closed sets that induce the so-called Zariski topology.

The dimension of an irreducible variety $V$ is equal to the number of algebraically independent rational functions defined on $V$. The polynomials that define the variety induce some dependencies between the variables. To illustrate this, we assume for simplicity that $V = V(f)$, for a $f \in k[X]$. A rational function $u = p/q$ is defined on $V$ if $q$ is not identical zero on $V$, i.e., if $f$ does not divide $q$. Two rational functions $u = p/q$ and $u' = p'/q'$ are equal on $V$, if the polynomial $p\cdot q' - q\cdot p'$ is divisible by $f$ (and hence is identical zero). The function field $k(V)$ is the field of distinct rational functions on $V$. Formally, $\text{tr.deg}_k k(V)$ is precisely the dimension of $V$. The dimension of an arbitrary variety is equal to the maximal dimension of its irreducible algebraic subsets.

Analogously, we define the projective varieties in the projective space. First, we define the concept of a projective space. Let $V$ be a vector space of $n + 1$ dimensions over the field $k$. The set of 1-dimensional vector subspaces, i.e., lines, of $V$ defines the $n$-dimensional projective space $\mathbb{P}^n = \mathbb{P}^n(k)$. In other words,

$$\mathbb{P}^n := \left( k^{n+1} \setminus \{0\} \right) / \sim,$$

where, for two non-zero elements $x, y \in k^{n+1}$ we have $x \sim y$ iff there exists a $0 \neq \lambda \in k$ such that $x = \lambda y$ (the multiplication is intended component-wise).

A polynomial $f \in k[X_0, \ldots, X_n]$ vanishes at $x \in \mathbb{P}^n$, if $f(x) = 0$. For this definition to be well-defined, $f(\lambda x) = 0$ must hold for every $0 \neq \lambda \in k$. This is possible, only if $f$ is homogeneous, i.e., if every monomial of $f$ has the same degree. A projective variety consists of all points of the projective space, for which a finite number of homogeneous polynomials in $k[X]$ vanish. In Section 2.6.3 we embed a communication protocol in the projective space. The reason we do so (instead of working with the affine space) is due to a nice property of the dimension of projective varieties. A projective variety in $\mathbb{P}^n$ specified by $r$ homogeneous polynomials has dimension at least $n - r$ (see [115]).

Another important contribution of algebraic geometry that we borrow for Section 2.8 is known as Hilbert’s Nullstellensatz (see [125]). In a slightly weaker form it states the following. Let $k$ be an algebraically closed field $k$ and $V(h)$ an affine variety. Suppose that
V(h) \subset V(f) holds for a polynomial f \in k[X]. Then f^i = g \cdot h, for an i \in \mathbb{N} and a g \in k[X].

A Brief Overview of Mechanism Design Theory

After having reviewed the concepts of algebra that we use as tools, we introduce the theory of Mechanism Design that motivates the first chapter. We begin with an example. Imagine to run a business that deals with the design of Peer to Peer software. You design a client to broadcast a live streaming through the Internet in such a way, that a receiver becomes a sender as well. The reason to do this is to exploit the users’ upload bandwidth. The increased total throughput permits to extend the set of potential receivers. Clearly, the ultimate goal of the business is to maximize the revenue (or, equivalently, the number of users). Even though this seems to be a profitable business, a catch becomes apparent if we try to look at the described scenario from the perspective of a user. A user’s interest is to profit from the live streaming, but at the same time to limit the usage of the upload bandwidth, as this is expensive and might slowdown other local processes. Hence, the user selfishly blocks or limits the upload band as a consequence of her personal interests, disregarding the wider social goal of reaching the maximal possible number of users. This example illustrates a general weakness of many multi-agents systems: the global objective of the society does not match the objective of every single entity.

Mechanism Design addresses such scenarios. It is an intriguing branch of Game Theory, and science in general, that has the peculiar feature to be an economic subject with a strong computer science perspective, and, conversely, to be a tool from economics to address computer science problems. We refer the reader to the book by Nisan, Roughgarden, Tardos and Vazirani [95] for a sound introduction to Game Theory in general, and Mechanism Design in particular.

Mechanisms are algorithms designed to optimize a social choice function that depends on the information held by different entities (agents, players) of the system (society) that act driven by personal interests. The social choice function has the purpose to aggregate each piece of information to perform a decision that affects the society. The issue is that each player’s piece of information – the type of a player – is a private information known only to her. A player’s
objective is not necessarily aligned with the goal of the society and is modeled by a personal utility function. Therefore, when queried by the mechanism, a player might report a wrong type in the attempt to influence the mechanism’s decision to a more favorable outcome (that has a higher utility). The purpose of the mechanism designer is to specify the mechanism to comply with this strategic setting, assuming that the players act according to the rational behaviour studied by Game Theory. A well-established approach is to foresee a payment to (or from) the players in such a way, that each player’s personal utility is maximized by reporting the true type. For instance, in the introductory example the player’s utility might be the payment granted from the firm to exploit the upload band minus the cost occurred by allocating it. This reduces the players’ willingness to limit the upload band.

The seminal paper by Nisan and Ronen [94] initiates the analysis of the computation tractability of mechanisms. Which problems can be both solved in the strategic scenario (economic question) while still being computationally accessible (engineering question)? This field is now known as Algorithmic Mechanism Design and received a lot of attention from the algorithmic community during the last years.

Needless to mention, the Internet, with its astonishing growth and increasing impact on the everyday life, is demanding for solutions to novel and pressing problems. More often than not, the problems rise principally because of two aspects of the Internet. On one side, the Internet is a distributed system, thus it necessitates scalable algorithms and protocols. On the other side it is composed by countless entities, each driven by personal interests and acting in a strategic way. This important source of motivation has pushed the specialization of mechanism design even further. The distributed nature of the Internet poses another question about the feasibility of mechanisms: can they be implemented in a distributed fashion? Feigenbaum, Papadimitriou, and Shenker [41] were the first to consider distributed aspects of mechanism design by investigating mechanisms for a problem called the Multicast Cost Sharing Problem in a distributed setting, establishing the Distributed Algorithmic Mechanism Design. In this chapter we study the Multicast Cost Sharing Problem under this point of view.

We informally review the classical concepts of Mechanism Design without considering the full generality. We denote by $\mathcal{O}$ the set of outcomes, i.e., an element $o \in \mathcal{O}$ represents the decision of the mecha-
anism and affects the society. Further, we have \( n \) players \( 1, \ldots, n \). Each player \( i \) has a type \( t_i \), known only to her, that fully characterize the player. Given an outcome \( o \), the valuation function \( v_i = v_i(o, t_i) \) describes the value that a player \( i \) assigns to the outcome \( o \). The quantity \( p_i \in \mathbb{R} \) represents the payment to (or from) player \( i \), determined by the mechanism. A broadly accepted model assumes the utility functions \( u_i, i = 1, \ldots, n \), to be quasi-linear, that is \( u_i = v_i - p_i \). A player assesses her personal happiness of an outcome by the value of the utility function. Hence, the selfish (or rational) player’s goal is to maximize the own utility function.

A mechanism \( M \) is specified by a pair of functions \( (\sigma, p) \). After querying the players and collecting the reported types \( a_1, \ldots, a_n \), the mechanism selects an outcome \( \sigma(a_1, \ldots, a_n) \in \mathcal{O} \) and charges (or pays) player \( i \) a quantity \( p_i = p_i(a_1, \ldots, a_n), i = 1, \ldots, n \). A mechanism is said to be strategyproof or truthful if the best strategy, i.e., the strategy that maximizes the utility function for a player is to report the true type, regardless of what the other players do. This concept can be generalized to prevent a group of players to form a coalition: a mechanism is said to be group-strategyproof if in every group of players that agree on a strategy to exploit the mechanism by lying, there is at least a player in the group that suffers, i.e., whose utility is strictly less than by reporting the truth.

### 2.1.2 The Multicast Cost Sharing Problem

Nowadays, the distribution of multimedia content through the Internet is very popular and is an interesting source of revenue for the movie and music industry. The increasing quality of the content and number of consumers demand for efficient ways to capitalize the network infrastructure. For the scientific community this necessity turns into challenging questions.

The Multicast Cost Sharing Problem is an effort to model the multicast routing protocol [31] under an economical point of view. Traditionally, the form of routing in the Internet is unicast: the transmission of a packet from a sender to multiple receivers is realized by sending a copy of the packet from the sender to each receiver. As a consequence, the traffic near to the sender is congested by multiple copies of the same packet. The multicast routing scheme addresses this problem. First, it builds a suitable rooted tree that connects the
sender to the receivers. The packet is then sent along this tree: it is duplicated only at the branches of the tree (see Figure 2.1).

![Unicast vs Multicast Routing](image)

**Figure 2.1:** The unicast routing protocol (left) is compared to the multicast routing protocol (right).

Suppose that we want to offer a movie in live streaming using a multicast routing scheme. Even though we employ a clever routing scheme, the link cost of sending a large streaming to many users remains non-negligible and has to be split among the receivers. Obviously, a user that receives the transmission cannot be fairly charged of the whole cost of the link to the sender, because some links are shared with other users. Moreover, a user is willing to receive the transmission only if the cost she is charged for is not higher than her interest in receiving the streaming. In turn, this might change the receiver set and require a new way of splitting the costs. A cost sharing mechanism asks the users about their interest of receiving the transmission, selects the receiver set, and determines the price every user has to pay.

Formally, in the Multicast Cost Sharing Problem, a tree, called the multicast tree, and a provider – the root of the tree – are given. The provider wants to send a transmission to \( n \) players, residing at the nodes of the tree. Each player has a valuation \( v_i \) for receiving the transmission and this value is known only to her. The utility for user \( i \) is defined as \( u_i = v_i - p_i \), if she receives the transmission where \( p_i \) is the cost charged, and 0 otherwise. The provider incurs a known cost \( c_e \) when she uses a particular edge \( e \) of the tree for the transmission, and the cost of a link from the sender to the receiver is the sum of the costs of the used edges. The nodes are capable of doing multicast, so edges that are shared by the users in the tree are payed only once. We assume that there is no centralized authority that knows all the
values. Instead the computation is done at the nodes of the trees and values have to be sent over the links. A mechanism for this problem is said to be **budget-balanced** if the revenue raised from the players cover the transmission cost exactly, while it is called **efficient** if it maximizes the net worth. The net worth measures the total benefit, i.e., the sum of the valuations of the selected users minus the total cost of the transmission. A classical result states that no strategyproof mechanism can be both budget balanced and efficient [53].

There have already been attempts to settle the communication complexity of the Multicast Cost Sharing Problem, i.e., to assess how many messages are sent by every mechanism that solves the problem in the multicast tree. Feigenbaum, Papadimitriou, and Shenker [41] investigate the communication complexity of computing budget-balanced mechanisms with linear operations, that is, only multiplication with scalars but not within variables are allowed. This implies that all the sent messages are linear combinations of the values held by the players. In this setting, they show that the computation of any budget-balanced mechanism requires a linear number of messages over a linear number of links, yielding an overall quadratic lower bound. Any function can be computed by this many messages by sending all inputs to one player, so this lower bound is optimal. They leave the extension of their results to non-linear operations, i.e., multiplications and divisions, as an open question. Further they show that a class of important mechanisms, called marginal cost mechanisms, have efficient distributed implementations for the Multicast Cost Sharing Problem with only a constant number of messages per link [41].

Later, Feigenbaum et al. [40] used Boolean communication complexity to prove lower bounds for the bit complexity of distributed algorithms for budget-balanced mechanisms. In the Boolean model, all numbers that occur are rational numbers (given as the quotient of two natural numbers in binary representation). The messages can be arbitrary bit strings. In this model, Feigenbaum et al. prove that any distributed Boolean algorithm has to send at least a linear number of *bits* over a linear number of links. While this seems to solve the problem at a first glance, there is a flaw hidden: if we measure the bit complexity of the messages, we have to relate it to the *bit size* of the input. In particular, in order to prove the mentioned lower bound, Feigenbaum et al. consider the following simple scenario: there is one link from the provider to one node $v$ at which $n/2$ players reside
and there is one link from the provider to another node $u$ at which other $n/2$ players reside. By instantiating the utilities of the players appropriately, it turns out that any distributed algorithm that computes a budget-balanced mechanism on this tree decides whether the sets of utilities at the nodes $u$ and $v$ are disjoint. If we now consider this to be a game between two players, one at $u$, the other at $v$, this becomes the well-known Set-Disjointness Problem. It is known that deciding the Set-Disjointness Problem in the Boolean setting where both players hold the characteristic vector of their subset of $\{1, \ldots, n\}$ requires $n$ bits of communication – see [69]. As argued before, the Multicast Cost Sharing Problem problem is a generalization, so we get the lower bound of $n$ bits as well. In [40], the utilities of the players are of the form $c_1 - c_2 \cdot i$ for some constants $c_1, c_2$ and $1 \leq i \leq n$. Thus the overall input size is $\ell = \Theta(n \log n)$, but the lower bound for the communication is only $\Omega(n^2) = \Omega(\ell^2 / \log^2 \ell)$ \footnote{To see this note that $\log \ell = \Theta(\log n)$.} In fact, if we replace the one link between $u$ and $v$ by a path of length $n$, we get $n$ bits over $n$ links, yielding the quadratic lower bound. While this is still bad enough, it is not matching the $O(\ell \cdot n) = O(\ell^2 / \log \ell)$ complexity of the naive algorithm that sends the whole information to the central node.

We will show that in the general algebraic model, we can get a quadratic lower bound. We provide tools to investigate the communication complexity of distributed computations in a general algebraic setting. This means, that the player may perform arbitrary arithmetic operations and not only linear ones, together with comparisons. The messages are now arbitrary rational functions in the inputs of the players. We will show that in this setting, the computation of any budget-balanced mechanism needs a linear number of messages over a linear number of links. This is now a tight lower bound.

## 2.2 Related Work

The Boolean model was proposed by Yao [123] and was typically motivated by VLSI design problems. However it quickly found other applications, see e.g. [56, 69]. It is mostly of combinatorial nature, whereas our model is based on algebraic structures in which the possible operations are the natural arithmetic operations. Compared to
the Boolean setting, we work over infinite fields and treat the arising numbers as entities. So our results do not follow from discrete models with larger (but finite) alphabets.

Abelson [1, 2] motivated and introduced continuous communication complexity over $\mathbb{R}$ assuming some differentiability properties of the involved functions. Luo and Tsitsiklis [82] improved Abelson’s results in certain cases making use of algebraic tools, but they only consider computational problems and not decision problems. The reason for this is that they only consider smooth messages and smooth problems.

Other than these works, there are also studies leading to more specific directions, like optimization within an error of a sum of two distributed convex functions, where every player has access to a single function [119].

Recently and independently, Grigoriev [54] introduced a model similar to ours, but he only deals with the one-way communication complexity. His model is actually an extension of the one-way model presented here: both parties send messages to some referee who then makes the decisions. We will use one of his results, which can be readily generalized to the two-way model to deal with comparisons. Grigoriev allows randomization, too, but he mainly uses rank-based lower bounds.

Feigenbaum, Papadimitriou, and Shenker [41] investigated the Multicast Cost Sharing Problem in a restricted algebraic setting. They only allow messages that are linear forms in the inputs of the players. They leave the proof of lower bounds in a general model as an open problem. In this thesis, we prove a quadratic lower bound for the communication complexity of budget-balanced mechanisms for the multicast cost sharing problem in this a general model.

2.3 Summary of Results

In this chapter, we introduce and develop the field of algebraic communication complexity and we provide the connection with the theory of distributed algorithmic mechanism design. We define a general algebraic model, where the involved functions can be computed with the natural operations additions, multiplications and divisions and possibly with comparisons. We provide various lower bound tech-
In particular, we show the following results:

- We completely solve the one-way case for the computation problem and give a characterization in terms of the transcendence degree.
- We solve the one-way case for the decision problem up to one message.
- We provide several lower bounds techniques for the two-way case.
- We extend our methods to decision problems with equality tests and with comparisons.
- We introduce nondeterminism and provide an interesting link to the decision and the computation problem in the one-way case.
- We use a reduction by Abelson to prove bounds in the multi-party case.
- We apply our methods to distributed mechanism design problems, like the multicast cost sharing problem and auctions with single minded bidders and we show that some important classes of these problems cannot be efficiently implemented in a network.

The results of this chapter have been established in a joint work with Markus Bläser [14].

2.4 Definitions and Model

Throughout this chapter, $k$ is a field of characteristic zero which we often assume to be algebraically closed. Our mechanism design problems will be reduced to a two-player algebraic communication problem; so we define and analyze this setting first. We give the players names: Alice ($A$) and Bob ($B$). Alice usually holds an input denoted by $X = (x_1, \ldots, x_n)$ and Bob holds $Y = (y_1, \ldots, y_m)$ and their aim is to evaluate a rational function $f : k^n \times k^m \rightarrow k$. Each player
may send messages that are rational functions in his/her inputs and the messages (s)he has received from the other player so far. Eventually, one of them has to be able to compute \( f(X, Y) \) (computation problem), to decide if the value is zero or not (decision problem), or to prove that indeed the input lies in the zero-set of \( f \) (nondeterminism).

We denote, as hinted in the previous section, by \( k[X, Y] \) the ring of polynomials with variables \((x_1, \ldots, x_n, y_1, \ldots, y_m)\) and coefficients in \( k \) and by \( k(X, Y) \) the field of rational functions over the same set of variables. Further, we denote by \( M_{A \rightarrow B} \) and \( M_{A \leftarrow B} \) the index-set of messages sent by Alice to Bob and vice versa, respectively. The network where messages are sent is completely reliable: there is no data loss and transmissions are error-free. In particular we assume that we can send field-elements as they are, meaning that every message counts as one, no matters how large the number is (for example as real number) and how we encode it. What is really important in this framework is the total number of messages sent during a protocol, hence we neglect the amount of computation performed by the players. In fact, our model relies on Luo and Tsitsiklis’ model [82], but the assumptions on the involved functions are different: we deal solely with polynomial/rational functions. Later, we introduce equality tests and comparisons. In this extended model, we can naturally deal with non-continuous functions and investigate decision problems, too.

A protocol \( P \) for computing a rational function \( f \in k(X, Y) \) is a list of instructions that determine the form and the order of the messages that the players send. Finally, it prescribes how the function value is computed with this information.

**Definition 2.1 (Protocol).** A two-way protocol \( P \) for computing \( f \) consists of:

1. **Disjoint inputs** \( X, Y \) distributed between player \( A \) and player \( B \), respectively.

2. **A collection of messages** \( m_1, \ldots, m_r \) belonging to some field extensions of \( k \), sent in this order, with the following property: for each \( 1 \leq i \leq r \), we have:
   - if \( i \in M_{A \rightarrow B} \), then \( m_i \in k(X, m_1, \ldots, m_{i-1}) \)
   - if \( i \in M_{A \leftarrow B} \), then \( m_i \in k(Y, m_1, \ldots, m_{i-1}) \)
3. We have: \( f \in k(X, m_1, \ldots, m_r) \) or \( f \in k(Y, m_1, \ldots, m_r) \).

\( P \) is called one-way if in addition we force \( M_{A-B} = \emptyset \) and \( f \in k(Y, m_1, \ldots, m_r) \).

The two-way communication complexity of \( f \) is defined as

\[
C(f) := \min_P \, r(P)
\]

where the minimum is taken over the set of all protocols \( P \) for \( f \) and \( r(P) \) is the number of messages sent in \( P \). Similarly we define the one-way communication complexity \( C^\to(f) \). It is obvious that \( C(f) \leq C^\to(f) \leq \min\{n, m\} \), because the player that owns the least number of variables can transmit them, one for every message, to the other player.

A message \( m \) is said to be feasible in step \( i \), if the second property in the definition of a protocol holds. If a message \( m \) is feasible, this exactly means that \( m \) can be computed by additions, multiplications, and divisions from the inputs of the particular player and all the messages she received so far.

We will also speak about division-free protocols. In this case, \( m \) is feasible if \( m \in k[X, m_1, \ldots, m_{i-1}] \) or \( m \in k[Y, m_1, \ldots, m_{i-1}] \), respectively. With such protocols, we can only compute polynomials.

Finally, we can also extend the notion of a protocol and communication complexity to a set of functions \( f_1, \ldots, f_\ell \). We require \( f_1, \ldots, f_\ell \in k(X, m_1, \ldots, m_r) \) or \( f_1, \ldots, f_\ell \in k(Y, m_1, \ldots, m_r) \) in the third item of the definition of a protocol.

**Example 2.2.** The gap between \( C(f) \) and \( C^\to(f) \) might be arbitrarily large. To see this, set \( k = \mathbb{R} \) and let \( \alpha = \prod_i x_i \), \( \beta = \prod_j y_j \) and

\[
f_n(X, Y) = \alpha y_1^n + \alpha^2 y_2^{n-1} + \ldots + \alpha^n y_n + \beta x_1^n + \ldots + \beta^n x_n.
\]

Then \( C(f_n) \leq 3 \). Indeed, it is evident that if Alice sends the value of \( \alpha \) to Bob, then Bob can send in two messages the value of \( \beta \) and of \( \alpha y_1^n + \alpha^2 y_2^{n-1} + \ldots + \alpha^n y_n \), which, in turn, enable Alice to compute the value of the whole function. On the other hand, one can easily show with the results of this chapter that \( C^\to(f_n) = n \) holds.
Chapter 2. Algebraic Communication Complexity

2.5 One-way Communication

In this section we deal exclusively with the one-way communication model. One-way communication corresponds to distributed one-pass algorithms. For the example of multicast pricing, this means that the result has to be computed in one sweep over the multicast tree. We provide a technique to compute exactly the value \( C_{\rightarrow}(f) \) for every rational function \( f \in k(X,Y) \).

Consider a function \( f \in k(X,Y) \) and take two coprime polynomials \( p,q \in (k(X))[Y] \) with coefficients in \( k(X) \) such that \( f = p/q \) and \( q \) has at least a non-zero coefficient in \( k \) (as a function of \( Y \)). We see \( f \) as a function of \( Y \), that is

\[
\tilde{f}_X(Y) = f(X,Y) = \frac{\sum_{\omega \in \mathbb{N}^m} p_{\omega} Y^\omega}{\sum_{\lambda \in \mathbb{N}^m} q_{\lambda} Y^\omega}
\]

where, as mentioned above, the non-zero rational functions \( p_{\omega}, q_{\lambda} \in k(X), \omega, \lambda \in \mathbb{N}^m \), depend only on \( X \), and \( q_{\lambda} \in k \) for at least one \( \lambda \). For such an \( f \in k(X,Y) \), we denote by \( \text{Coeff}_Y f \) the field extension over \( k \) generated by adding the coefficients \( p_{\omega}, q_{\lambda} \) to \( k \), \( \omega, \lambda \in \mathbb{N}^m \). We call \( \text{Coeff}_Y f \) the coefficient field of \( f \) with respect to \( Y \). Notice that \( \text{Coeff}_Y f \subset k(X) \). This definition extends naturally to \( \text{Coeff}_Y(f_1,\ldots,f_r), f_1,\ldots,f_r \in k(X,Y) \). The reason for which we require \( q \) to have a non-zero \( k \)-coefficient is justified by the next lemma. The lemma states that the concept is well-defined and that in this way we get the “smallest” possible coefficient field that describes the function \( f \).

**Lemma 2.3 ([105]).** Let \( f \in k(X,Y) \). Then \( \text{Coeff}_Y f \) is well-defined. Further let \( g,h \in k(X)[Y], q \neq 0 \) such that \( f = g/h \). Then

\[
\text{Coeff}_Y f \subset \text{Coeff}_Y(g,h).
\]

Note that by that by the definition of a protocol and Lemma 2.3 Bob is able to compute the value of \( f \) from the messages \( m_1,\ldots,m_r \) received from Alice if and only if \( \text{Coeff}_Y f \subset k(m_1,\ldots,m_r) \).

**Theorem 2.4 (Transcendence degree bound).** For every field \( k \) and rational function \( f \in k(X,Y) \), we have

\[
C_{\rightarrow}(f) \geq \text{tr deg}_k \text{Coeff}_Y f.
\]
2.5. One-way Communication

Proof. We proceed by induction on $q := \text{tr} \deg_k \text{Coeff}_Y f$. For $q = 0$, we have $f \in k(Y)$. For $q > 0$, let $m_i$ be the first message such that $\text{tr} \deg_k (m_1, \ldots, m_i) k(m_1, \ldots, m_i) = 1$. Then we have that $\text{tr} \deg_k (m_1, \ldots, m_i) \text{Coeff}_Y (f) = \text{tr} \deg_k \text{Coeff}_Y (f) - 1$ and the induction hypothesis applies. Hence Alice sends at least $q - 1 + i \geq q$ messages. 

Example 2.5. Theorem 2.4 settles the claim that $C^{-\epsilon}(f_n) = n$ for the function $f_n$ of Example 2.2. In fact, $\text{Coeff}_Y (f_n)$ contains the terms $x_1^n, x_2^{n-1}, \ldots, x_n$ which are algebraically independent over $\mathbb{R}$. Thus, $\text{tr} \deg_\mathbb{R} \text{Coeff}_Y f_n = n$ and the claim follows.

It is easy to prove with the Primitive Element Theorem (see [17]), that $\text{tr} \deg_k \text{Coeff}_Y f + 1$ is an upper bound for $C^{-\epsilon}(f)$. Alice sends a complete transcendence basis of $\text{Coeff}_Y f$ over $k$ to Bob using $\text{tr} \deg_k \text{Coeff}_Y f$ messages. Fields of characteristic zero are separable, so the Primitive Element Theorem for algebraic extensions assures that at most one more message makes the extended field equal to $\text{Coeff}_Y f$ establishing the claim. However this result can be strengthened and we get tight bounds for $C^{-\epsilon}(f)$ in the one-way case.

Lemma 2.6. Let $f_1, \ldots, f_r \in k(X)$ and $q = \text{tr} \deg_k k(f_1, \ldots, f_r)$, then there exist $g_1, \ldots, g_q \in k(f_1, \ldots, f_r)$ with the property that $k(f_1, \ldots, f_r) \subseteq k(g_1, \ldots, g_q)$.

Plausibly, the result might not be original, however we add our proof for the reader’s convenience.

Proof. We proceed by induction on $d := \text{tr} \deg_k k(f_1, \ldots, f_r)$ and on $r$. For $d = r = 1$ the claim is trivial. Assume now $d = 1$ and $r = 2$. Because of the algebraic dependency of $f_1, f_2$ we can find a non-zero irreducible polynomial $f \in k(X,Y)$ such that $f(f_1, f_2) \equiv 0$. Consequently $k(f_1, f_2)$ is isomorphic to $k(X,Y)/(f)$, since the ideal $(f)$ is the kernel of the map sending $f_1$ to $X$ and $f_2$ to $Y$. It turns out that $V = V(f)$ is a rational curve: choose the rational functions $\tilde{\varphi}, \tilde{\psi}$ such that at least one of $f_1(\tilde{\varphi}(t), \tilde{\psi}(t))$ or $f_2(\tilde{\varphi}(t), \tilde{\psi}(t))$ is not a constant, then we obviously have $f(\varphi(t), \psi(t)) \equiv 0$ for $\varphi(t) = f_1(\tilde{\varphi}(t), \tilde{\psi}(t))$, $\psi(t) = f_2(\tilde{\varphi}(t), \tilde{\psi}(t))$.

This induces the following chain of natural isomorphisms:

$$k(f_1, f_2) \simeq k(X,Y)/(f) \simeq k(\varphi(t), \psi(t)) = k(\omega(t)) \simeq k(t).$$
$k(\varphi(t), \psi(t)) = k(\omega(t))$ (for a suitable rational function $\omega(t)$), follows from Lüroth’s Theorem, which states that every subfield of $k(t)$ is of the form $k(\omega(t))$ for some rational function $\omega$. Altogether we get an isomorphism $\sigma : k(f_1, f_2) \sim k(t)$.

Consider $\hat{f}_i := \sigma(f_i)$, then it follows that $\hat{f}_i = \chi_i(t)$ for some $\chi_i \in k(X)$, $i = 1, 2$. Since $\sigma$ is an isomorphism, the polynomial relation is preserved, and so

$$f_i = \sigma^{-1}(\hat{f}_i) = \sigma^{-1}(\chi_i(t)) = \chi_i(\sigma^{-1}(t)) = \chi_i(s), \quad i = 1, 2$$

for $s = \sigma^{-1}(t) \in k(f_1, f_2)$. It clearly implies that $k(f_1, f_2) \subset k(s)$.

For $d = 1$ and an arbitrary $r > 2$, we get the claim inductively as follows: by induction hypothesis we have $k(f_1, \ldots, f_{r-1}) \subset k(s)$ for an $s$, therefore $k(f_1, \ldots, f_{r-1}, f_r) \subset k(s, f_r) \subset k(s^*)$ for a suitable $s^*$, since the transcendence degree of $k(s, f_r)$ over $k$ is 1. Indeed, in order to get $k(f_1, \ldots, f_{r-1}) \subset k(s)$, one applies the previous case $r - 2$ times, showing that $f_i = \chi_i(s)$, $1 \leq i \leq r - 1$.

To complete the proof, we show the step from $d - 1$ to $d$ with fixed $r$: consider a subset $F$ of $\{f_1, \ldots, f_r\}$ such that $k(F)$ has transcendence degree $d - 1$ over $k$. By the induction hypothesis, we can choose a set $F^*$ of $d - 1$ elements such that $k(F) \subset k(F^*)$. Furthermore, by the properties of the transcendence basis, we have that $\text{tr deg}_{k(F)} k(F, F^c) = 1$, where $F^c = \{f_1, \ldots, f_r\} \setminus F$, allowing us to apply the induction hypothesis on the extension $k(F) \subset k(F, F^c)$. We therefore get an element $s^* \in k(F, F^c)$ such that:

$$k(f_1, \ldots, f_r) \subset k(F, s^*) \subset k(F^*, s^*)$$

with $|F^* \cup \{s^*\}| = d$. □

This lemma assures that Alice can send a transcendence basis $\{m_1, \ldots, m_q\}$ with the property that $\text{Coeff}_Y f \subset k(m_1, \ldots, m_q)$, $q = \text{tr deg}_k \text{Coeff}_Y f$, so the Primitive Element Theorem becomes superfluous. This establishes $\text{tr deg}_k \text{Coeff}_Y f$ as the correct number of messages for every optimal protocol in the one-way communication model. It can be easily computed as the rank of the matrix, whose columns are given by the gradient of the coefficients of $f$, see [82].

**Corollary 2.7.** For every field $k$ of characteristic zero and rational function $f \in k(X, Y)$ it holds that:

$$C^\rightarrow(f) = \text{tr deg}_k \text{Coeff}_Y f.$$
2.6 Two-way Communication

Next we study the two-way model. Here, we do not have a tight characterization as in the one-way case, but we provide several lower bound techniques that show tight bounds for some specific functions, in particular for the ones arising from our mechanism design problems.

2.6.1 Rank Bound

Let \( f \in k[X, Y] \) be given. The goal of this section is to determine a lower bound for the two-way algebraic communication complexity of \( f \) if the messages are only polynomials. Let us denote by \( f_1, \ldots, f_s \) the monomials of \( f \), i.e.

\[
f_k = a_k \cdot X^\alpha_k \cdot Y^\beta_k, \quad 1 \leq k \leq s,
\]

for \( a_k \in k, \alpha_k \in \mathbb{N}^n, \beta_k \in \mathbb{N}^m \). We say that \( x_i \) or \( y_j \) are contained in \( f_k \), if \( (\alpha_k)_i > 0 \) or \( (\beta_k)_j > 0 \), respectively. Further, we define \( S_{x_i} := \{k \mid x_i \text{ is contained in } f_k\} \). Accordingly, we define \( S_{y_j} \).

The coefficient matrix \( M_f \) of \( f \) is defined as follows:

\[
(M_f)_{i,j} = \sum_{k \in S_{x_i} \cap S_{y_j}} a_k
\]

In other words, \((M_f)_{i,j}\) is the sum of the coefficient of the monomials that contain both the variables \( x_i \) and \( y_j \). Note that we disregard the exponent of every variable.

For instance, for \( f(X, Y) = x_1 y_1 + 2x_2 y_2 \), we get

\[
M_f = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}.
\]

Let a protocol \( P \) with messages \( m_1, \ldots, m_r \) be given. Let the polynomial \( h \) be the function that allows a player to compute the value \( f \) given the messages \( m_1, \ldots, m_r \). Hence, one of the following equations are fulfilled for all \( X, Y \):

\[
\begin{cases}
  f(X, Y) = h(X, m_1, \ldots, m_r), & \text{if } A \text{ computes the value of } f \\
  f(X, Y) = h(Y, m_1, \ldots, m_r), & \text{if } B \text{ computes the value of } f
\end{cases}
\]
In the same spirit of the matrix $M_f$, we can define the coefficient matrix $M_{m_\rho}$ of a message $m_\rho$ of $P$. $M_{m_\rho}$ is constructed with the information provided by $h$. Its role is to encode the additional information that a message carries to the player who gets it. For the sake of the definition we assume that Alice eventually computes the value of $f$. For a message $m_\rho$, $1 \leq \rho \leq r$, let $M_{m_\rho}$ be the matrix obtained as follows: consider $h$ as a function of $m_\rho$:

$$\tilde{h}(m_\rho) := h(X, m_1, \ldots, m_n) = \sum_{u \geq 0} h_{u,m_\rho} m_\rho^u$$

whereby $h_{u,m_\rho}$, $u \geq 0$, describe the dependency of the other arguments of $h$.

In turn, we assume that $m_\rho$ was sent by Alice (the other case is symmetric) and we write

$$\tilde{m}_\rho(X) := m_\rho(X, m_1, \ldots, m_{\rho-1}) = \sum_{\sigma \in \mathbb{N}^n} g_{\sigma,m_\rho}^m X^\sigma.$$

where, again, $g_{\sigma,m_\rho}^m$, $\sigma \in \mathbb{N}^n$, describe the dependency of the arguments $m_1, \ldots, m_{\rho-1}$.

Fix two variables $x_i, y_j$, $1 \leq i \leq n, 1 \leq j \leq m$. $x_i$ is contained in some monomials $g_{\sigma,m_\rho}^m$ and $y_j$ is contained in some monomials $h_{u,m_\rho}$. First, set $m_k \equiv 1$, $1 \leq k \leq \rho - 1$, in all the monomials $h_{u,m_\rho}$. Take the sum of the products of the coefficients (the elements of $k$) of the monomials $g_{\sigma,m_\rho}^m$ that contain $x_i$ with the monomials $h_{u,m_\rho}$ that contain $y_j$. The coefficient matrix of the message $m_\rho$ (with respect to the protocol $P$) $(M_{m_\rho})_{ij}$ is then precisely this quantity. Note that we set the messages sent before $m_\rho$ to one to avoid counting products more than once. Roughly speaking, the matrix $M_{m_\rho}$ encodes the set of new products that are computable after receiving the message $m_\rho$. The other case case – if the message $m_\rho$ has been sent by Bob – is symmetric: we just interchange the role of $X$ and $Y$.

**Example 2.8.** We illustrate the definition of $M_f$ and $M_{m_\rho}$ by an example. Let $f(X, Y) = x_1 y_1 + 2x_2 y_2$. Consider the following (non-optimal) protocol: messages are $m_1(X) = x_1$, $m_2(Y, m_1) = m_1 y_1$, $m_3(Y) = y_2$ and the evaluating function $h$ is:

$$f(X, Y) = h(X, m_1, m_2, m_3) = m_2(m_1) + 2 \cdot m_3 \cdot x_2$$
so we get:

\[ M_{m_1} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad M_{m_2} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad M_{m_3} = \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix} \]

Note that \( M_{m_2} \) does not count the first product \( x_1y_1 \) again.

With these definitions of the coefficient matrices, the following claims hold.

**Lemma 2.9.** If \( m_1, \ldots, m_r \) are the messages of a protocol \( P \) for computing \( f \), then \( M_{m_1} + \ldots + M_{m_r} = M_f \).

**Proof.** We show the contraposition. We assume that \( M_{m_1} + \ldots + M_{m_r} \neq M_f \). Thus we can find a component \((i, j)\), such that the left-hand side is either bigger or smaller than the right-hand side. The first case can not occur, as we argued above each product cannot be counted more than once. The second case would mean that we have a combination given by \( x_i \) and \( y_j \) in \( f \), which has to be (completely) contained in some message or in the coefficients of \( h \). Since a player does not know both the \( X \)-variables and the \( Y \)-variables without getting the information from a message, we conclude that the protocol \( P \) cannot evaluate \( f \). \qed

**Lemma 2.10.** For a protocol \( P \) that computes \( f \), we have \( \text{rk} \ M_m \leq 1 \) for every message \( m \) in \( P \).

**Proof.** We assume that for a message \( m \) we have \( \text{rk} \ M_m \geq 2 \). Then we can find a \( 2 \times 2 \)-submatrix of \( M_m \) of rank two. Without loss of generality we call the involved variables \( x_1, x_2, y_1, y_2 \).

First note that \( m \) has to contain at least one variable \( x_i \) and one variable \( y_j \), for \( i, j \in \{1, 2\} \), because if \( m \) would not contain, say, any \( Y \)-variable, then the \( X \)-variables in \( m \) multiply with the same monomials in \( h \) (see the definition of \( M_m \)) that contain \( y_1, y_2 \), leading to a submatrix of rank 1.

Assume that \( m \) is sent by Alice, then, because of the feasibility of \( m \), Alice has received the information about the \( Y \)-variables of \( m \) from some previous messages. Since these messages are set to one before identifying the products, the \( Y \)-variables would vanish and we get the zero matrix, a contradiction. \qed
Chapter 2. Algebraic Communication Complexity

The properties stated above allow us to use the rank function as an invariant to determine the communication complexity (with polynomial messages) of a function $f$.

**Theorem 2.11 (First rank bound).** For every polynomial $f$, every division-free protocol requires at least $\text{rk } M_f$ messages.

**Proof.** Let $m_1, \ldots, m_r$ be the collection of messages of a protocol $P$, which computes $f$. From $M_f = M_{m_1} + \ldots + M_{m_r}$ and $\text{rk } m_i \leq 1$ for all $i$, we get by using the subadditivity of the rank function:

$$\text{rk } M_f \leq \text{rk } M_{m_1} + \ldots + \text{rk } M_{m_r} \leq r.$$  

So, every protocol needs at least $\text{rk } M_f$ messages. \hfill \qed

**Example 2.12.** Using the rank bound, it is easy to show that the scalar product $f = x_1 y_1 + \cdots + x_n y_n$ has communication complexity equal to $n$, since $M_f$ is the identity matrix and has full rank.

Abelson [2] proved a similar bound that is also valid in our model. Let $H_f$ be defined by

$$(H_f)_{i,j} = \left. \frac{\partial^2 f}{\partial x_i \partial y_j} \right|_{X=p, Y=q}, \quad 1 \leq i \leq n, 1 \leq j \leq m$$

where $p$ and $q$ can be chosen from some open subset $D$ where $f$ is defined.

**Theorem 2.13 (Second rank bound).** For every function $f$, $C(f) \geq \text{rk } H_f$.

### 2.6.2 Substitution Method

The idea of the substitution method is to make the first message trivial by adjoining it to the ground field. Since we deal with several ground fields in this section, we write occasionally $C(f; k)$ instead of $C(f)$ to stress the underlying ground field.

**Lemma 2.14.** Let $P$ be some protocol for computing a rational function $f$ over some field $k$ that uses $r > 0$ messages. Then there is an extension field $k' \supset k$ with $\text{tr deg}_k k' \leq 1$ and either $k' \subset k(X)$ or $k' \subset k(Y)$ and there is a protocol $P'$ for computing $f$ with $r - 1$ messages over $k'$. 
2.6. Two-way Communication

Proof. Let $m_1$ be the first message of the protocol. Set $k' = k(m_1)$. Then we do not need to send the first message, since it is now known to both players in advance. The claim on the transcendence degree is obvious and also that either $k' \subset k(X)$ or $k' \subset k(Y)$, since $m_1$ is the first message.

Example 2.15. Assume that Alice and Bob hold $n$ and $m$ input values, respectively. We want to know whether they have at least one value in common. This is called the Set-Disjointness Problem. It is modelled by the following function

$$\text{Disj}_{n,m}(X, Y) = \prod_{i=1}^{n} \prod_{j=1}^{m} (x_i - y_j).$$

It is now possible to show its high communication complexity by induction.

Theorem 2.16. For every rational function $g = p/q$ such that $q$ is coprime with $\text{Disj}_{n,m}$, $C(g \cdot \text{Disj}_{n,m}) \geq \min\{m, n\}$ holds.

Proof. The proof is by induction on $\min\{m, n\}$. Since $q$ is coprime with $\text{Disj}_{n,m}$, we need at least one message. If $\min\{m, n\}$ is 1, then the claim is trivial. Otherwise we want to apply Lemma 2.14. We treat the case $k' \subset k(X)$, the other case is symmetric. Pick an $X$-variable contained in $m_1$ and rename the variables such that $x_n$ is a variable contained in $m_1$, where $m_1$ is the message substituted. Hence $x_1, \ldots, x_{n-1}, m_1$ are algebraically independent. Note that $x_1, \ldots, x_{n-1}$ and $y_1, \ldots, y_m$ are algebraically independent over $k' = k(m_1)$. We can write $\text{Disj}_{n,m} = g' \cdot \text{Disj}_{n-1,m}$ where $g'$ consists of all terms that contain $x_n$.

By the inductions hypothesis $C(gg' \cdot \text{Disj}_{n-1,m}; k') \geq \min\{n-1, m\}$ and by Lemma 2.14, we see $C(g \cdot \text{Disj}_{n,m}; k) \geq C(gg' \cdot \text{Disj}_{n-1,m}; k') + 1$. This proves the claim.

2.6.3 Dimension Bound

In this section, we show lower bounds using methods from algebraic geometry and in particular, we relate results about the dimension of a variety to our problem.
Chapter 2. Algebraic Communication Complexity

We here consider only homogeneous polynomials $f_1, \ldots, f_\ell$ and we want to decide whether the input of Alice and Bob lies in the variety $V(f_1, \ldots, f_\ell)$ – the zero-set of all of them. As explained in the introduction, we restrict ourselves to \emph{projective protocols}, i.e., protocols in which all messages are homogeneous polynomials. Now we can work over the projective space $\mathbb{P}^{2n} := (k^{2n+1} \setminus \{0\})/ \sim$, where $a \sim b$ iff there exists a $0 \neq \lambda \in k$ such that $a = \lambda b$. The inputs of the players are embedded canonically into the projective space by adding a new coordinate to their input and setting it to 1. In other words, the input $(x_1, \ldots, x_n, y_1, \ldots, y_n)$ is mapped to $(x_1, \ldots, x_n, y_1, \ldots, y_n, 1)$. Whenever we speak of the point $(0,0)$, we mean the corresponding point $(0, \ldots, 0, 0, \ldots, 0, 1)$ in $\mathbb{P}^{2n}$. Since we always work in the affine subspace of $\mathbb{P}^{2n}$ of points with the non-zero last coordinate, we will usually omit the last 1.

**Theorem 2.17.** Let $k$ be an algebraically closed field and let $f : k^n \times k^n \to k^\ell$ be a homogeneous polynomial. Further assume that $\dim V(f(0, Y)) = q < n$. Then deciding whether $(X, Y) \in V(f) = \{(X, Y) \in k^n \times k^n \mid f(X, Y) = 0\}$ requires at least $n - q$ homogeneous messages in the two-way model.

**Proof.** Let $P$ be a homogeneous protocol and assume without loss of generality that Alice eventually decides whether $f(X, Y) = 0$ for a $(X, Y) \in k^{2n}$. We embed $(X, Y)$ into $\mathbb{P}^{2n}$ by adding a component 1 as described above. For the sake of contradiction, we assume that $n - q - 1$ messages are enough for Alice to decide whether $f(X, Y) = 0$; we denote them by $m_1, \ldots, m_{n-q-1}$. Since the messages are homogeneous, it follows in particular that $m_i(0,0) = 0$, $1 \leq i \leq n - q - 1$. Also $M := V(m_1, \ldots, m_{n-q-1})$ is a non-empty projective variety in $\mathbb{P}^{2n}$ with $\dim M \geq n + q + 1$.

Consider the variety $E := V(x_1, \ldots, x_n)$ of dimension $n$. By the properties of the projective space it follows that $\dim M \cap E \geq q + 1$. On the other hand, $\dim V(f(0, Y)) = q$, so we can find a point $(0, b)$ in the intersection $M \cap E$ with $f(0, b) \neq 0$. If we run $P$ on the inputs $(0,0)$ and $(0, b)$, we notice that Alice has the same input and receives vanishing messages in both instances. Hence she is unable to distinguish between $f(0,0) = 0$ and $f(0,b) \neq 0$; this is a contradiction. \hfill $\Box$

**Example 2.18.** This theorem applies in a straightforward manner to the equality problem. We want to decide whether the two inputs are
identical. The problem is modelled by the $n$ functions $f_1 = x_1 - y_1, \ldots, f_n = x_n - y_n$. Let $f(X, Y) = (f_1(X, Y), \ldots, f_n(X, Y))$. Of course we have $\dim V(f(0, Y)) = 0$, since only $(0, 0)$ belongs to it. Thus any homogeneous protocol for deciding whether the two inputs are the same requires $n$ messages.

This argument is interesting in another important context as well. If the theorem should extend to the equality problem over every field $k$ of characteristic zero not necessarily closed, then this would rule out the existence of an injective polynomial $p \in k[X, Y]$, which is still an open problem for general fields. Indeed, if such a polynomial could be found, applying it iteratively would lead to an injective polynomial $\tilde{p} : k \times \ldots \times k \to k$, which, in turn could be used to find a protocol with one message for the previous problem: both players compute the value of $\tilde{p}$ on their inputs and check whether the images are equal.

On the other hand note that the equality problem can be solved with one message on certain ground sets. For instance, if the inputs are natural numbers, one can use the previous argument with the polynomial $(a, b) \mapsto \frac{1}{2}((a + b)^2 + 3a + b)$, which is a bijection $\mathbb{N}^2 \to \mathbb{N}$.

### 2.7 Decision Problems

Now we allow the players to perform equality tests and comparisons, the latter only over $\mathbb{R}$. In this setting, players do not only compute and send the same messages for every input, but they are allowed to go through a decision tree, where at every node they check whether some two functions of their input are equal. Every internal node in the tree has two successors, one corresponds to the outcome of the fact that the compared functions are equal, one to the outcome that they are not equal. Depending on the result of the test, the players follow the respective branch in the decision tree. When they reach a leaf, they send a message accordingly. Furthermore, we do not want to compute $f$ but it is sufficient to decide whether the input of the players lies in the zero-set $V(f)$ of $f$ or not.

If we allow comparisons, then we can also decide semi-algebraic sets. Every test now has three outcomes, $<$, $=$, or $>$, and every node in the decision tree has three successors.

**Definition 2.19** (Protocol with equality test). A protocol $P$ for de-
ciding if an input \((X,Y)\), distributed over two players \(A\) and \(B\), lies in the zero-set of \(f\), is defined on decision trees: we consider the straight-line program (cp. Definition 2.1), where every node expands to a rooted binary tree \(T\), for which the following properties hold:

1. There is a labelling \(l : \{\text{roots of trees}\} \rightarrow \{A, B\}\), such that for every root \(v\), \(l(v)\) denotes the player who goes through the respective tree and sends a message (or decide the function) once a leaf is reached.

2. At every branching \(\alpha\) of the tree with root \(v\), a test is performed, i.e. \(l(v)\) checks if

\[
g^{(1)}_{\alpha}(X, m_1, \ldots, m_t) \overset{?}{=} g^{(2)}_{\alpha}(X, m_1, \ldots, m_t), \quad \text{if } l(v) = A
\]

\[
g^{(1)}_{\alpha}(Y, m_1, \ldots, m_t) \overset{?}{=} g^{(2)}_{\alpha}(Y, m_1, \ldots, m_t), \quad \text{if } l(v) = B
\]

holds, whereby \(g^{(1)}_{\alpha}\), \(g^{(2)}_{\alpha}\) are rational functions and \(m_1, \ldots, m_t\) are the messages sent previously in the protocol (dependent on the input). If the test is positive, (s)he proceeds with the right child, otherwise with the left child.

3. When a leaf \(\beta\) of the tree with root \(v\) is reached, player \(l(v)\) sends a message \(m_{\beta}(X,Y)\) with the following property:

- \(m_{\beta} \in k(X,m_1,\ldots,m_t)\), if \(l(v) = A\)
- \(m_{\beta} \in k(Y,m_1,\ldots,m_t)\), if \(l(v) = B\)

4. At the last node of the protocol-tree, \(l(v)\) goes through a binary tree as before, but its leaves are now labelled with 0 or 1, with the following meaning:

- If \((X,Y)\) lies in the zero-set of \(f\), then a 0-leaf is reached at the end of the protocol.
- If \((X,Y)\) does not lie in the zero-set of \(f\), then a 1-leaf is reached at the end of the protocol.

In the same way, we can define protocols with comparisons over \(\mathbb{R}\). Here the internal nodes in the tree have three successors, i.e. every test now has three outcomes, \(<, =, \text{ or } >\).

The communication complexities \(C_{\text{dec}}^{-}(f)\) and \(C_{\text{dec}}(f)\) are defined in the same way as \(C^{-}(f)\) and \(C(f)\), but over the larger class.
2.7. Decision Problems

of protocols with equality tests. Therefore, it is clear that \( C_{\text{dec}}(f) \leq C^\rightarrow(f) \) and \( C_{\text{dec}}(f) \leq C(f) \) for all rational functions \( f \). In the same way, we define \( C_{\text{dec},<}(S) \) and \( C_{\text{dec},<}(S) \) for semi-algebraic sets \( S \). Further \( C_{\text{dec},<}(f) \) and \( C_{\text{dec},<}(f) \) shorthand \( C_{\text{dec},<}(V(f)) \) and \( C_{\text{dec},<}(V(f)) \), respectively.

Intuitively, one would expect that these additions would indeed decrease the number of messages needed between the two players. We will see that often this decrease is very modest.

2.7.1 Equality Tests

In this section we study the impact of allowing only equality tests on the communication complexity of a function.

\textbf{Lemma 2.20.} For every irreducible polynomial \( f \in k[X,Y] \) over an algebraically closed field, there is a rational function \( h = p/q \) with \( q \) and \( f \) being coprime, such that \( C_{\text{dec}}(f) \geq C_{\text{dec}}(hf) \) and \( C_{\text{dec}}(f) \geq C(hf) \).

\textit{Proof.} Consider a protocol \( P \), one-way or two-way, deciding the membership of the inputs in \( \Omega := V(f) \), the variety defined by \( f \). Since the possible inputs are infinite (\( k \) is closed), almost every input (in the Zariski sense) follows the same path \( \pi_0 \). Let \( \pi \) be the typical path of an element from \( \Omega \) and let \( \nu \) be the node where \( \pi \) and \( \pi_0 \) separate for the first time. Following the path \( \pi_0 \) up to \( \nu \), we find rational functions \( g^{(1)}_1, g^{(2)}_1, \ldots, g^{(1)}_r, g^{(2)}_r \) such that \( g^{(1)}_i \neq g^{(2)}_i \) is tested by some player. Obviously \( g_i := g^{(1)}_i - g^{(2)}_i \) is not identically zero and because we follow the path taken by most inputs, \( g_1, \ldots, g_{r-1} \) do not vanish on the given input when we follow \( \pi \) or \( \pi_0 \). Since \( \Omega \) is a closed set in the Zariski topology, it follows that the elements of \( \Omega \) reaching \( \nu \) lie also in \( V(g_r) \), and not in its complement. If an element \( (X, Y) \) of \( \Omega \) does not reach \( \nu \) (i.e., it branches from \( \pi_0 \)), it has to fulfill \( g_i(X, Y) = 0 \), for a \( 1 \leq i \leq r-1 \). In other words \( g := g_1 \cdot \ldots \cdot g_r \) vanishes on \( \Omega \). Therefore, applying the Nullstellensatz on the numerator of \( g \) and noting that \( f \) is irreducible, we have \( g = h \cdot f \), for a rational function \( h = p/q \), \( q \) coprime with \( f \). Thus the protocol \( P \) that decides \( V(f) \) implicitly computes \( hf \) in the standard model. \( \square \)

For the one-way case, we get an almost tight characterisation meaning that performing equality tests does not bring any significant
help in the communication task.

**Corollary 2.21.** Given an irreducible polynomial \( f \in k[X, Y] \) over an algebraically closed field \( k \), \( C_{\text{dec}}(f) \geq \text{tr deg}_k \text{Coeff}_Y f - 1 \).

**Proof.** Let \( h \) be the function constructed in the proof of Lemma 2.20. Then we have \( \text{tr deg} \text{Coeff}_Y h \cdot f \geq \text{tr deg} \text{Coeff}_Y f - 1 \), see Lemma 1 in [21].

Note that we cannot apply Lemma 2.20 to \( \text{Disj}_{n,m} \), since \( \text{Disj}_{n,m} \) is not irreducible. To deal with this case, we prove another Lemma.

**Lemma 2.22.** Let \( \ell_1, \ldots, \ell_t \) be linear forms such that any two of them are linearly independent. Let \( L = \ell_1 \cdot \ldots \cdot \ell_t \). Then there is a rational function \( h = p/q \) with \( L \) and \( q \) being coprime, such that \( C_{\text{dec}}(L) \geq C(hL) \).

**Proof.** Consider a protocol \( P \) for deciding \( L \) and let \( \pi_0 \) be the path taken by almost all inputs. Let \( \pi_\tau \) be the path taken by almost all inputs in \( V(\ell_\tau) \). Let \( v_\tau \) be the node where these two paths separate for the first time. Let \( g_\tau = p_\tau/q_\tau \) be the rational function tested at \( v_\tau \). By Gauss’ Lemma \( ^3 \), \( p_\tau \) has a unique factorization and because \( \ell_\tau \) is linear, we have \( \ell_\tau | p_\tau \). Thus \( g_1 \cdots g_t \) can be written as \( hL \).

**Corollary 2.23.** \( C_{\text{dec}}(\text{Disj}_{n,m}) \geq \min\{n, m\} \).

**Proof.** We have \( C_{\text{dec}}(\text{Disj}_{n,m}) \geq C(h \text{Disj}_{n,m}) \geq \min\{n, m\} \) by Lemma 2.22 and Theorem 2.16.

### 2.7.2 Comparisons

At this point, we introduce the capability of performing inequality tests as well. We consider the lexicographic monomial ordering on the variables \( y_n, \ldots, y_1, x_n, \ldots, x_1 \), i.e., a monomial \( m \) is smaller than a monomial \( m' \) if the exponent vector of \( m \) is smaller than that of \( m' \) in the lexicographic ordering (with variables in the order as above). For a polynomial \( p \), \( \text{lt}(p) \) denotes the least term with respect to the chosen monomial ordering. Grigoriev [54] essentially proves the following result.

\(^3\text{Gauss’ Lemma states that if } k \text{ is a field (or, in general, a factorial domain), then every polynomial } p \in k[X] \text{ admits a unique factorization in irreducible polynomials.}\)
Lemma 2.24 ([54]). Let $\ell_1, \ldots, \ell_n$ be linear forms with the property that $x_1, \ldots, x_n, \ell_1, \ldots, \ell_n$ are linearly independent. Let $V$ be the union of some hyperplanes, among them $V(\ell_1), \ldots, V(\ell_n)$. Then there is a polynomial $f$ such that $C_{\text{dec},<}(V) \geq C(f)$ and $\ell_1, \ldots, \ell_n$ divide $\text{lt}(f)$. The same holds for the semialgebraic set $S$ defined by $\ell_1 > 0, \ldots, \ell_t > 0$.

Now the substitution method from Section 2.6.2 allows us to get rid of one message for each linear form $\ell_i$ that divides $\text{lt}(f)$ and get a bound for $C_{\text{dec},<}(\ell_1, \ldots, \ell_n)$ in this way. Note that every indeterminate is substituted by a polynomial (cp. the proof of Theorem 2.16), so the term $\text{lt}(f)$ does not change and we can proceed by induction.

Corollary 2.25. $C_{\text{dec},<}(\text{Disj}_{n,m}) = \min\{n, m\}$.

Proof. $\text{Disj}_{n,m}$ is a product of linear forms, among them we have $x_1 - y_1, \ldots, x_n - y_n$. From the lemma above, we get a function $f$ so that these linear forms divide $\text{lt}(f)$. Now we can apply the substitution method to $f$ in the same manner as we did before to $\text{Disj}_{n,m}$. 

In the same way, we can show that the set defined by $x_1 > y_1, \ldots, x_n > y_n$ has communication complexity equal to $n$. Grigoriev shows the last two bounds using rank based methods.

### 2.8 Nondeterministic Communication Complexity

From a complexity point of view, investigating nondeterministic algebraic communication complexity seems to be promising, too. Following the line of Boolean communication complexity and computational complexity, we introduce the concept of nondeterministic communication complexity, trying to characterize the communication complexity of a polynomial $f \in k[X, Y]$ in a different fashion. Eventually, we would like to link the two different points of view, being also able to recover information for the regular deterministic communication complexity from the nondeterministic one.

Roughly speaking, we allow players to “guess” the messages they have to send – this is where nondeterminism comes in. Recall that in the case of decision problems, we are given a polynomial $f \in$
$k[X, Y]$ and a fixed input $(X, Y)$ and we ask the question whether $f(X, Y) = 0$. Differently, in the nondeterministic setting the input is precisely chosen in the variety of $f$, however the task is to make sure that the players (or at least one of them) are convinced of this. To do so, they send nondeterministic messages. We now come to the precise definition.

**Definition 2.26 (Nondeterministic protocol).** Let $f \in k[X, Y]$ and $(X, Y) \in V(f)$. A nondeterministic protocol $P$ is a list of instructions for the two players to build messages $m_1, \ldots, m_r$ (depending on the input) with the goal of convincing at least one of them that $f(X, Y) = 0$ holds. It has to meet following properties.

1. For $1 \leq i \leq r = r(X, Y)$:
   - if $i \in M_{A \to B}$ holds, then $A$ guesses a message $m_i \in k(X, m_1, \ldots, m_{i-1})$ depending on $(X, Y)$.
   - if $i \in M_{A \leftarrow B}$ holds, then $B$ guesses a message $m_i \in k(Y, m_1, \ldots, m_{i-1})$ depending on $(X, Y)$.

2. For each input $(X, Y)$ there exists a polynomial certificate function $h \in k[X, m_1, \ldots, m_r]$ or $h \in k[Y, m_1, \ldots, m_r]$ with the property that $h$ vanishes in $(X, Y)$ and that for all $(X', Y') \in k^n \times k^n$, we have
   \[(X', Y') \in V(h) \Rightarrow (X', Y') \in V(f)\]

Let $r_{X,Y}(P)$ denote the maximum number of messages sent during the protocol $P$ on input $(X, Y)$, then the **nondeterministic communication complexity** $NC_{\text{dec}}(f)$ of $f$ is defined as

\[NC_{\text{dec}}(f) = \max_{(X, Y) \in V(f)} \min_P r_{X,Y}(P)\]

The following lemma follows immediately.

**Lemma 2.27.** For every $f \in k(X, Y)$:

\[C(f) \geq C_{\text{dec}}(f) \geq NC_{\text{dec}}(f)\]

**Proof.** A regular protocol with or without tests that computes $f$ can be seen as a nondeterministic protocol for the related decision problem. \qed
The interesting feature of nondeterminism is that we can close this chain of inequalities in certain cases.

**Theorem 2.28.** Let $k$ be algebraically closed and $f \in k[X, Y]$ irreducible, then

$$\text{NC}_{\text{dec}}(f) = \min_i C(f^i).$$

**Proof.** Let $(X, Y)$ be the input that attains the maximum in the definition of $\text{NC}_{\text{dec}}(f)$ and let $h \in k[X, m_1, \ldots, m_r]$ be its related certificate function. We assume without loss of generality that Alice is the player who is first convinced that the input lies in the variety of $f$. By definition we have $V(h) \subset V(f)$, so, by the Nullstellensatz we can write $f^i = g \cdot h$. We can assume that the polynomial $g$ is relatively prime to $f$, since $f$ is irreducible and $i > 0$ is an appropriate integer. The left-hand side is a multiple of $f$, this implies that $h$ is also divisible by $f$, so $h = p \cdot f^j$, where we choose $j > 0$ big enough such that $p$ is coprime with $f$. Since $k$ is algebraically closed, it follows that $p \in k \setminus \{0\}$, because otherwise it would be a polynomial with a zero $(X', Y') \not\in V(f)$, but on which $h$ vanishes, contradicting the definition of $h$. The evaluation of $h$ needs $C(pf^j) = C(f^j) \geq \min_i C(f^i)$ messages in the deterministic model.

By Lemma 2.27, the property that $\min_i C(f^i) \geq \text{NC}_{\text{dec}}(f)$ is always true and the claim is established.

The next corollary follows immediately and demonstrates the usefulness of nondeterminism.

**Corollary 2.29.** Let $k$ be algebraically closed and $f \in k[X, Y]$ irreducible. Then

$$\min_i C(f^i) = C_{\text{dec}}(f).$$

Of course, one needs to relate $\min_i C(f^i)$ to $C(f)$, which is currently only possible for the one-way model.

**Corollary 2.30.** Let $k$ be algebraically closed and $f \in k[X, Y]$ irreducible. Then

$$C^{-}(f) - 1 \leq \min_i C^{-}(f^i).$$

**Proof.** Let $j > 0$ be the exponent that leads to the minimum in $\min_i C^{-}(f^i)$, then

$$\min_i C^{-}(f^i) = C^{-}(f^j) = C^{-}(f^{j-1} \cdot f) \geq C^{-}(f) - 1.$$
The last inequality follows from the argument in the proof of Corollary 2.21 and Theorem 2.4. The claim follows then by the previous corollary.

2.9 Multi-party Communication Complexity

A natural generalization of our model is to consider more than two players communicating over an underlying network, modeled as a graph. There is a player residing at each node $i$, who has access to a vector $X^i \in k^n$ of $n$ values. Two players can directly send messages to each other, if they are neighbors in the given graph. As customary, messages are sent one after the other and they can be elements of the field extension generated from $k$ adding $X^i$ and the messages received so far. Our previous model is a special case of this one with $K_2$ as the underlying network. We denote by $C_G(f)$ the communication complexity of the function $f$ over the communication graph $G$.

We reduce this problem to the two-player case by employing a technique due to Abelson [2]. We present this idea by means of a generalized version of the Set-Disjointness Problem over an even cycle $C_r$ with $r$ nodes. The goal is to determine whether the sets induced by $X^1, \ldots, X^r$ are all pairwise disjoint, or, in other words, to resolve whether the following function evaluates to zero:

$$\text{Disj}_n(X^1, \ldots, X^r) = \prod_{\alpha \neq \beta} \prod_{i,j} (X_i^\alpha - X_j^\beta).$$

Suppose that the players are allotted consecutively from 1 to $r$ over the ring. An upper bound is easily established with the naive protocol: every player $j$ sends its input to the player 1 with $d(1,j) \cdot n$ total messages, where $d(i,j)$ is the length of the shortest path between player $i$ and $j$. Overall

$$C_{C_r}(\text{Disj}_n) \leq \sum_{i=1}^{r/2} i \cdot n + \sum_{i=1}^{r/2-1} i \cdot n = \frac{r^2}{4} n.$$

messages are sent.
To prove a lower bound, we reduce the problem to the two-player case. We denote by $N_{i,i+1}$ the total number of messages flowing through the edge between player $i$ and $i+1$ in an optimal protocol $P$, where $r+1 = 1$. Obviously we have $C_{Cr}(\text{Disj}_n) = \sum_{i=1}^{r} N_{i,i+1}$. Further, consider an edge-cut defined by the edges $\{i, i+1\}$ and $\{i+r/2, i+r/2+1\}$, for a $i = 1, \ldots, r/2$. This partitions the players into two sets $V_1$ and $V_2$ and we can write the function $\text{Disj}_n(X^1, \ldots, X^r)$ as follows:

$$\text{Disj}_n(X^1, \ldots, X^r) = \text{Disj}_n(V_1) \cdot \text{Disj}_n(V_2) \cdot \prod_{\alpha \in V_1} \prod_{\beta \in V_2} (X^\alpha_i - X^\beta_j),$$

where

$$\text{Disj}_n(V_k) := \prod_{\alpha \neq \beta} \prod_{i,j} (X^\alpha_i - X^\beta_j), \quad k = 1, 2.$$ 

Note that the last quantity on the right-hand-side corresponds to the Set-Disjointness Problem with two players, each having the union of the sets of the corresponding partition as the ground set, respectively and communicating along the two edges of the edge-cut. To solve this problem, $nr/2$ messages are necessary (Theorem 2.25). By repeating this argument for every $i$, we get:

$$N_{i,i+1} + N_{i+r/2,i+r/2+1} \geq \frac{r}{2} n, \quad i = 1, \ldots, r$$

in $P$. So, summing over $i$:

$$C_{Cr}(\text{Disj}_n) = \sum_{i=1}^{r} N_{i,i+1} = \frac{1}{2} \sum_{i=1}^{r} \frac{r}{2} n \geq \frac{r^2}{4} n.$$ 

The receipt described for showing the upper and lower bounds above can be used more generally. The reduction technique by Abelson is used to prove the lower bounds in Corollary 2.32.

An easy upper bound for an arbitrary graph $G$ is given by the naive protocol that sends all the inputs to one player. In some cases this is the best possible, as the following theorem shows.

**Theorem 2.31.** For every function $f$ over a graph $G$ on $r$ nodes, there is a protocol with at most $\overline{R}(G)rn$ messages, whereby

$$\overline{R}(G) := \frac{1}{r} \min_{u \in V(G)} \sum_{v \in V(G)} d(u, v).$$
is the average radius of $G_r$. Obviously we have $\overline{R}(G) \leq R(G)$, where $R(G)$ is the usual radius of a graph.

**Corollary 2.32.** For the multi-party communication complexity of the Set-Disjointness Problem on general networks, we have:

\[
C_C(r \text{Disj}_n) = \frac{r^2}{4} n, \quad r \text{ even}
\]

\[
\frac{r^2 - r}{4} n \leq C_C(r \text{Disj}_n) \leq \frac{r^2 - 1}{4} n, \quad r \text{ odd}
\]

\[
C_T(r \text{Disj}_n) = \overline{R}(T_r) rn
\]

\[
C_P(r \text{Disj}_n) = \frac{r^2}{4} n, \quad r \text{ even}
\]

\[
C_P(r \text{Disj}_n) = \frac{r^2 - 1}{4} n, \quad r \text{ odd}
\]

\[
\frac{r}{2} n \leq C_K(r \text{Disj}_n) \leq (r - 1)n
\]

where $C_r, T_r, P_r$ and $K_r$ denote a cycle, a tree, a path and a complete graph on $r$ vertices, respectively.

**Proof.** Almost all the presented bounds follow straightforwardly. We show the bound:

\[
C_T(r \text{Disj}_n) \geq \overline{R}(T_r) rn.
\]

Let $V = V(T_r), E = E(T_r)$, and $u \in V$ be a vertex which attains the minimum in the definition of $\overline{R}(T_r)$. For every edge $e \in E$, define $T_1^u(e)$ and $T_2^u(e)$ to be the two subtrees induced by the components of $T_r \setminus \{e\}$, where $V(T_1^u(e))$ is the subtree containing $u$. Further let $w_u(e) := |V(T_2^u(e))|$. Then $r\overline{R}(T_r) = \sum_{e \in E} w_u(e)$.

We claim that $|V(T_1^u(e))| \geq |V(T_2^u(e))|$ for all $e \in E(T_r)$. Otherwise, we can find an edge $e'$ such that $|V(T_2^u(e'))| > |V(T_1^u(e'))|$. Note that every edge along the path from $u$ to $e'$ satisfies the same inequality, so we assume that $e' = \{uu'\}$. Then we have

\[
\frac{1}{r} \sum_{v \in V} d(u', v) = \frac{1}{r} \sum_{v \in V} (d(u, v) + |V(T_1^u(e))| - |V(T_2^u(e))|)
\]

\[
< \frac{1}{r} \sum_{v \in V} d(u, v) = \overline{R}(T_r)
\]
in contradiction with the minimality of \( u \). So \( w_u(e) \) is the least number of vertices of the components of \( T_r \setminus \{ e \} \). By the Abelson’s reduction, it follows that \( N_e \geq n w_u(e) \), for all \( e \), where \( N_e \) is the number of messages over \( e \) of a given protocol \( P \) for the Set-Disjointness Problem. Hence

\[
C_{T_r}(\text{Disj}_n) = \sum_e N(e) \geq n \sum_e w_u(e) = nrR(T_r)
\]

for every protocol \( P \).

\[ \square \]

### 2.10 Applications

As hinted before, the high communication complexity for solving the Set-Disjointness Problem in the boolean model is the key ingredient used by Feigenbaum et al. [40] to show the communication inefficiency of distributed mechanisms for the Multicast Cost Sharing Problem that fulfill determined properties. Unfortunately, we cannot use the same approach in our case, because our results do not imply that the algebraic communication complexity of the Set-Disjointness Problem is maximal if the players pick their subset from a finite basis set. To settle the problem we still take inspiration from [40], but we proceed in a more direct fashion.

We briefly review the definitions and techniques of [40], that we adapt for our setting. We denote by \( P \) the set of users in a instance of the Multicast Cost Sharing Problem. Let \( g : 2^P \to \mathbb{R}_{\geq 0} \) be the cost-sharing function, i.e., for \( S \subset P \), \( g(S) \) is the non-negative cost charged to every player of \( S \), if \( S \) is the selected receiver set. The function \( g \) is called strictly cross-monotonic, if \( g(S \cup \{ j \}) < g(S) \) for all \( S \subset P \) and \( j \notin S \). A (non-necessarily) strictly cross-monotonic function induces a cost sharing mechanism: the receiver set is the unique maximal subset \( S \subset P \) such that \( g(S) \) is not higher than the type of each player in \( S \) and the cost charged to all of them is precisely \( g(S) \).

A mechanism that solves the Multicast Cost Sharing Problem collects the values claimed by the players and aggregates them to come to a decision. If we allow only algebraic operations and comparisons to the mechanism, we can see the mechanism as an algebraic communication protocol. Our strategy is to show that the mechanism necessarily computes a function with high communication complexity during
its execution. We argue that already the question whether the receiver set is empty or not cannot be answered with few messages. The function that models this question in a decision tree can be the sum of the characteristic function of every player, which evaluates to one, if the player is in the receiver set, or to zero otherwise. For a technical reason, we take the complement of this function, i.e., we take a function that evaluates to zero if and only if the receiver set is non-empty. Obviously, this assumption does not change the communication complexity of the function because a decision tree of a function can be easily turned into a decision tree of its complement, and vice versa. We call the described function $f_g$.

The network is a path with the property that the node that determines the receiver set is the middle node and that there are $|P| = 2r$ players in the network, $r$ residing on each side. We call the players in one side $a_1, \ldots, a_r$ and the players on the other side $b_1, \ldots, b_r$, respectively. Due to the strictly cross-monotonicity of $g$, we can fix a quantity $d > 0$ with the property that $g(S \cup \{j\}) < g(S) - d$, for all $S \subset P$ and $j \notin S$. Further, we choose a vector $(\varepsilon_1, \ldots, \varepsilon_r) \in (0, d/2)^r$. To fully specify an instance of Multicast Cost Sharing Problem, we have to set the players’ types. First, we define the following quantities. For $i \in \{1, \ldots, r\}$ we set:

$$t_i^H = g(\{a_1, b_1, \ldots, a_i, b_i\}) - \varepsilon_i,$$
$$t_i^L = g(\{a_1, b_1, \ldots, a_i, b_i\}) - d + \varepsilon_i.$$

Note that $t_i^L < t_i^H$ holds for all $i$. Let us pick two subsets $S_1, S_2 \subset \{1, \ldots, r\}$, with the property that $S_1 \cup S_2 = \{1, \ldots, r\}$. We define the players’ type as follows:

$$t_{a_i} = \begin{cases} t_i^H, & \text{if } i \in S_1 \\ t_i^L, & \text{if } i \notin S_1 \end{cases}$$
$$t_{b_i} = \begin{cases} t_i^H, & \text{if } i \in S_2 \\ t_i^L, & \text{if } i \notin S_2 \end{cases}, \quad i \in \{1, \ldots, r\}.$$  

Notice that the instance of Multicast Cost Sharing Problem that we describe here is similar to the one described in Feige et al [40] (Theorem 1). By the same argument as theirs, it is not hard to prove that the receiver set is empty if and only if $S_1$ and $S_2$ are disjoint. In other words, the receiver set is non-empty if and only if $t_{a_i} = t_{b_i}$. This implies that $t_{a_i} - t_{b_i}$ divides $f_g$. Note that this holds for every choice of $(\varepsilon_1, \ldots, \varepsilon_r)$ in the open set $(0, d/2)^r$ and hence globally (see [115]). We are in a situation similar to Corollary 2.25 and
an identical argument settles the high communication complexity of $f_g$ and, consequently, of the Multicast Cost Sharing Problem if the corresponding budget-balanced cost-sharing function is strictly cross-monotonic. A result of [40] implies that this restriction is weak and is fulfilled by many instances, including those for which all the link costs in the multicast tree are non-zero. As a last remark, note that the assumption that the cost-sharing function is equal to all the players is not restrictive. For instance, it is the case if the cost of the link from every receiver to the sender in the multicast tree is the same. Summarizing, we have the following result.

**Theorem 2.33.** There is a tree with $r$ players such that every algebraic algorithm with comparisons and equality tests that computes a group-strategyproof and budget-balanced mechanism for the Multicast Cost Sharing Problem sends at least $n$ messages over a linear number of links.

As a second application of our results to the field of Mechanism Design, we consider a distributed version of a combinatorial auction with single minded bidders, see e.g. [75, 18]. We have $n$ players and a collection of objects. There is a partial order on these objects. Each player bids on exactly one set of objects. A selection algorithm receives the bids of the players and selects those players that get an object. Such an algorithm is called monotone if a selected player that makes a higher or equal bid on the same set of objects (or a subset thereof) in a second instance is still selected (assuming that all other players’ bids are the same). From such an algorithm, one can construct a truthful mechanism via the critical value scheme. In short, the critical value scheme assigns the objects in such a way that the revenue is maximized and charges the players by the quantity known as the critical value. The critical value of a bidder is the value $\alpha$ that divides winning bids from losing bids: if a player bids more than $\alpha$ she wins, if she bids less than $\alpha$, she loses, see [75, 18] for details. We now assume that the players are distributed in a network. The players can send their bids along the links to a predefined node at which the selection process takes place. Consider the following scenario: we have a network of two subgraphs connected by one link. Each subgraph contains $n$ players. We have $n$ objects and one player of each subgraph bids on one of them. The mechanism that assigns the objects to the players takes a decision on every pair and picks the player with the highest bid. In particular it is able to determine whether the
bids in one subgraphs are all larger than the corresponding bids in the other subgraph. This corresponds to the task of determining the membership in the semialgebraic set defined by \( x_1 > y_1, \ldots, x_n > y_n \), which has communication complexity \( n \) (see the remark after Corollary 2.21).

**Theorem 2.34.** Any distributed truthful critical value scheme for a combinatorial auction with single minded bidders needs to send at least \( n \) messages over linearly many links in the worst case.

We get linearly many links by replacing the edge between the subgraphs by an appropriate path.

### 2.11 Open Problems

From the point of view of distributed mechanism design, it would be definitely appealing to find more examples to which the techniques presented in this chapter can be applied.

From the point of view of algebraic communication complexity, the most interesting open problem is to relate the complexity of a polynomial \( f \) to the complexity of a product \( g \cdot f \) with an arbitrary non-zero polynomial \( g \) in the two-way model. A strong result for this question would directly imply lower bounds for the decision problem (Lemma 2.20). For instance, if the rank bound of Abelson [2] yields a lower bound of \( q \) for the complexity of \( f \), then it yields a lower bound for \( q - 2 \) for \( g \cdot f \); under some strict assumptions on the functions. However, there are cases where the rank bound is not tight at all. It is not clear to us how to prove the general case.

We conclude with the following conjectures.

**Conjecture 2.35.** For all irreducible polynomials \( f \) over algebraically closed fields, it holds:

1. \( C(f) \geq C(gf) - 1 \) for every coprime polynomial \( g \neq 0 \) and
2. \( C(f) \geq C(f^j) - 1 \) for every \( j \geq 1 \).

If both conjectures were true, then the decision complexity and the nondeterministic complexity would be closely related to the communication complexity of computing \( f \). The case \( f = x_1 y_1 + x_1^2 x_2 y_2 \), \( g = x_2 \) shows that \( C(f) \geq C(gf) \) is in general not true.
Part II

Computing

Distributed Algorithms for Wireless Ad Hoc Sensor Networks
Prologue

Milan, 1506 – Mona Lisa is looking at Leonardo, who is working on some project.

_Mona Lisa:_ “Leonardo, you look very concentrated.”

_Leonardo:_ “I’m thinking about a problem. It’s nothing really important, but it’s interesting nevertheless.”

_Mona Lisa:_ “Do you mind telling me what it is about?”

_Leonardo:_ “Are you aware of the legend of Archimedes and the battle of Syracuse? Back in 212 BC Syracuse was besieged by Marcus Claudius Marcellus. The Romans were attacking from the sea. Archimedes’ idea was to set the Roman fleet to fire from afar. His weapon was a large set of burning glasses.”

_Mona Lisa:_ “Clever. A burning glass is a paraboloid lens that concentrates the sun’s rays to its focal point. It can generate very high temperatures. To possibly achieve what the legend says, you need to aim at a precise point of every ship with many burning glasses.”

_Leonardo:_ “Correct. According to the legend, Archimedes succeeded in burning down the Roman fleet in this way. However Syracuse was taken anyway.”

_Mona Lisa:_ “What is the problem you are thinking about?”

_Leonardo:_ “On the practical side, I think that it is very unlikely that a number, even if large, of burning glasses would be able to ignite a wooden ship at a considerable distance. Still, suppose for a second that this is possible. The scenario at the bay of Syracuse suggests an intriguing theoretical problem.”

_Mona Lisa:_ “I’ll love to hear that.”

_Leonardo:_ “Well, you should imagine a large fleet approaching the harbor. To attack so many ships, you need to dispose a large number of men, each with a burning glass, along the coast.”

_Mona Lisa:_ “Yes, I think that this is a correct scenario.”
**Leonardo:** “Every men should be made responsible to one ship. He needs to know which ship to aim his burning glass at.”

**Mona Lisa:** “I’m starting to see the problem now. In practice there is no time for a coordinator to go from one side of the coast to the other and tell everybody where to aim at.”

**Leonardo:** “This is exactly the point. Every ship has to be targeted by a certain number of burning glasses, but the defenders cannot agree on a strategy in a global way. They need to decide only according to what the men close to them are doing.”

**Mona Lisa:** “Eventually you want to achieve some sort of global coordination in a very localized way.”

**Leonardo:** “Exactly. I think that ants do this very well.”

**Mona Lisa:** “I agree with you, it is an appealing problem. Have you found a solution yet?”

**Leonardo:** “I’m still working on it.”

*Note from the author.* Different studies tried to verify whether it is possible to ignite a piece of wood with a burning glass from afar. The conclusions of these studies are mostly contradictory. For further information, see [122].
Chapter 3

Introduction

You see, wire telegraph is a kind of a very, very long cat. You pull his tail in New York and his head is meowing in Los Angeles. Do you understand this? And radio operates exactly the same way: you send signals here, they receive them there. The only difference is that there is no cat. – Albert Einstein (March 14, 1879 - April 18, 1955), German, Swiss and American physicist and Nobel Prize winner in physics.

3.1 Motivation

The design of distributed algorithms for problems appearing in the realm of ad hoc and wireless sensor networks has gained considerable momentum in recent years. One of the main reasons for this is that the distributed paradigm lies in the bridge connecting theory and practice. Several problems from theory have been considered in the distributed setting and practical implementations have been successful. This variety of applications ranges from agricultural management targeted at Indian farmers [98] to body area networks for human activity recognition [60]. The advantages of a wireless sensor network with respect to a more classical network are many-sided. A wireless sensor network is typically self-organizing and hence more robust against changes in topology and sensor failures. The requirement of only little infrastructure makes the deployment of such a network cheaper. The huge
technological advancements over the last years are making wireless sensor networks very attractive by enabling the development of more and more cheaper, smaller and sophisticated sensors. To assist these developments, it is becoming even more important to find solutions to the fundamental challenges in employing such networks.

From a theoretical perspective, wireless sensor networks offer an appealing computational model. The challenges are related particularly to the size, the dynamism, the cost constraints and the decentralization of such systems. These aspects demand for novel theoretical models and algorithmic solutions. Ideal algorithms designed for wireless sensor networks are highly scalable, fault-tolerant and operate in a distributed fashion to deal with the lack of a central authority that organizes and directs the operations in the system. In many deployments of wireless sensor networks in real scenarios, the topology frequently changes due to the mobility or to the possible failures of the sensors. In such a setting distributed algorithms are often more efficient and respond more rapidly to variations of the network than centralized algorithms that continuously send all the information to a single node. Further, the important energy constraints of wireless ad hoc networks ask for taking into account both the computation and transmission time (and thus the “active time”) of the nodes required to complete the desired tasks.

A wireless sensor network typically consists of a large number of sensors (nodes), each having wireless communication capability, modest computational power, and a battery with limited capacity. Two nodes can communicate directly if they are within mutual communication range. If they are not within mutual communication range, they talk by letting intermediate nodes forward the messages. The deployment of such a network is often done by randomly spreading sensors on the area of interest. To model the particular connectivity structure inherent to this type of deployment, usually a restricted class of graphs is considered. By far the most prominent such class is the class of Unit Disk graphs. However, for various reasons Unit Disk graphs are a very idealized model of real wireless networks, and many generalizations have been proposed \cite{93,111}. In this thesis, we adopt the class of Growth-Bounded graphs, which is widely considered to be general enough to model wireless sensor networks since it contains Unit Disk graphs and many other relevant models as special cases. The omnidirectional antenna of a prototypical wireless sensor
broadcasts a message to all sensors within its communication radius. It is likely that most of the sensors that are within this ball are close to each other and, in turn, can directly communicate. Growth-Bounded graphs capture substantially this property.

A well-established theoretical communication model used to deal with distributed systems is the message-passing model described in more detail in Section 3.2.3. In a variant of the model, called LOCAL, a node can send a message of any size to each of its neighbors in each round. Thus, a distributed algorithm whose output depends only on the topology of the network and on the labels of the nodes can be seen as a centralized algorithm that is executed independently in every node of the network. This is because in every single run, a node can broadcast the complete information about the topology of the network that it has gathered so far. This operation allows the nodes that receive the information to extend their knowledge of the network correspondingly. This operation is repeated in every round. More precisely, the input to the algorithm executed at node \( v \) at the time step \( k \) is given by the graph induced by the nodes within a distance \( k \) to \( v \) and is called the \( k \)-view. \(^1\) After the execution of the round \( k + 1 \), the algorithm has a bigger input at disposal (the \((k + 1)\)-view as opposed to the \( k \)-view). After a certain number of rounds, the input enables the computation of an output. In other words, the information available to a node is initially very localized and the piece of new information that becomes available at every round – the view – depends on the network topology. For this reason, the problems in such a setting exhibit the locality property common to this thesis. Section 3.2.3 presents more details about these concepts.

Many distributed algorithms designed to execute various fundamental tasks of wireless sensor networks have been proposed in the literature. They solve cardinal problems such as routing \([5, 121]\), clustering \([10, 67]\), topology control \([88]\) and data gathering \([62]\) – only to mention a few. There is a clear feature shared by many distributed algorithms: they employ some basic subroutines that, more often than not, aim to solve classical graph-theoretical problems, such as finding a Maximal Independent Set or a Spanning Tree of the network. For instance, if a Maximal Independent Set or a Coloring with few colors \(^2\) of a network are known, then the operation in the network

\(^1\)To be precise, we do not include the edges with both endpoints at distance exactly \( k \) to \( v \) in the view of \( v \).

\(^2\)By coloring we intend the coloring of a graph in the classical graph-theoretical
can be scheduled in such a way that no two neighboring nodes operate in parallel, which, in turn, prevents conflicts. Consequently it is of primary importance to understand how difficult graph-theoretical problems in a distributed setting are in order to be able to fully capture the principles behind the organization and the operation of wireless ad hoc networks. Countless results have already been presented by the distributed computing community to assess the locality of such problems – \cite{33, 58, 63, 64, 65, 90, 91, 92, 99} are only selected examples. For instance, Linial \cite{77} uses advanced graph-theoretical techniques to study the locality of distributed systems and discusses the importance of the efficient local computation of a Maximal Independent Set. In this part we study three such problems, namely the Maximal Independent Set Problem, the Minimum Connected Dominating Set Problem and the Small Connected Spanning Subgraph Problem.

\section{Definitions and Model}

\subsection{Terminology}

As mentioned in the introduction, in this part we deal with distributed (or local) algorithms. In the literature, there is no consensus about the precise meaning of the adjective \textit{local}. In some papers (e.g. \cite{91}) a local algorithm is a distributed algorithm that runs in constant time for every instance, independently of the size of the network. The other common interpretation is to consider \textit{local} and \textit{distributed} as synonyms in this context \cite{10}. In this thesis, we are concerned with some problems for which constant-time algorithms are provably impossible. Hence we adopt the latter interpretation.

We model a network as an undirected simple graph $G = (V, E)$ on $n$ nodes and $m$ edges. For $V' \subset V$, we denote by $G[V']$ the subgraph of $G$ \textit{induced} by $V'$: the vertex-set of $G[V']$ is $V'$ and the edge-set consists of the edges of $G$ with both endpoints in $V'$. The \textit{distance} between two nodes $u, v \in V$ is the number of edges of a shortest path connecting $u$ and $v$ in $G$ and is denoted by $d(u, v)$. The distance between two sets $S_1, S_2 \subset V$, denoted by $d(S_1, S_2)$, is the least distance between two nodes $s_1 \in S_1$ and $s_2 \in S_2$. The maximal sense, i.e., a labelling of the vertices of the graph, in such a way that no two neighbors have the same label.
distance among two nodes in $G$ is called the \textit{diameter} $d(G)$ of $G$.

The \textit{neighborhood} of a node $u \in V$ is

$$N(u) := \{u\} \cup \{v \in V \mid (u,v) \in E\},$$

i.e., it is the set of nodes that are adjacent to $u$ and $u$ itself. The \textit{neighborhood} of a set $S \subseteq V$ is defined as $N(S) := S \cup \{u \in V \mid (u,v) \in E \}$, for a $v \in S$. We define the $i$-\textit{neighborhood} $N_i(S)$ of $S$ recursively as $N_0 := S$ and $N_i(S) := N(N_{i-1}(S))$, for $i > 0$. The \textit{reduced} neighborhood $\Gamma_j(v,V')$ is defined for all $j \geq 0$ as $N_j(v)$. The \textit{degree} $d_v$ of a node is the number of nodes adjacent to it. For technical reasons, we introduce the \textit{size} $\bar{d}_v$ of a node $v \in V$. The size is the cardinality of $N(v)$, that is, the degree of $v$ plus one. $\Delta$ denotes the maximum degree of any node in $G$. For convenience, we define $\bar{\Delta} := \Delta + 1$.

A set $T \subseteq V$ is said to be \textit{independent} in $G$ if no two nodes $u, v \in T$ are neighbors in $G$. An independent set $T$ is a \textit{Maximal Independent Set} of $G$ (MIS), if no superset $T' \supset T$ is independent in $G$. The problem of finding a MIS is called the Maximal Independent Set Problem. Further, $T$ is a \textit{k-ruling set} if every node of $G$ is within distance $k$ from some node of $T$. Note that a MIS is a 1-ruling set. A \textit{Maximum Independent Set} (MaxIS) is a MIS of largest size. For a subset $S \subseteq V$, MaxIS$(S)$ is a Maximum Independent Set on the induced subgraph $G[S]$. A \textit{Dominating Set} (DS) $D$ for a given graph $G = (V,E)$ is a subset $D \subseteq V$ such that every node $v \in V \setminus D$ has a neighbor in $D$. Note that a MIS is always a DS. A \textit{Connected Dominating Set} (CDS) for a given graph $G = (V,E)$ is a dominating set $M$ with the additional requirement that $G[M]$ is a connected subgraph of $G$. A \textit{Minimum} (or \textit{Optimal}) \textit{Connected Dominating Set} (MCDS) is a CDS of smallest possible cardinality, and the Minimum Connected Dominating Set Problem is the problem of providing such a MCDS for a given graph. The function $C(A), A \subseteq V$, returns a smallest subset of $N(A)$ that dominates all nodes in $A$ and induce a connected graph. Observe that $C(A)$ might contain nodes that are not in $A$ (but that are in $N(A)$). A \textit{Spanning Tree} $T$ of a graph $G$ is a connected, acyclic subgraph of $G$ that spans all the vertices of $G$.

Throughout the thesis, we use $\log x$ to denote the binary logarithm of $x$, and $\ln x$ for the natural logarithm of $x$. $\log^* n$ – read \textit{log-star}
– is the smallest non-zero \( t \in \mathbb{N} \) such that the \( t \)-fold iteration of the logarithm on \( n \) yields a value smaller than 2.

### 3.2.2 Graph Models

In this thesis, we model networks as Growth-Bounded graphs. A graph \( G \) is called a \textit{\( f \)-Growth-Bounded graph} if any independent set of the \( r \)-neighborhood graph \( G[N_r(u)] \) of any node \( u \) in \( G \) has size at most \( f(r) \). The function \( f \) is called the \textit{bounding function} of \( G \). Note that the function \( f \) depends solely on \( r \). While discussing the asymptotic running time of the algorithms, we assume tacitly that the graphs that represent the input to the algorithm have the same bounding function. Intuitively, the class of Growth-Bounded graphs captures a characteristic feature of a wireless ad hoc network, namely that if some nodes are located close to each other, then most of them can directly communicate to each other. If the bound function of a Growth-Bounded graphs \( G \) is a polynomial, then we call \( G \) a \textit{p-Growth-Bounded graph}. Chapter 5 deals only with \textit{p-Growth-Bounded} graphs.

A subclass of Growth-Bounded graphs are \textit{Unit Disk graphs}. Unit Disk graphs are often used to model wireless communication networks. A graph \( G = (V, E) \) is a Unit Disk graph if it can be represented by placing a point \( p_v \) for each node \( v \in V \) on the Euclidean plane \( \mathbb{R}^2 \), such that an edge \( (u, v) \in E \) exists if and only if the Euclidean distance \( \|p_v - p_u\|_2 \) is at most 1. Observe that Unit Disk graphs are \( p \)-Growth-Bounded with a bounding function \( f(r) \in \mathcal{O}(r^2) \). To see this consider any node \( v \) and any independent set \( I \subset N_r(v) \), for a \( r \geq 0 \). The corresponding points of two (independent) nodes in \( I \) must have distance greater than 1. Thus each point corresponding to a node in \( I \) exclusively occupies a disk of radius \( 1/2 \) with an area of \( (1/2)^2 \pi \). As all these disks must lie inside a circle of radius \( r+1/2 \), it follows that \( |I| \leq \frac{(r+1/2)^2 \pi}{(1/2)^2 \pi} = 4r^2 + 4r + 1 \). Furthermore, most other graph classes used to model wireless ad-hoc networks such as Quasi Unit Disk graphs, Cover Area graphs and other intersection graphs are Growth-Bounded [68][93].
3.2.3 The Message-Passing Model

In this thesis we employ the synchronous message-passing model as a framework for our algorithms. The message-passing model is a well-established paradigm to describe a network of independent entities that communicate without a global perception of the network. The entities learn the network state and topology in a neighborhood of increasing radius by means of messages. This exactly reflects the thesis’ flavor of locality. The message-passing model is equally suited for networks of very different nature; examples are wireless sensor or ad hoc networks, the Web graph, Peer to Peer graphs, and so on.

With applications on wireless sensor networks in mind, we limit the class of network topologies that we investigate. The network is modeled as a Growth-Bounded graph $G = (V, E)$. The nodes of the network are small computational devices, with their own processor, memory, battery and wireless equipment. Two nodes can directly communicate if an edge is present between the corresponding nodes in the graph $G$. Messages between distant nodes must be forwarded via intermediate nodes. The nodes are solely provided with local connectivity information, i.e., any node knows only about its direct neighbors. Consequently, they may not know the full network topology (nor the number of nodes) and cannot sense the (local) embedding of the deployed network. For instance, they cannot measure distances between nodes. Further, each node has a unique identifier of $O(\log n)$ bits that can be possibly transmitted in a message. In a real network, the identifier might be an IP address or a MAC address.

In each round, each node can send a message of any size to each of its neighbors. In some cases, we are able to bound the message size to $O(\log n)$ bits. In the literature, the presence or absence of the limitation on the message size gives rise to two profoundly different models, called CONGEST and LOCAL [101], respectively. The main difference lies in the fact that in the CONGEST model, nodes can send only a constant number of node identifiers per round. If a node has a linear number of neighbors $\Theta(n)$, say, it needs $\Theta(n/\log n)$ rounds to inform its neighbors about its 1-hop neighborhood. On the other hand, a single message is sufficient in the LOCAL model. More generally, after $k$ rounds of an algorithm in the LOCAL model, a node $v$ learns its $k$-view, i.e., the graph that is induced by its $k$-neighborhood $N_k(v)$ but without the edges between nodes at distance exactly $k$ (see Figure 3.1). This reasoning holds obviously in the other direction too:
in order for a node to be able to build its $k$-view, $k$ rounds are necessary. Due to this analogy between a distributed algorithm and the view of a node, we say that a distributed algorithm is merely a function that produces an output (available only to the node that runs the algorithm) from the view of a certain depth. The maximal depth of a view required by an algorithm to terminate corresponds to the running time of the algorithm. Another consequence of this fact is that the interesting algorithms in the LOCAL model are those which run in $O(d(G))$ rounds, where $d(G)$ represents the network diameter of $G$. In fact, after $d(G)$ rounds, every node knows the network completely.

![Figure 3.1: The filled nodes and the solid edges in the graph depict the 2-view of the vertex $v$. Note that the edge $e$ is not present in the 2-view, because its endpoints are at distance exactly two to $v$.](image)

Assuming that a physical layer for direct communication of neighbors has already been established, we do not consider collisions or other transmission failures. Hence, every message arrives reliably and instantaneously.

We study the problems at hand in synchronous networks. In a synchronous network, the nodes wake up at the same time. A global synchronization mechanism determines discrete points in time. Two consecutive such points in time define a round, which are identical in the whole network. The execution of an algorithm is organized in these rounds, which, in turn, are divided into a computational and a transmission phase that take place concurrently for every node. We assume that the length of a single round permits to every node to perform the necessary computations and a single transmission (both sending and listening). An asynchronous network has no such synchronization mechanism. Nodes wake up at arbitrary points in time and have a dif-
3.2. Definitions and Model

Different transmission and computational speed. We simply assume that these operations are performed within one time unit. A distributed algorithm that assumes a synchronous network can comply with an asynchronous setting at the cost of an increased time or message complexity [8][112].

The complexity of an algorithm is the number of rounds required from start to completion in the worst-case (with respect to the network topology and to the assignment of the identifiers to the nodes). Other aspects might be of interest, such as the total number of messages sent in the network, the maximal number of messages that flow on a single link, and so on. Notice that we neglect the amount of computation that every node has to carry out.

We refer the reader interested in the message-passing model to the book [7].
Chapter 4

The Maximal Independent Set Problem –
A Randomized Algorithm

*The single biggest problem in communication is the illusion that it has taken place.* – George Bernard Shaw (July 26, 1856 - November 2, 1950), Irish playwright, critic and Nobel Prize winner in literature.

4.1 Introduction and Motivation

The question of how to compute a Maximal Independent Set (MIS) in a distributed setting has been studied for many years [10, 26, 77, 78, 81]. This problem asks for a subset $S$ of the vertices of a graph with the property that (1) no two vertices of $S$ are connected by an edge, and that (2) there is no strict superset of $S$ with the same property. Its importance is twofold. On a practical level, as hinted in the introduction, it serves as a basic building block for many distributed algorithms. On the theoretical level it is the classical problem which is related to the concept of symmetry breaking. Furthermore it is a nice
example of a graph-theoretical problem that admits a trivial greedy
solution in a centralized setting, but is nevertheless a challenging re-
search topic for the distributed computation community.

With the advent of wireless ad hoc and sensor networks, the study
of distributed MIS computation has received further attention [87,
[100]. This is because algorithms for topology control and routing
often construct a MIS in an initial phase [47, 28, 5, 103]. Typically,
the time required for this phase dominates the total running time for
the algorithms. For example, constructing a \((1+\varepsilon)\)-approximate Minimum Dominating Set is possible in time \(O(T_{\text{MIS}} + \log^* n/\varepsilon^{\Omega(1)})\) in Growth-Bounded graphs, where \(T_{\text{MIS}}\) is the time for a MIS com-
putation [64]. As we will show in the next chapter, this holds for a
\((1 + \varepsilon)\)-approximate Minimum Connected Dominating Set as well.

Various approaches for distributively computing a MIS in Unit
Disk graphs are available. However, many of them have a worst case
running time of \(O(n)\), e.g. [5]. As an interesting variant, [66] consid-
ers the distributed computation of a MIS in a setting where nodes are
embedded in a metric space, and some geometric information about
the embedding are available to the nodes. It is shown that if each node
knows the distance to each of its neighbors, one can compute a MIS
in \(O(\log^* n)\) rounds. If nodes are located in the Euclidian plane, and
each node knows its position, a MIS can even be computed in constant
time.

Thus, very fast running times can be achieved under the above as-
sumptions. However, it is not always the case that these features are
implementable in a practical scenario. It may be that the energy these
devices require is unacceptable for the wireless devices at hand, or
these features (e.g. measuring distance) may simply not be available,
due to hardware constraints or the type of deployment of the sensors,
or in presence of an abstract graph like the Web graph. In any case,
a distance measuring device might increase the hardware costs con-
siderably. In this respect, one might want to compute a MIS without
using these features, yet being similarly fast. We therefore propose
to use randomization instead of distance measuring. This seems like
a natural choice as, intuitively, randomization should ease symmetry-
breaking. This can be seen with the following example. In a greedy
algorithm to compute a MIS, every node joins the solution if it has the
least label among its active neighbors and if no neighbor has joined
the solution so far. If the latter is the case, the node becomes passive.
4.2. Related Work

If we apply this algorithm to a path on \( n \) active nodes where the labels are ordered increasingly along the path, the algorithm requires \( \Theta(n) \) steps to complete, which is unacceptable. On the other side, if the labels are chosen randomly (simulated by a randomized algorithm) the running time enormously benefits, since the length \( L_n \) of the longest monotone increasing or decreasing subsequence of a random permutation fulfills \( \lim_{n \to \infty} \mathbb{E} \left[ L_n / \sqrt{n} \right] = 2 \) \cite{79, 120} and is highly concentrated around its mean \cite{43}. Hence the algorithm operates concurrently in many parts of the path – rather than in one – and terminates in \( O(\sqrt{n}) \) rounds with high probability. In this chapter we use randomization, but we exploit it in a different way.

A distributed algorithm for the MIS problem runs on every node independently and assigns the value 1 to a node if the node joins the solution and 0 otherwise. We say that the distributed algorithm solves the problem, if the set of nodes with value 1 at the algorithm termination form a MIS of the original graph.

4.2 Related Work

Randomization has proved successful in computing a MIS in general graphs: while the fastest known deterministic algorithm for general graphs due to Panconesi and Srinivasan \cite{99} runs in \( O\left(n^d\sqrt{1/\log n}\right) \) rounds (where \( d \) is a constant), Luby proposed an almost exponentially faster \( O(\log n) \) randomized algorithm \cite{81}. This is close to the lower bound \( \Omega(\sqrt{\log n / \log \log n}) \) given in \cite{65}, that holds for randomized algorithms too.

In contrast, a deterministic MIS algorithm with a moderate running time of \( O(\log \Delta \log^* n) \) exists for Growth-Bounded graphs \cite{63} (where \( \Delta \) is the maximum degree of the graph), while the best known lower bound for this class is \( \Omega(\log^* n) \) (even for randomized algorithms) \cite{78, 90}. This raises the question whether randomization enables a running time improvement of the same magnitude as in general graphs. Our results, summarized in the next section, show that using randomization, one can indeed compute a MIS in Growth-Bounded graphs almost as quickly as with the help of distance information.
Table 4.1: Summary of known results on the distributed computation of a MIS in different graph classes: (a) General graphs, (b) Growth-Bounded graphs, (c) Constant-Degree graphs, (d) Unit Disk graphs with distance information. The lower bounds hold also for randomized algorithms.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>$\Theta(n^{d\sqrt{1/\log n}})$</td>
<td>$\Theta(\log n)$</td>
<td>$\Omega(\sqrt{\log n/\log \log \log n})$</td>
</tr>
<tr>
<td>(b)</td>
<td>$\Theta(\log \Delta \log^* n)$</td>
<td>$\Theta(\log \log n \log^* n)$</td>
<td>$\Omega(\log^* n)$</td>
</tr>
<tr>
<td>(c)</td>
<td>$\Theta(\log^* n)$</td>
<td>-</td>
<td>$\Omega(\log^* n)$</td>
</tr>
<tr>
<td>(d)</td>
<td>$\Theta(\log^* n)$</td>
<td>-</td>
<td>$\Omega(\log^* n)$</td>
</tr>
</tbody>
</table>

Only very recently there have been two breakthrough papers. In the first paper, Schneider and Wattenhofer [113] describe a deterministic algorithm that computes a MIS in the class of Growth-Bounded graphs in an asymptotically optimal number of rounds, i.e., within $\Theta(\log^* n)$ rounds. The second paper by Barenboim and Elkin [11] shows the first sublogarithmic algorithm with a proved running time of $\Theta(\log n/\log \log n)$ rounds that finds a MIS in the rich class of graphs of bounded arboricity, a large family of graphs that includes graphs of bounded degree, planar graphs and graphs of bounded genus among others.

4.3 Summary of Results

Our main contribution is a synchronous randomized distributed algorithm for computing a MIS in Growth-Bounded graphs with $n$ vertices running in $\Theta(\log \log n \log^* n)$ rounds with high probability. Specifically, the probability that the algorithm requires more than $c \log \log n \log^* n$ rounds for a certain constant $c$ is at most $\Theta(1/n^k)$ for a $k > 3$. The nodes require only connectivity information about the graph, and all messages are of size $\Theta(\log n)$. Hence, the algorithm complies with the CONGEST model.
This running time compares on one side with the running time $O(\log \Delta \log^* n)$ of the best known deterministic algorithm designed for Growth-Bounded graphs [63], and on the other side with the running time of $O(\log^* n)$ of the deterministic algorithm which assumes that the graph is embedded in a metric space, and requires the nodes to know the geometric distance to their neighbors [66]. In view of the $\Omega(\log^* n)$ lower bound for randomized algorithms [90, 78], our solution is an interesting step towards the optimal algorithm.

It is worth noting that the algorithm presented in this chapter achieves the fastest known constant-factor approximation for the Dominating Set Problem in the class of Unit Disk graphs, since any MIS is at most five times larger than an optimal dominating set in this class of graphs [83].

The main technical novelty introduced by this chapter is a new randomized algorithm to find a $2t$-ruling set with low induced maximum degree in $O(t)$ rounds. Even in general graphs, our approach finds a $O(\log \log \Delta)$-ruling set with induced degree $O(\log^5 n)$ in $O(\log \log \Delta)$ rounds (with high probability), which might be of independent interest. In each step of this algorithm, a subset of the nodes is selected. This subset consists of three different sets, which are designed such that their combination guarantees the desired ruling-property and a rapidly decreasing maximum degree. Two of these sets are chosen deterministically, and the third set is chosen randomly using an approach resembling Luby’s algorithm, but with a different choice of probabilities.

The results of this chapter have been established in a joint work with Beat Gfeller [50].

### 4.4 Useful Facts

This section summarizes two well-established results of probability theory that are used in this part.

The Chernoff bound is used to express the tail distribution of the sum of independent $0-1$-variables, which are called Poisson trials.

**Fact 4.1** (Chernoff Bound, [86]). Let $X_1, \ldots, X_n$ be independent Poisson trials and let $X = \sum X_i$. Then the following bound, called
Chernoff bound, holds:

\[ \Pr [X \geq (1 + \delta)E[X]] \leq e^{-E[X]\delta^2/3}, \quad \text{for } 0 < \delta \leq 1. \]

**Fact 4.2** (Union bound, [86]). Let \( E_1, \ldots, E_n \) be some non-necessarily independent events. Then

\[ \Pr \left[ \bigcup_{i=1}^{n} E_i \right] \leq \sum_{i=1}^{n} \Pr [E_i]. \]

### 4.5 Distributed MIS Algorithm

In this section, we present the Algorithm \textsc{MaxIndepSet}, which computes with very high probability a MIS in Growth-Bounded graphs in \( \mathcal{O} (\log \log n \log^2 n) \) rounds. First, we outline the structure of the algorithm.

In the first phase of Algorithm \textsc{MaxIndepSet}, a subset \( T \subset V \) of nodes of \( G \) is selected, such that (1) \( T \) is a \( \mathcal{O} (\log \log n) \)-ruling set of \( G \), and (2) the subgraph induced by \( T \) has a maximum degree of \( \mathcal{O} (\log^5 n) \). This set is obtained by repeatedly applying Algorithm \textsc{RandStep} (see below), using \( V \) as the initial set. Then in a second phase, the set \( T \) is further thinned out, such that the remaining set of nodes \( T' \) is an independent \( \mathcal{O} (\log \log n) \)-ruling set of \( G \). As in the first phase, this is achieved by repeatedly selecting particular subsets of the previous set (starting with \( T \)). Finally, in a third phase, this sparse independent set \( T' \) is extended into a Maximal Independent Set.

The main novelty of our approach lies in the first phase of the algorithm, where the logarithm of the maximum degree induced by the remaining nodes decreases geometrically in each step, using randomization. For the second and the third phase, we use two deterministic algorithms from [63].

Note that the transition from the first phase to the second is triggered by a threshold of \( \mathcal{O} (\log^5 n) \) for the maximum degree of the remaining graph. Not knowing \( n \), the nodes cannot resolve locally when the maximum degree has been reduced enough. This is why we interchange the executions of the algorithms of the first two phases. We will argue that this does not harm the effectiveness of either phase.
4.5. Distributed MIS Algorithm

The next sections describe the building blocks of our algorithm in more detail. In particular, we will show that each phase requires at most $O(\log \log n \log^* n)$ rounds (Phase 1 uses $O(\log \log \Delta)$ rounds with high probability), leading to the following result.

**Theorem 4.3.** There exists a randomized algorithm that computes a MIS for any Growth-Bounded graph $G$ within $O(\log \log n \log^* n)$ rounds of synchronous distributed computation with probability at least $1 - O(1/n^k)$ (for any $k > 3$) in the message-passing model. All messages are of size $O(\log n)$. The algorithm complies with the CONGEST model.

**4.5.1 Phase 1: A $O(\log \log n)$-ruling set with small induced degree**

This phase consists of repeated executions of the Algorithm RAND-STEP. Let $T_1$ be the set of nodes returned by the first execution of RAND-STEP. This set induces a graph $G[T_1]$, on which RAND-STEP is applied again, yielding $T_2$ and thus a new graph $G[T_2]$. Continuing in this way, we obtain a sequence of sets $T_i$ which are increasingly sparse, while at the same time remaining $O(\log \log n)$-ruling.

The central idea behind Algorithm RAND-STEP is to choose a random subset of all nodes, such that the maximum degree of the induced subgraph decreases rapidly, i.e., from $\Delta$ to $\Delta^c$ with $c < 1$, but assuring that the remaining set of nodes is a 2-ruling set of the previous graph. The key question is how to choose the probabilities such that nodes with large degree will have small degree afterwards: if all nodes have the same degree $d$, and each node decides independently with probability $p = 1/d^{1/4}$ whether to stay in the set, then the expected degree is $d^{3/4}$. This achieves that the maximum degree (or, correspondingly, the size) decreases from $\Delta$ to $\Delta^c$ with high probability, and furthermore the probability that a node and all of its neighbors leave the set is rather low. Clearly, when nodes have different degrees, they will use different probabilities, so the above reasoning does not apply immediately. As we will see however, if the degrees of neighboring nodes do not differ too much, then one can still obtain a similarly fast decreasing maximum degree (with high probability).

---

1Recall that the size is the degree of a node plus one.
Algorithm 1: Randomized sparsify the graph

1. **Algorithm**: RANDSTEP
   
   **Input**: A Growth-Bounded Graph $G = (V, E)$.
   
   **Output**: A 2-ruling set $T \subset V$ of $G$.

2. $R := \emptyset$

3. $S := \{u \in V \mid \bar{d}_u^2 < \bar{d}_u \text{ for some } v \in N(u)\}$

4. $U := N(S) \setminus S$

5. $B := V \setminus (U \cup S)$ ($B$ is the set of black nodes)

6. Each $u \in B$ independently joins $R$ with probability $p = \frac{1}{\bar{d}_u^{1/4}}$. ($R$ is the set of red nodes)

7. $\mathcal{G} := B \setminus (N(R) \cup N(U))$ (set of green nodes)

8. return $T := S \cup R \cup \mathcal{G}$

Algorithm RANDSTEP works as follows (see Figure 4.1): first, nodes which have a neighbor with size much larger than their own have a relatively low size, so they can simply stay in the graph for the next round (this is the set $S$ in the algorithm). Additionally, their neighbors (set $U$ in the algorithm) leave the graph because they have at least a neighbor that stays. All other nodes then have only neighbors with similar sizes. We refer to these nodes as black nodes and denote them by the set $B$. A black node $u$ becomes red (and stays in the graph) independently with probability $p = 1/\bar{d}_u^{1/4}$. The respective set is called $R$ in Algorithm RANDSTEP. Finally, with low probability, there may be black nodes for which no node within distance two stays in the graph. In order to guarantee the 2-ruling property (in contrast to merely knowing a high probability bound for it), we add these nodes to the set of green nodes $\mathcal{G}$ which also stay in the graph.

For convenience, the algorithm is formulated in a global fashion, but a distributed version, where each node can execute these steps by communicating only with its direct neighbors in the CONGEST model, is immediate.

---

2For ensuring the 2-ruling property, it would suffice to define $\mathcal{G} := B \setminus (N_2(R) \cup N(U))$, but replacing $N_2(R)$ by $N(R)$ simplifies the analysis.
4.5. Distributed MIS Algorithm

Figure 4.1: Illustration of Algorithm RANDSTEP. The sets $S$, $R$ and $G$, which are drawn as filled disks, remain in the graph. Their union $S \cup R \cup G$ is a 2-ruling set of $G$.

Analysis

In the following, we prove that with high probability, iterating Algorithm RANDSTEP for $O(\log \log n)$ times yields a $O(\log \log n)$-ruling set whose induced subgraph has a maximum degree $O(\log^5 n)$.

First, we examine the ruling property of Algorithm RANDSTEP when applied repeatedly.

Lemma 4.4. After iterating Algorithm RANDSTEP $t$ times, the obtained set $T_t$ is a $2t$-ruling set.

Proof. Consider the iteration from $T_i$ to $T_{i+1}$. Note that after every execution $i$ of Algorithm RANDSTEP, the sets $S$, $U$ and $B$ form a partition of $T_i$. Any node in $S$ stays in the set. Any node in $U$ is dominated by some node in $S$. $R \cup G$ is by construction a dominating set of $B \setminus N(U)$. Thus, the only nodes in $T_i$ that may not be dominated by $S \cup R \cup G$ are those in $N(U) \setminus U$. These nodes have by definition a 2-hop neighbor in $S$. Thus, after each iteration, $S \cup R \cup G$ is a 2-ruling set of $T_i$. The claim now follows by induction over $i$ as, initially, $T_0 = V$ is a 0-ruling set.

For the further analysis, we consider only one execution of Algorithm RANDSTEP on the graph $G = (V, E)$. During the exposition and the analysis of the algorithms, the sizes and neighbors of a node are to be understood with respect to the current graph. In the analysis
we choose the constants with the purpose to get simple terms. A more careful selection might lead to smaller constant factors in the running time.

The first lemma states that the sizes of neighboring nodes in $B$ do not differ too much.

**Lemma 4.5.** For any $u \in B$, and for each $v \in N(u)$, we have

$$(\overline{d}_v)^{1/2} \leq \overline{d}_u \leq (\overline{d}_v)^2.$$ 

**Proof.** By the assumption of the lemma, $u \notin S$. Thus we have $\overline{d}_u^2 \geq \max_{w \in N(u)} \overline{d}_w \geq \overline{d}_v$, which yields the left inequality. From $u \in B$ we have $u \notin U$ and hence $v \notin S$. So $\overline{d}_u^2 \geq \max_{w \in N(u)} \overline{d}_w \geq \overline{d}_u$ and the right inequality follows.

**Lemma 4.6.** The probability that a node $u \in B$ with $\overline{d} := \overline{d}_u \geq k^2 \ln^2 n$ becomes green is at most $\frac{1}{nk}$, for $k > 1$.

**Proof.** Let $u \in B$ be a node with the property that $N(u)$ consists of only black nodes. Only such nodes may potentially become green. If, after the algorithm execution, one of the nodes in $N(u)$ becomes red then $u$ does not become green.

As the $\overline{d} - 1$ neighbors are all black, their sizes are all at most $\overline{d}^2$ by Lemma 4.5. Thus, each of them becomes red with probability at least $\overline{d}^{-1/2}$. The probability that neither $u$ nor any of its $\overline{d} - 1$ neighbors become red is therefore at most

$$\left(1 - \overline{d}^{-1/2}\right)^{\overline{d}} \leq e^{-\overline{d}^{1/2}},$$

using the basic inequality $1 - x \leq e^{-x}$. So for $d \geq k^2 \ln^2 n$, we have $\mathbb{P}[u \text{ becomes green}] \leq \frac{1}{nk}$.

As we show in the following lemmas, as long as the maximum degree $\Delta$ of $G$ is at least $\Omega(\log^5 n)$, one iteration of RANDESTEP almost surely decreases the maximum size $\overline{\Delta}$ of the graph to $2\overline{\Delta}^{7/8}$ or below.

**Lemma 4.7.** For any $k > 1$, the probability that a black node $u \in B$ with $\overline{d} \geq 9k^2 \ln^2 n$ has more than $2\overline{d}^{7/8}$ red neighbors (including itself) is at most $\frac{1}{nk}$. 
4.5. Distributed MIS Algorithm

Proof. Let $u$ be a black node with size $\bar{d}$. Further, let $Y_1, \ldots, Y_{\bar{d}}$ be $\bar{d}$ Poisson trials, one assigned to each neighbor of $u$ and one assigned to $u$ itself. For $1 \leq i \leq \bar{d}$, $Y_i$ is 1 if the corresponding node joins $\mathcal{R}$, and 0 otherwise. Each of these Poisson trials may have a different success probability. Recall that each black node $v$ becomes red with probability $\frac{1}{\bar{d}^{1/2}}$, independently of the choices of its neighbors, hence $Y_1, \ldots, Y_{\bar{d}}$ are independent. Further, $X = Y_1 + \ldots + Y_{\bar{d}}$ is the number of red neighbors of $u$, plus one if $u$ is red, i.e., $X := |N(u) \cap \mathcal{R}|$. By Lemma 4.5, however, all neighbors of $u$ (and $u$ itself) have size at least $\frac{d^{1/2}}{2}$, so each neighbor of $u$ (and $u$ too) joins $\mathcal{R}$ with probability at most $\frac{1}{\bar{d}^{1/2}}$. Clearly, $\mathbb{E}[X] \leq \bar{d} \cdot \frac{1}{\bar{d}^{1/2}} = \bar{d}^{7/8}$. Using the Chernoff bound (Fact 4.1) with $\delta = 1$ and $\bar{d} \geq 9k^2 \ln^2 n$, we obtain

$$\mathbb{P}\left[X \geq 2\bar{d}^{7/8}\right] \leq e^{-\frac{1}{3} \bar{d}^{7/8}} \leq e^{-(k \ln n)} \leq \frac{1}{n^k}.\]$$

Lemma 4.8. The probability that a node $u \in B$ with $\bar{d} := \bar{d}_u \geq 9k^4 \ln^4 n$ has more than $2\bar{d}^{7/8}$ neighbors (including itself) in $\mathcal{R} \cup \mathcal{G}$ is at most $\frac{2}{n^{k-1}}$, $k > 1$.

Proof. Let $A$ be the event that $u$ has more than $2\bar{d}^{7/8}$ red neighbors, and let $B$ be the event that $u$ has any green neighbor. By Lemma 4.5, all neighbors of $u$ have at least size $3k^2 \ln^2 n$, so by Lemma 4.6 every neighbor of $u$ becomes green with probability at most $\frac{1}{n^k}$. From the union bound, it follows that the probability of $u$ having any green neighbor is at most $\frac{1}{n^{k-1}}$, because $u$ can have at most $n$ neighbors.

Using this fact and Lemma 4.7 for the second inequality, we have

$$\mathbb{P}[A \cup B] \leq \mathbb{P}[A] + \mathbb{P}[B] \leq \frac{1}{n^k} + \frac{1}{n^{k-1}} \leq \frac{2}{n^{k-1}}.\]$$

Lemma 4.9. Let $\Delta$ be the maximum degree of the subgraph induced by $T_i$ after $i \geq 0$ iterations of Algorithm RANDSTEP. Assume that $\Delta \geq 6k^5 \ln^5 n$, $k > 2$, hold. After one more iteration, the maximum size of the subgraph induced by $T_{i+1}$ is at most $2\Delta^{7/8}$ with probability at least $1 - \frac{2}{n^{k-2}}$.

$^3$Recall that $\bar{\Delta} = \Delta + 1$. 


Chapter 4. The MIS Problem

Proof. Consider any node $u \in T_i$ with $\bar{d}_u \geq 2\Delta^{7/8}$. This node will only be in $T_{i+1}$ after the next iteration if it is in $S \cup R \cup G$. However, $u$ cannot be in $S$ (because otherwise there would exist a node $v$ with $\bar{d}_v > \bar{d}_u \geq 4\Delta^{14/8} \geq \Delta$, a contradiction.). So $u$ is either in $R$ or in $G$. Moreover, by construction any node in $B$ has no neighbors in $S$, so its size is the number of neighbors in $R \cup G$. As $\bar{d}_u \geq 2\Delta^{7/8} \geq 2(6k^5 \ln^5 n)^{7/8} \geq 9k^4 \ln^4 n$, Lemma 4.8 can be applied, so the size $\bar{d}'_u$ of $u$ after the iteration satisfies $\bar{d}'_u \leq 2\Delta^{7/8}$ with probability at least $1 - \frac{2}{n^k-1}$. Hence the probability that any of the nodes with size $\bar{d} \geq 6k^5 \ln^5 n$ has a size $\geq 2\Delta^{7/8}$ is at most $\frac{2}{n^k-2}$, again using the union bound.

The above bounds together yield a high-probability bound for the number of RANDSTEP iterations required to reduce the maximum degree to polylogarithmic size. Note that this result holds for arbitrary graphs, not only for Growth-Bounded graphs.

**Theorem 4.10.** For any constant $k > 3$, and some suitable constant $c = c(k)$, when RANDSTEP is run on any graph $G = (V, E)$ with maximum degree $\Delta$, then the maximum degree in the subgraph of $G$ induced by $T_i$ is at most $6k^5 \ln^5 n$ after no more than $c \ln \ln \Delta$ iterations with probability at least $1 - 2c/n^{k-3}$.

Proof. We call an iteration (of RANDSTEP) successful if the current maximum size of any node in the subgraph induced by $T_i$ is reduced from $\Delta$ to $2\Delta^{7/8}$. Thus, after $m$ successful iterations, the degree is at most

$$2^{1+7/8+(7/8)^2+\ldots+(7/8)^m-1} \cdot \Delta^{(7/8)^m} = 2^8 2^{1-(7/8)^m} \Delta^{(7/8)^m} \leq 512 \Delta^{(7/8)^m}.$$  

In order to reduce the degree to below $6k^5 \ln^5 n$, $m$ must satisfy

$$512 \Delta^{(7/8)^m} \leq 6k^5 \ln^5 n,$$

which holds for $m \geq c \ln \ln \Delta$ for some constant $c = c(k)$.

By Lemma 4.9, each iteration is successful with probability at least $1 - \frac{2}{n^k-2}$ as long as the degree is at least $6k^5 \ln^5 n$. Since we require at most $c \ln \ln \Delta$ successes and each iteration succeeds with
probability at least \(1 - \frac{2}{n^{k-2}}\), as long as the maximum degree is big enough (Lemma 4.9), the probability that all rounds are successful is high. Formally, let \(A_i\) be the event that round \(i\) is not successful. There are at most \(c \ln \ln \Delta\) iterations that could potentially fail. Hence, the probability that at least one of these iterations fails is at most

\[
\Pr \left[ \bigcup_i A_i \right] \leq \sum_i \Pr [A_i] \leq \frac{2c \ln \ln \Delta}{n^{k-2}} \leq \frac{2c}{n^{k-3}}.
\]

By taking into account that \(\Delta\) might reach the threshold \(6k^5 \ln^5 n\) earlier in the algorithm execution, the probability that it ends within \(c \ln \ln \Delta\) iterations only grows. Thus, after \(c \ln \ln \Delta\) iterations, \(\Delta \leq 6k^5 \ln^5 n\) holds and hence the maximum degree of the graph is at most \(6k^5 \ln^5 n\) with high probability.

\[\square\]

### 4.5.2 Phase 2: A Deterministic Algorithm for a \(O(\log \log n)\)-Ruling Set

As the maximum degree of the remaining graph becomes smaller, RandStep becomes less effective in reducing it. This is because a node \(u\) in \(B\) with small \(\overline{d}_u\) will only be removed from the graph with a small probability (and its neighbors, too). On the other hand, the deterministic algorithm Sparsify from \([63]\) (called Algorithm 1 in that paper), which computes a MIS in Growth-Bounded graphs in \(O (\log \Delta \log^* n)\) rounds, guarantees to halve the maximum degree of a Growth-Bounded graph within a constant number of steps. Thus, the latter algorithm is slower than RandStep while \(\Delta\) is large, but is faster than RandStep once \(\Delta\) is only polylogarithmic.

The algorithm Sparsify is roughly described as follows: in each iteration, called SparsifyStep here, a subgraph \(\overline{G}\) of \(G\) is selected such that every node in \(\overline{G}\) has degree 1 or 2, and each node of \(G\) has a neighbor in \(\overline{G}\). Due to the bounded degree of \(\overline{G}\), a MIS of \(\overline{G}\) can be computed in time \(O (\log^* n)\), using the algorithm from \([26]\). Only nodes in this MIS stay in the graph for the next iteration. This subset is a 2-ruling set of the nodes in \(G\).

It can be seen from the proof of Lemma 5 in \([63]\) that this algorithm halves the degree of the graph after a constant number of iterations. Furthermore, inspection of this proof also shows that adding interleaved iterations of RandStep does not harm the analysis.
**Algorithm 2**: Deterministic sparsifier of the graph

1. **Algorithm**: SPARSIFYSTEP
   - **Input**: A Growth-Bounded graph $G = (V, E)$.
   - **Output**: A 2-ruling set $T \subset V$ of $G$
2. Send an invitation to one arbitrarily chosen neighbor.
3. Accept one invitation, if any was received.
4. $E' := \{(u, v) \in E \mid u$ accepts the invitation of $v$ or vice versa$\}$
5. Compute a MIS $T$ on the edge-induced subgraph $\overline{G} := G(V, E')$.
6. return $T$

**Lemma 4.11** ([63]). *For a Growth-Bounded graph $G$, SPARSIFY reduces the maximum degree of the graph from $\Delta$ to $\Delta/2$ in $h \in O(1)$ iterations. Each iteration takes $O(\log^* n)$ rounds.*

### 4.5.3 Combining Phase 1 and Phase 2: The Independent Ruling-Set Algorithm

The key observation for obtaining Algorithm RULINGSET which is fast in both of these phases is that the two algorithms can be interleaved: after executing one call of RANDSTEP, we execute one call of SPARSIFYSTEP. The situation is depicted in Figure 4.2 and is formalized by Algorithm[3]

This is possible because for both algorithms, one iteration takes a Growth-Bounded graph as input, and returns as output a subset of its nodes which is a 2-ruling set of the input graph. Hence, after $t$ iterations of the combined algorithm, the remaining set of nodes is a $4t$-ruling set of the original graph. Furthermore, Theorem 4.10 still holds for the combined algorithm, as the inserted iterations of SPARSIFYSTEP never increase the degree of any node. Once the degree of the remaining subgraph is at most $\Delta \leq 6k^5 \ln^5 n$, by Lemma 4.11 the deterministic steps guarantee that the degree is rapidly decreased to zero (i.e., all remaining nodes form an independent set) within $O(\log(6k^5 \ln^5 n)) = O(\log \log n)$ iterations of the combined algorithm, as also the iterations of the RANDSTEP algorithm never increase any node degree.
4.5. Distributed MIS Algorithm

Figure 4.2: Interleaving Phase 1 and Phase 2 yields an efficient algorithm that runs without the knowledge of the maximal degree of the network.

Algorithm 3: Ruling set step

1 Algorithm: RULINGSET

   Input: A Growth-Bounded graph $G = (V, E)$
   Output: An independent ruling-set $I$ of $G$

2 $I := V$

3 while $I$ is not independent do
4     Set $I$ to be the output of RANDSTEP on $I$
5     if $I$ is independent then exit while-loop
6     Set $I$ to be the output of SPARSIFYSTEP on $I$

7 end

8 return $I$

Of course, interleaving one single step of the deterministic algorithm (and not the required $h$ steps to guarantee the bisection of the degree) deteriorates the constant factor of the running time, but allows on the other hand the execution of the algorithm without knowledge of $h$. This discussion is summarized in the following theorem.

Theorem 4.12. For any $k > 1$ and any Growth-Bounded graph $G$, Algorithm RULINGSET computes an $O(\log \log n)$-ruling independent set of $G$ in $O(\log \log n \log^* n)$ rounds of synchronous distributed computation, with probability at least $1 - O(1/n^k)$. 
4.5.4 Phase 3: Obtaining the Maximal Independent Set

At this point the $O(\log \log n)$-ruling independent set computed by Algorithm RULINGSET is condensed to yield a Maximal Independent Set. We invoke the condensing algorithm from [63] (Algorithm 2), which extends any $t$-ruling independent set to a MIS in $O(t \cdot \log^* n)$ rounds in Growth-Bounded graphs. Thus, in our case we get a MIS in $O(\log \log n \log^* n)$ rounds. For the sake of completeness we describe shortly how this algorithm works. It is, in turn, split into two phases: the first phase condenses the existing ruling set until the resulting set is 3-ruling. This is achieved by letting every active node (i.e., a node in the current independent set) compute an independent set in its 4-neighborhood under the constraint that every node in the 3-neighborhood is dominated by an active node. The resulting new set might be no longer independent, but the growth-bounded property of the graph permits to efficiently compute an independent set out of this denser set of active nodes. This phase is repeated until the final set is 3-ruling and takes $O(\log \log n \log^* n)$ rounds.

The final phase, i.e., the task of producing the desired MIS from the 3-ruling set, is achieved by defining a cluster graph that enables the local completion of the independent set achieved so far. A coloring of the cluster graph defines an order of precedence on the active nodes that, in turn, ensures that the final set is independent. Again, the efficiency of the algorithm relies on the growth-bounded property of the original graph, which leads to the final MIS in $O(\log^* n)$ rounds.

Combined, Theorem 2 and Lemma 8 from [63] imply the following theorem.

**Theorem 4.13.** For a Growth-Bounded graph $G$, Phase 3 transforms a $O(\log \log n)$-ruling independent set of $G$ into a MIS of $G$ in $O(\log \log n \log^* n)$ rounds of synchronous computation.

4.6 Removing Global Coordination

This section deals with issues that arise when the algorithms which we formulated in a global way are to be transformed into local distributed algorithms. We assume these techniques to be well-known,
but we include some explanations to the example of our MIS algorithm for the sake of completeness.

In the descriptions of our algorithms so far, we have used global criteria at some stages. For example, in Algorithm \texttt{3 (RULINGSET)} we sequentially call two different distributed algorithms. The call to \texttt{RANDSTEP} is easy to implement locally, because its execution requires the same (constant) number of rounds at each node. The call to \texttt{SPARSIFYSTEP}, however, is more subtle: as a part of its execution, a MIS is computed with the algorithm from \cite{26}, whose global termination time is not available to the nodes. The third global criteria used in Algorithm \texttt{RULINGSET} is the termination of the while loop (which determines when Phase 3 of our algorithm is started). If this condition had to be checked globally as stated, then the running time of the algorithm would be at least linear in the network diameter. Fortunately, this is not required: one can replace the global termination condition by a local termination condition for each node. Each node \(u\) terminates Algorithm \texttt{RULINGSET} as soon as it has either joined the independent set \(I\) or has been removed from \(G\) without joining \(I\) (in both cases, it is determined whether node \(u\) will be part of the independent set when Algorithm \texttt{RULINGSET} terminates globally). Thus, some nodes may terminate Algorithm \texttt{RULINGSET} earlier than others. If fact, all the global termination conditions we have just mentioned have two crucial properties. Firstly, there is a local termination condition for each node, such that the global termination condition is true if and only if all local termination conditions are true. Further, no local termination condition can become false again once it has previously become true.

We briefly describe how an algorithm which is split into several phases, where the termination condition of each phase satisfies the aforementioned properties, can be implemented without global coordination. The idea is to delay the execution of nodes which are ahead of their neighbors, thus locally maintaining synchrony. More precisely, a node \(u\) which has completed phase \(i\) starts phase \(i + 1\) only when all its neighbors have also completed phase \(i\). If some neighbor of node \(u\) has not yet completed phase \(i\), node \(u\) enters a “waiting” state and sends a “pause \(i + 1\)” message to all those neighbors which have already completed phase \(i\). Upon receiving a “pause \(i + 1\)” message, a node forwards it to all neighbors which have not yet completed phase \(i + 1\). Then, if it has already completed phase \(i\), it
Chapter 4. The MIS Problem

enters a “waiting” state itself. Otherwise, it continues its computation and enters the “waiting” state only after completing phase $i$. Whenever a node $u$ has completed phase $i$, it informs all its neighbors. This may allow some nodes (those which now have only neighbors that have completed phase $i$) to start with phase $i + 1$, thus they in turn inform all neighbors to which they have sent a “pause $i + 1$” message with a “continue $i + 1$” message. A node in waiting state in phase $i + 1$ continues its execution as soon as it has received a “continue $i + 1$” message from all neighbors which have previously sent a “pause $i + 1$” message, and in turn sends a “continue $i + 1$” message to all neighbors waiting for it.

It should be clear that by this procedure, the execution is equivalent to one in which the beginning of a new phase is globally coordinated. Note that the mechanism just described is related to the synchronizers from [8], viewing the different phases of the algorithm as rounds in the synchronized model which must be ensured to be in synchrony in the asynchronous setting.

Next, we argue that the asymptotic running time of the algorithm is not increased by replacing the global termination conditions by local conditions. To that end, consider any critical node, i.e., a node whose computation has advanced the least. Such a node does not need to wait for any neighbor. If it is in the waiting state when it becomes critical, then it will receive a “continue” message in the next round. Thus, any critical node will continue its execution at most one round after becoming critical. Hence it follows that the worst case time for completing each phase is at most doubled, and hence the asymptotic running time of the algorithm is not increased by replacing the global termination conditions by local conditions.

In addition, we remark that using the $\alpha$-synchronizer of [8], our algorithm also terminates in $O(\log \log n \log^* n)$ time in an asynchronous setting, at the cost of an increased message complexity.

4.7 Open Problems

The optimal deterministic algorithm for the MIS problem in Growth-Bounded graphs has been proposed very recently [113]. The most intriguing open problem is to determine the optimal algorithm in the case of general graphs. Barenboim and Elkin [11] performed a first
step in this direction by describing a sublogarithmic algorithm for a large class of graphs. However, as for today there is nothing better than the randomized $O(\log n)$-algorithm by Luby [81].
Chapter 5

The Minimum Connected Dominating Set Problem

A Distributed Approximation Scheme

A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable. – Leslie Lamport (born February 7, 1941), American computer scientist and initial developer of Latex.

5.1 Introduction and Motivation

In contrast to wired networks, where usually a dedicated backbone infrastructure with high-throughput capabilities is available for long-distance routing, ad hoc networks do not have any a priori means to manage the routing and scheduling of messages. In the absence of an organized routing scheme, simple flooding (i.e., the first time a message is received from any neighbor, it is forwarded to all other
neighbors) could be used to transmit messages. However, this is very wasteful in terms of energy and causes interference problems if many nodes transmit a message at the same time. To organize routing more cleverly, a virtual backbone can be computed, i.e., a subset of the nodes which participate in multi-hop routing. In this fashion, even when messages are still forwarded by flooding inside the virtual backbone, the energy savings are significant.

When modelling the network as a graph, the most widely used concept for defining a backbone is the Connected Dominating Set (CDS). Recall that a CDS is a subset \( S \) of the nodes of a graph \( G \) that induces a connected subgraph of \( G \), such that every node that is not in \( S \) has a neighbor in \( S \).

Usually, the energy savings are higher if the number of nodes in the CDS is small. However, computing a Minimum CDS is NP-hard even on Unit Disk graphs [25], and would require global information. Considering the highly dynamic nature of ad hoc networks, it is important that the CDS can be computed locally within a short time; a linear running time in the diameter of the network (as required to obtain global information) is clearly inappropriate. Moreover, a minimum CDS lacks some desirable properties which we demand from an efficient backbone. For instance, routing a message from a node \( v \) to a node \( w \) should not need many more intermediate hops than a shortest path in the original network. The maximum ratio over all vertex pairs between these two hop-distances is called the stretch factor. The stretch factor of a MCDS can be as bad as linear in the number of nodes (for instance, see Figure 5.1), and one would like to prevent this effect. Another desirable property is that nodes have constant degree in the CDS-induced graph, which helps to address interference issues. For these reasons, a natural trade-off is to find a CDS with only near-optimal size but fulfilling the aforementioned properties.

## 5.2 Related Work

The concept of a virtual backbone (in analogy to backbones in wired networks) was introduced in [29]. Since then, the construction of small Connected Dominating Sets in Unit Disk graphs has been intensively studied. A recent overview can be found in [15]. For the centralized setting with given coordinates of the nodes (which are
embedded in the Euclidean plane), a *polynomial-time approximation scheme* (PTAS\textsuperscript{\textdagger}) was proposed in [23]. Approximation schemes for related problems like the Minimum Dominating Set Problem in Unit Disk graphs were given in [57]. The approach of [28] (as well as our approach) yields a PTAS that does not require coordinate information about the nodes.

However, many of the first distributed algorithms either did not guarantee a good approximation ratio in the worst case, or had a linear running time (see [15]). The first approach achieving a constant approximation ratio in polylogarithmic time was [100]. Alzoubi et al. [4] were the first to provide an algorithm for computing a CDS with low stretch and low degree, which was coined *well-connected* CDS in [100]. Recently, Czygrinow et al. [28] presented a distributed algorithm with running time $O\left(\frac{1}{\varepsilon^6} \cdot \log(1/\varepsilon) \cdot \log^3 n\right)$ achieving the approximation ratio $1 + \varepsilon$ for any $\varepsilon > 0$, but without proving any stretch or degree bounds.

In general graphs, the best known distributed algorithm for the MCDS problem has a polylogarithmic running time and achieves an approximation ratio of $O\left(\log \Delta\right)$, where $\Delta$ is the maximum degree of the network [33]. This approximation ratio is asymptotically optimal unless all problems in NP can be solved by deterministic algorithms.

\textsuperscript{\textdagger}A PTAS is an approximation algorithm that approximate the optimal solution of an optimization problem within a factor $\varepsilon$, for every chosen $\varepsilon > 0$. A PTAS runs in a time polynomially bounded by $n$ only.
with running time \( n^{O(\log \log n)} \) \[39\].

5.3 Summary of Results

In this chapter we present a distributed approximation scheme for the problem of finding a Minimum Connected Dominating Set in the class of p-Growth-Bounded graphs. An important feature of our algorithm is that the only information required by the nodes is the set of their direct neighbors. Distances between them or even coordinate information are not required.

The algorithm computes a well-connected \((1 + \varepsilon)\)-approximation of a Minimum Connected Dominating Set, for any \( \varepsilon > 0 \). This takes \( O(T_\text{MIS} + 1/\varepsilon^{O(1)} \cdot \log^* n) \) rounds of synchronous computation in the \( \text{LOCAL} \) model, where \( T_\text{MIS} \) is the number of rounds needed to compute a Maximal Independent Set. Before the publication of \[113\], the fastest deterministic distributed algorithm for computing a \( \text{MIS} \) in p-Growth-Bounded graphs was due to Kuhn et al. \[63\] and runs in \( O(\log \Delta \cdot \log^* n) \) time. As we have proved in the previous chapter, a \( \text{MIS} \) can be computed in \( O(\log \log n \cdot \log^* n) \) time in Growth-Bounded graphs using randomization. With the algorithm of Schneider and Wattenhofer \[113\], the running time drops to \( O\left(1/\varepsilon^{O(1)} \cdot \log^* n\right) \). So, we improve on the running time for computing a \((1+\varepsilon)\)-approximate CDS, while adding the guarantee that the computed CDS has constant stretch, constant degree, and therefore a linear number of edges.

The algorithm we propose builds substantially on the approach of Nieberg et al. \[92\] for computing a \((1 + \varepsilon)\)-approximate Minimum Dominating Set (for p-Growth-Bounded graphs). In a nutshell, their solution partitions the graph into clusters of appropriate radius, computes an optimal DS on each of these, and takes the union of these sets to yield a DS for the graph. Employing the same idea, we cluster the graph and compute an optimal CDS for each cluster. However, we let the clusters overlap such that the union of the small CDS solutions forms a connected DS of the graph. We prove the approximation ratio of the CDS by adapting the proof from \[92\], which requires an additional non-trivial step addressed in Lemma 5.6 and Lemma 5.7. In the proof of Lemma 5.7 ideas related to \[28\] are used. In addition, we prove the (well-)connectedness of the computed set, which has no
In order to turn our centralized approximation scheme for CDS into a distributed algorithm, we follow the lines of [64].

Intriguingly, such a relatively simple modification of known techniques yields a distributed approximation scheme which runs substantially faster than the previously known solution [28], and additionally guarantees well-connectedness.

The results of this chapter have been established in a joint work with Beat Gfeller [49][51].

5.4 Useful Facts

To end this section, we introduce two properties of optimal connected dominating sets in p-Growth-Bounded graphs, which we use in our approach.

**Lemma 5.1.** For any p-Growth-Bounded graph $G = (V, E)$ with bounding function $f$, there is a polynomial $p$ with $p(r) \leq 3 \cdot f(r)$, $r > 0$, such that for any $v \in V$, $|C(N_r(v))| \leq p(r)$ holds for all $r > 0$.

**Proof.** Let $f(r)$ be the polynomial bounding function of the p-Growth-Bounded graph $G$. Consider a Maximal Independent Set $I$ of $N_r(v)$ (for a fixed $r \geq 0$) and set $Q := I$. Let $k = |I|$ be the number of components of $G[I]$. We proceed by induction over $k$. If $k = 1$, $G[Q]$ is connected and the claim follows. Otherwise, since $I$ is also a dominating set of $N_r(v)$, we can find two connected components $A, B \subset Q$ such that $d(A, B) \leq 3$. By adding to $Q$ the vertices of a shortest path between them, we decrease the number of components by at least one and we increase $|Q|$ by at most two. We proceed inductively until $Q$ induces a connected graph. Since $k = |I|$, we get $|C(N_r(v))| \leq |Q| \leq |I| + 2|I| \leq 3f(r)$. As this holds for every $r > 0$, the claim is proved. \(\square\)

**Lemma 5.2.** Let $G = (V, E)$ be a p-Growth-Bounded graph with bounding function $f$, and choose a $S \subset V$ that induces a connected subgraph. Then for any MIS $M$ of $S$, it holds: $|M| \leq f(1) \cdot |C(S)|$.

---

\(^2\text{Recall that } C(N_r(v)) \text{ represents an optimal CDS of } N_r(v).\)
Proof. As \(C(S)\) dominates \(S\), each node in \(M\) must have a neighbor in \(C(S)\) (or be in \(C(S)\) itself). But by definition, at most \(f(1)\) nodes in \(M\) can have the same neighbor in \(C(S)\), so the claim follows. \(\square\)

### 5.5 Finding a Small Connected Dominating Set

In the following, we describe a (sequential) procedure to construct for each \(\varepsilon > 0\) a Connected Dominating Set of size at most \((1 + \varepsilon)\) times the minimum. This procedure can be executed efficiently in a centralized way, and thus leads to a PTAS. Moreover, in Section 5.6 we show that the same procedure can be implemented efficiently in a distributed way, using the same technique as in [64].

The CDS is constructed by computing optimal connected dominating sets for small parts of the graph, and taking the union of these CDSs. We will construct the small CDSs such that their union leads to a connected set, as required. Each small CDS is an optimal solution of a small cluster specified as follows: choose any node \(v \in V\), and consider the \(r\)-neighborhood of \(v\) for increasing values of \(r = 0, 1, 2, \ldots\) until we find a large enough \(r^*\) such that

\[
\left| \max_{S}(N(\Gamma_{r^*}(v)) \setminus \Gamma_{r^*}(v)) \right| \leq \varepsilon \cdot \left| \max_{S}(\Gamma_{r^*}(v)) \right| \quad (5.1)
\]

holds. We call this operation an expansion of \(v\). As we show in Lemma 5.4 below, \(r^*\) is bounded by a function in \(O\left(\frac{1}{\varepsilon} \cdot \log(\frac{1}{\varepsilon})\right)\) depending solely on \(\varepsilon\) for any class of \(p\)-Growth-Bounded graphs.

Our algorithm for finding a CDS for \(G\) proceeds as follows: starting with an empty set \(D\), it chooses any node \(v_1 \in V\) of \(G\), finds a corresponding \(r_1^*\) such that Inequality 5.1 holds, and adds the nodes of \(C(\Gamma_{r_1^*+4}(v_1, V))\) to the current solution \(D\). After that, the algorithm removes all nodes in \(\Gamma_{r_1^*+2}(v_1, V)\) from the graph \(G\) and we denote the set of remaining nodes by \(V'\). Note that here we do not remove all nodes that are dominated by the current solution \(D\) from the graph. This is an important difference to the approach in [92]. As we will show, this modification guarantees that the final solution will be a connected dominating set.

In the reduced graph, the algorithm chooses another node \(v_2 \in V'\), considers growing neighborhoods of \(v_2\), until a \(r_2^*\) satisfying In-
equality 5.1 is found. Note that the bounding function $f$ of the original graph is still valid for the reduced graph, because any set which is independent in the reduced graph is also independent in the original graph. Further note that nodes in the sets $C(\Gamma_{r^2}(v_2, V'))$ and $C(\Gamma_{r^2+4}(v_2, V'))$ might be dominated by nodes that have already been removed. Successively, $C(\Gamma_{r^2+4}(v_2, V'))$ is added to the current solution $D$, and all nodes in $\Gamma_{r^2+2}(v_2, V')$ are removed from the graph, just as before. Then, this procedure is repeated until all nodes have been removed from the graph.

The algorithm is described formally in Algorithm 4. A schematic representation is depicted in Figure 5.2. Since the set of remaining nodes should always be clear from the context, we omit the second argument of $\Gamma(\cdot, \cdot)$ in the rest of the chapter.

Algorithm 4: Computes a $(1 + O(\varepsilon))$-approximate MCDS

1 Algorithm: CONNECTEDDOMINATINGSET

   Input: A p-Growth-Bounded graph $G = (V, E)$, $\varepsilon > 0$

   Output: A $(1 + O(\varepsilon))$-approximate Connected Dominating Set

2 $D := \emptyset$

3 Set $i := 1$

4 while $V \neq \emptyset$ do

5     Choose any $v \in V$ and call it $v_i$

6     Take the smallest $r$ that fulfills:

7          $|\text{MaxIS}(N(\Gamma_r(v)) \setminus \Gamma_r(v))| \leq \varepsilon \cdot |\text{MaxIS}(\Gamma_r(v))|$

8     Set $S_i := \Gamma_r(v), T_i := \Gamma_{r+4}(v)$

9     $D := D \cup C(T_i)$

10    $V := V \setminus \Gamma_{r+2}(v)$

11    $i := i + 1$

12 end

13 return $D$

For proving Lemma 5.4, we need the following fact.

**Lemma 5.3.** Consider any class of p-Growth-Bounded graphs with bounding function $f$, and a graph $G = (V, E)$ of this class. Then for any $r \geq 1$ and $v \in V$ it holds:

$$|\text{MaxIS}(N(\Gamma_r(v)) \setminus \Gamma_r(v))| \leq f(2) \cdot |\text{MaxIS}(\Gamma_r(v) \setminus \Gamma_{r-1}(v))|$$
Figure 5.2: The balls represent the expansions performed by Algorithm 4. The algorithm sequentially adds the optimal CDS’s of the larger intersecting balls to the solution and removes the nodes of the shaded (smaller) balls.

Proof. Clearly, each node in MaxIS(N(Γ_r(v))\Γ_r(v)) has a neighbor in Γ_r(v) in G. As MaxIS(Γ_r(v)\Γ_{r-1}(v)) is a dominating set of Γ_r(v)\Γ_{r-1}(v), each node in MaxIS(N(Γ_r(v))\Γ_r(v)) has a node in MaxIS(Γ_r(v)\Γ_{r-1}(v)) within distance at most two. However, the number of nodes in MaxIS(N(Γ_r(v))\Γ_r(v)) that lie within two hops of the same node in MaxIS(Γ_{r-1}(v)) can be at most f(2), as otherwise these nodes could not be mutually independent.

Lemma 5.4. Consider any class of p-Growth-Bounded graphs with bounding function f. Then, for any ε > 0, there is a R_f^* = R_f^*(ε) = O(1/ε \cdot \log(1/ε)) such that for each graph G of this class, and each node v of G, it holds

$$|\text{MaxIS}(N(Γ_r^*(v))\Gamma_r^*(v))| ≤ ε \cdot |\text{MaxIS}(Γ_r^*(v))|$$

for some r^* ≤ R_f^*.

Proof. Fix an ε > 0 and assume for contradiction that no such R_f^* exists. This implies that for arbitrarily large values r’, there is a graph in the class such that for some node v, $|\text{MaxIS}(N(Γ_{r'}(v))\Gamma_{r'}(v))| > ε \cdot |\text{MaxIS}(Γ_{r'}(v))|$ holds for all $0 ≤ r ≤ r'$. Consider such a value $r' ≥ 2$. From

$$|\text{MaxIS}(N(Γ_{r'}(v))\Gamma_{r'}(v))| > ε \cdot |\text{MaxIS}(Γ_{r'}(v))|$$

and Lemma 5.3 we have

$$|\text{MaxIS}(Γ_{r'}(v)\Gamma_{r'-1}(v))| > ε \cdot |\text{MaxIS}(Γ_{r'-2}(v))|,$$
5.5. Finding a Small Connected Dominating Set

for \( \bar{\epsilon} = \epsilon / f(2) \). Hence, for all \( 2 \leq r \leq r' \), we get:

\[
|\text{MaxIS}(\Gamma_r(v))| \geq |\text{MaxIS}(\Gamma_r(v) \setminus \Gamma_{r-1}(v))| + |\text{MaxIS}(\Gamma_{r-2}(v))|
\]
\[
> (1 + \bar{\epsilon}) \cdot |\text{MaxIS}(\Gamma_{r-2}(v))|.
\]

Assume for the moment that \( r' \) is an even number. Then we have

\[
|\text{MaxIS}(\Gamma_{r'}(v))| > (1 + \bar{\epsilon}) \cdot |\text{MaxIS}(\Gamma_{r'-2}(v))|
\]
\[
> (1 + \bar{\epsilon})^2 \cdot |\text{MaxIS}(\Gamma_{r'-4}(v))|
\]
\[
> \ldots > (1 + \bar{\epsilon})^{r'/2} \cdot |\text{MaxIS}(\Gamma_0(v))| = (1 + \bar{\epsilon})^{r'/2}.
\]

Since \( |\text{MaxIS}(\Gamma_{r'}(v))| \) grows only polynomially in \( r' \), but the term \( (1 + \bar{\epsilon})^{r'/2} \) grows exponentially in \( r' \) (note that \( (1 + \bar{\epsilon})^{1/2} > 1 \), the above inequality will be violated for some large enough \( r' \), which is a contradiction. If \( r' \) is odd, the same reasoning can be applied.

The claimed bound on \( R^*_f \) follows easily from the inequality \( (1 + \bar{\epsilon})^{1/\bar{\epsilon}} > e - 1 \), for \( \bar{\epsilon} \) small enough.

It is clear that Algorithm 4 terminates, and that \( D \) then contains a Dominating Set, because only dominated nodes are removed from the graph. Let \( S_1, S_2, \ldots, S_k \) and \( T_1, T_2, \ldots, T_k \) be the sets \( \Gamma_r(v_i) \) and \( \Gamma_{r+4}(v_i) \) respectively as chosen in each iteration of the outer while-loop of Algorithm 4. We now show that the computed solution \( D \), which consists of the union of the \( C(T_i) \), forms a connected subgraph of \( G \).

**Lemma 5.5.** The union \( D := \bigcup_{i=1}^k C(T_i) \) induces a connected subgraph of \( G \).

**Proof.** First, we show that any two nodes in \( D \) with distance exactly two in \( G \) belong to the same connected component in the subgraph induced by \( D \). To that end, pick any two such nodes \( u, v \in D \) and consider a node \( w \in V \) such that \( u, v \in N(w) \) (see the left part of Figure 5.3). Let \( s \) be the first node among \( N(w) \) (that includes \( w \)) that is removed in line 9 of Algorithm 4. When \( s \) is removed in the \( i \)th iteration, it holds that \( s \in \Gamma_{r_i+2}(v_i) \) and so all nodes in \( N(w) \) are in \( T_i = \Gamma_{r_i+4}(v_i) \), whereby \( v_1, v_2, \ldots, v_k \) represent the centers of the expansions. Hence they are dominated by \( C(\Gamma_{r_i+4}(v_i)) \) (they can possibly join \( D \) in another expansion). Therefore, any pair of nodes in \( D \cap N(w) \) is connected by construction. Note that by the definition of
p (Lemma \[5.1\]), the path consisting only of nodes in \(D\) that connects \(u\) to \(v\) has length at most \(p(R^* + 4) + 1\) hops.

Second, consider any pair \(u, v \notin D\) of nodes adjacent in \(G\) (see the right part of Figure 5.3). We show that there must exist nodes \(u' \in N(u) \cap D\) and \(v' \in N(v) \cap D\) such that \(u'\) and \(v'\) are connected by a path of length at most \(p(R^* + 4) - 1\) consisting only of nodes in \(D\). To that end, assume without loss of generality that \(u\) is removed (line 9) before or during the same iteration as \(v\). When \(u\) is removed in the \(j\)th iteration (i.e., \(u \in \Gamma_{r_j^*(v)} + 4(v_j)\)), also \(v\) is dominated by \(C(\Gamma_{r_j^*+2(v_j)})\). The second claim follows. Combining this claim with the first claim, we have that any two nodes \(u, v \in D\) with distance \(d(u, v) = 3\) in \(G\) are connected by a path of length at most \(3p(R^* + 4) - 1\) consisting only of nodes in \(D\).

The argument above easily imply that \(D\) induces a connected subgraph. Indeed, if \(D\) were disconnected, then the shortest path in \(G\) between two closest components of \(D\) would consist solely of nodes in \(V \setminus D\). However, such a path cannot be of length two or three by the above facts, respectively, and not longer either, because then \(D\) would not dominate all the nodes.

\[\square\]

\begin{figure}[h]
\centering
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{figure53a.png}
\caption{Illustration of Lemma \[5.5\]. The dashed edge represent a path.}
\end{subfigure}
\hfill
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{figure53b.png}
\end{subfigure}
\end{figure}

Now that we have shown that the set \(D\) computed by Algorithm \[4\] is a Connected Dominating Set, we prove that its size is at most \(1 + \varepsilon\) times larger than the optimum. To this end, we need two lemmas.

**Lemma 5.6.** Let \(\varepsilon > 0\) and \(r > 0\) be such that the following inequality is fulfilled:

\[|\text{MaxIS}(N(\Gamma_r(v)) \setminus \Gamma_r(v))| \leq \varepsilon \cdot |\text{MaxIS}(\Gamma_r(v))|.\]
Then, for \(\varepsilon' := \varepsilon \cdot (3f(4) + 3) \cdot f(1)\), it follows:

\[
|C(\Gamma_{r+4}(v))| \leq (1 + \varepsilon') \cdot |C(\Gamma_r(v))|.
\]

**Proof.** We show how to extend \(C(\Gamma_r(v))\) to a Connected Dominating Set of \(\Gamma_{r+4}(v)\) by adding only relatively few nodes so that the claim follows. Let \(M\) be a Maximal Independent Set of \(\Gamma_{r+1}(v) \setminus \Gamma_r(v)\). Clearly any node in \(\Gamma_{r+4}(v)\) lies within three hops of some node in \(\Gamma_{r+1}(v) \setminus \Gamma_r(v)\), and thus within four hops of some node in \(M\). Thus, all nodes in \(\Gamma_{r+4}(v)\) are dominated if we add to our solution the set \(C(\Gamma_4(w))\) for each \(w \in M\). Recall that \(|C(\Gamma_4(w))| \leq p(4) \leq 3f(4)\). In order to connect these sets to \(C(\Gamma_r(v))\), we need to add at most 3 additional nodes for each \(w \in M\). Thus in total, we have:

\[
|C(\Gamma_{r+4}(v))| \leq |C(\Gamma_r(v))| + (3f(4) + 3)|\text{MaxIS}(\Gamma_{r+1}(v) \setminus \Gamma_r(v))| \\
\leq |C(\Gamma_r(v))| + (3f(4) + 3)|\text{MaxIS}(N(\Gamma_r(v)) \setminus \Gamma_r(v))| \\
\leq |C(\Gamma_r(v))| + \varepsilon \cdot (3f(4) + 3)|\text{MaxIS}(\Gamma_r(v))| \\
\leq |C(\Gamma_r(v))| + \varepsilon \cdot (3f(4) + 3) \cdot f(1) \cdot |C(\Gamma_r(v))|,
\]

using Lemma 5.2 for the last inequality. \(\square\)

Let \(V^*\) be the set of nodes chosen as centers \(v\) for the growing neighborhoods in the algorithm. As for each \(v \in V^*, \Gamma_{r+2}(v)\) is removed from the graph before choosing a new node, the collection \(\{S_1, S_2, \ldots, S_k\}\) consists of 2-separated sets. We have the following lower bound for the size of an optimal CDS for \(G\).

**Lemma 5.7.** Let \(S_1, S_2, \ldots, S_k\) be the collection of 2-separated sets in \(G = (V, E)\) computed by Algorithm 4. Then,

\[
(1 + \varepsilon'') \cdot |C(V)| \geq \bigcup_{i=1}^{k} C(S_i),
\]

for \(\varepsilon'' := 4f(1)\varepsilon\) and \(\varepsilon \leq \frac{1}{4f(1)}\)\(^3\)

**Proof.** Since the \(S_i\) are 2-separated, the sets \(N(S_i)\) are pairwise disjoint, so the sets \(C(V) \cap N(S_i)\) are pairwise disjoint, too. Furthermore, as \(C(V)\) must dominate all nodes of \(G\), including those in \(S_i\),

\(^3\)Note that this assumption is without loss of generality. If \(\varepsilon > \frac{1}{4f(1)}\), we can set \(\varepsilon = \frac{1}{4f(1)}\): this only improves the approximation ratio.
the set \( C(V) \cap N(S_i) \) must dominate all nodes in \( S_i \). To complete the proof, we now show that \( |C(S_i)| \leq (1+\varepsilon'') \cdot |C(V) \cap N(S_i)| \) for all \( i \), and thus, \( |C(V)| \geq \sum_{i=1}^{k} |C(V) \cap N(S_i)| \geq \frac{1}{1+\varepsilon''} \cdot \sum_{i=1}^{k} |C(S_i)| \).

To that end, we add some nodes to \( C(V) \cap N(S_i) \) in order to obtain a connected set which dominates \( S_i \). Let \( x \) be the number of connected components in \( C(V) \cap N(S_i) \) and suppose that \( x \geq 2 \), otherwise \( C(V) \cap N(S_i) \) is connected and hence \( |C(S_i)| \leq (1 + \varepsilon'') \cdot |C(V) \cap N(S_i)| \) is trivial. Then the individual connected components of \( C(V) \cap N(S_i) \) can be connected by adding at most \( 2x \) nodes (see the proof of Lemma 5.1). Note that each connected component of \( C(V) \cap N(S_i) \) must contain one node from \( N(N_r(v)) \setminus N_r(v) \) to ensure the global connectivity of the solution. Thus, by choosing one such node for each connected component of \( C(V) \cap N(S_i) \), we obtain an independent set of size \( x \). Therefore, we have \( x \leq |\text{MaxIS}(N(S_i))| \). By construction of the sets \( S_i \), we have that \( |\text{MaxIS}(N(S_i))| \leq \varepsilon \cdot |\text{MaxIS}(S_i)| \). Further, by Lemma 5.2 the inequality \( |C(S_i)| \geq |\text{MaxIS}(S_i)|/f(1) \) holds. This shows that 

\[
|C(S_i)| \leq |C(V) \cap N(S_i)| + 2x \leq |C(V) \cap N(S_i)| + 2\varepsilon f(1) |C(S_i)|,
\]

and hence

\[
|C(V) \cap N(S_i)| \geq (1 - 2\varepsilon f(1)) |C(S_i)|.
\]

The claim now follows by choosing \( \varepsilon'' = \frac{2f(1)\varepsilon}{1-2f(1)\varepsilon} \leq 4f(1) \cdot \varepsilon \), if \( \varepsilon \leq \frac{1}{4f(1)} \).

\[\square\]

**Theorem 5.8.** The set \( D \) computed by Algorithm 4 is a \((1 + O(\varepsilon))\)-approximation for a Minimum Connected Dominating Set.

**Proof.** Let \( \{S_1, S_2, \ldots, S_k\} \) and \( \{T_1, T_2, \ldots, T_k\} \) be as defined in Algorithm 4. By Lemma 5.6 (and with variables defined there), it holds that \( |C(T_i)| \leq (1 + \varepsilon') \cdot |C(S_i)| \) for all \( i = 1, \ldots, k \), and \( D = \bigcup_{i=1}^{k} C(T_i) \). Hence we have

\[
|D| = |\bigcup_{i=1}^{k} C(T_i)| \leq \sum_{i=1}^{k} |C(T_i)| \leq (1 + \varepsilon') \cdot \sum_{i=1}^{k} |C(S_i)|
\]

\[
= (1 + \varepsilon') \cdot |\bigcup_{i=1}^{k} C(S_i)| \leq (1 + \varepsilon')(1 + \varepsilon'') |C(V)|
\]

\[
= (1 + O(\varepsilon)) |C(V)|,
\]

Chapter 5. The MCDS Problem
where the last inequality follows from Lemma 5.7.

We now shortly discuss how Algorithm 4 can be implemented in a centralized fashion to obtain a PTAS. Most steps of the algorithm can be trivially computed efficiently. The crucial part is the computation of the Maximum Independent Sets MaxIS\((N(Γ_r(v)) \backslash Γ_r(v))\) and MaxIS\((Γ_r(v))\), and of \(C(Γ_r∗ +4(v))\). Note that \(r\) is bounded by the constant \(R∗\) for any fixed \(ε\), so from the growth-bounded property we know that the size of the MaxIS is bounded by a constant. Thus, by enumerating all node subsets of cardinality at most \(f(r)\), and selecting the largest of those which is both independent and maximal, a Maximum Independent Set is found in polynomial time. Since the considered subsets have only constant size, independence and maximality of the subsets can be checked in constant time. The same arguments apply to the computation of \(C(Γ_r∗ +4(v))\), due to Lemma 5.1. Hence, Algorithm 4 has polynomial time complexity for any fixed \(ε > 0\), but exponential time complexity in \(1/ε\).

5.6 A Fast Distributed Approximation Scheme

The main goal of this chapter is to provide a fast distributed algorithm that computes a \((1 + ε)\)-approximation for the MCDS problem in the class of p-Growth-Bounded graphs.

The algorithm that we describe here is an adaptation from [64], adjusted to compute a CDS instead of a DS.

A naive distributed implementation would be that in each round all nodes which have the highest identifier within their \(2R∗ + 9\)-hop neighborhood are expanded concurrently. However, this approach requires a linear number of rounds in the worst case, because there can be a linear waiting chain of nodes, similarly as the case of the MIS. The observation that every expansion affects only neighbors within a small radius leads to a more efficient algorithm: expansions of nodes with sufficient mutual distances can be scheduled concurrently. Roughly speaking, this can be achieved by computing a MIS of \(G\), and then coloring this MIS with few colors such that two nodes with the same color are distant enough. The coloring is achieved using a clustergraph \(\bar{G}\) of radius \(c = c(ε)\) with centers \(W ⊂ V\). The clustergraph
\( \bar{G} = (\bar{V}, \bar{E}) \) is defined by \( \bar{V} := W \), and for every \( u, v \in W \) we have \((u, v) \in \bar{E} \) iff \( d(u, v) \leq c \). Note that if \( W \) is an Independent Set and \( G \) is Growth-Bounded, \( \bar{G} \) has a maximum degree of \( \Delta_{\bar{G}} = O(f(c)) \). Hence using a MIS of \( G \) to construct a clustergraph \( \bar{G} \) of radius \( c \), \( \bar{G} \) can be colored with \( O\left(\Delta_{\bar{G}}^2\right) \) colors in \( O\left(c \cdot \log^* n\right) \) time \[78\]. Note that a communication round in the cluster graph costs \( O\left(\text{rounds} \right) \) in the original graph.

The coloring is then used to schedule the expansions. Notice that a MIS-node \( v \) might already have left the graph, so the expansion is performed from a neighbor of \( v \) (or \( v \) itself). If no node of \( N(v) \) is still present in the graph, the expansion of \( v \) is clearly not necessary. We choose \( c = 2R^* + 11 \) for reasons that will become apparent in the proof of Lemma 5.10.

A more detailed description is given in Algorithm 5.

---

**Algorithm 5**: Computes a \((1 + O(\varepsilon))\)-approximate MCDS distributively

1. **Algorithm**: **DISTRIBUTEDCONNECTEDDOMINATINGSET**
   - **Input**: A p-Growth-Bounded graph \( G, \varepsilon > 0, R^* \) (according to Lemma 5.4)
   - **Output**: A \((1 + O(\varepsilon))\)-approximate Connected Dominating Set
2. Compute a MIS \( I \) of \( G \)
3. Construct the clustergraph \( \bar{G} \) of \( I \) using radius \( 2R^* + 11 \)
4. Color \( \bar{G} \) with \( \gamma = O\left(\Delta_{\bar{G}}^2\right) \) colors
5. \( D := \emptyset \)
6. for \( k = 1 \) to \( \gamma \) do
   7. for every \( v \in I \) of color \( k \) do concurrently
   8. Take a \( u \in N(v) \cap V \) (if any), find the least \( r^* \) with:
      \[ |\text{MaxIS}(N(\Gamma_{r^*}(u)) \setminus \Gamma_{r^*}(u))| \leq \varepsilon |\text{MaxIS}(\Gamma_{r^*}(u))| \]
     Compute \( C(\Gamma_{r^*+4}(u)) \)
     Inform \( \Gamma_{r^*+4}(u) \) about \( r^* \) and \( C(\Gamma_{r^*+4}(u)) \)
   9. \( D := D \cup C(\Gamma_{r^*+4}(u)) \)
   10. \( V := V \setminus \Gamma_{r^*+2}(u) \)
7. end
8. end
9. return \( D \)
Lemma 5.9. Algorithm 5 terminates in $O \left( T_{\text{MIS}} + \frac{1}{\varepsilon} \cdot \log^* n \right)$ time.

**Proof.** Computing a MIS of $G$ takes time $T_{\text{MIS}}$. Then, the clustergraph can be efficiently constructed, as its edges only span at most distance $2R^* + 11 = O \left( \frac{1}{\varepsilon} \cdot \log(1/\varepsilon) \right)$. Furthermore, $\bar{G}$ can be colored with $O \left( \Delta_{\bar{G}}^2 \right) = O \left( f^2(R^*) \right)$ colors in $O \left( R^* \cdot \log^* n \right)$ time using the well-known algorithm of [78].

The outer for-loop is executed $O \left( \Delta_{\bar{G}}^2 \right)$ times. Inside the for-loop, the number of different MaxIS that each node $u$ (as in line 8) must compute is $2r^* = O \left( R^* \right) = O \left( \frac{1}{\varepsilon} \cdot \log(1/\varepsilon) \right)$. For computing each MaxIS and MCDS for a neighborhood of radius $r$, $u$ collects all information about $\Gamma_r(u)$ and then computes the set locally. As $r \leq R^*$, all steps in lines 8 to 11 can be executed in $O \left( \frac{1}{\varepsilon} \cdot \log(1/\varepsilon) \right)$ time.

Lemma 5.10. The set $D$ computed by Algorithm 5 is a $(1 + O(\varepsilon))$-approximate Minimum Connected Dominating Set.

**Proof.** By construction, any two nodes that are concurrently used for an expansion have distance at least $2R^* + 9$, because they are respective neighbors of two MIS nodes of distance at least $2R^* + 11$. The radius used by either expansion is at most $R^*$, and since each expansion only involves the nodes within a radius of at most $R^* + 5$, all concurrent expansions would have the same result if they were executed sequentially. Therefore, there exists an execution of the sequential Algorithm 4, which computes the same set $D$ as Algorithm 5. It follows that Algorithm 5 achieves the same approximation ratio as Algorithm 4.

These two lemmas lead to our main theorem.

**Theorem 5.11.** For any $\varepsilon > 0$, Algorithm 5 computes a $(1 + O(\varepsilon))$-approximate MCDS in $O \left( T_{\text{MIS}} + \frac{1}{\varepsilon} \cdot \log^* n \right)$ rounds in the LOCAL model in the class of $p$-Growth-Bounded graphs.

### 5.7 Well-Connectedness

The Connected Dominating Set computed by Algorithm 4 is not only a $(1 + \varepsilon)$-approximation of a Minimum CDS, but has additional prop-
erties which are desirable for its usage as a backbone in a wireless
network. Let $G' = (V', E')$ be the graph induced by the CDS$_A$ of the
(p-Growth-Bounded) graph $G = (V, E)$ computed by Algorithm 4. Then, the following statements hold for any $\varepsilon > 0$:

1. The backbone graph $G'$ has maximum degree $O\left(1/\varepsilon^{O(1)}\right)$, and therefore it has only $O\left(1/\varepsilon^{O(1)} \cdot |V'|\right)$ edges;

2. $G'$ has a $O\left(1/\varepsilon^{O(1)}\right)$-stretch with respect to $G$.

We assume in the following that source $s$ and destination $d$ of a rout-
ing request are both members of the CDS. If this is not true for either or both of them, then we can easily choose a neighbor inside the CDS as a representative. This will add at most two hops to the routing path, so if the stretch is low for any pair $s, d$ inside the CDS, the stretch of any pair $s, d$ inside $G$ is also low. To make the second statement precise, define $\lambda := \max_{u, v \in V'} \frac{d_{G'}(u, v)}{d_G(u, v)}$ as the hop-stretch of $G'$. Furthermore, if $G$ is a Unit Disk graph and if $D_G(u, v)$ denotes the geometric length of a shortest path in $G$, then the geometric stretch of $G'$ is $\mu := \max_{u, v \in V'} \frac{D_{G'}(u, v)}{D_G(u, v)}$.

**Lemma 5.12.** Let $\text{CDS}_A$ be the CDS computed by Algorithm 4. The subgraph $G'$ of $G$ induced by the nodes in $\text{CDS}_A$ has maximum degree $O\left(1/\varepsilon^{O(1)}\right)$.

**Proof.** First, note that each partial CDS of $T_i$ computed by the algorithm covers a subgraph with diameter $O\left(R^*\right)$, so according to Lemma 5.1 $|T_i|$ and therefore its degree is at most $O\left(f(R^*)\right)$. Second, each node $u$ of $G$ can only be contained in $O\left(f(R^*)\right)$ many different $T_i$, because any expansion that leads to some $T_j$ containing $u$ must have as its center a MIS-node in distance at most $R^* + 4 = O\left(1/\varepsilon \cdot \log(1/\varepsilon)\right)$ from $u$ (and there are only $O\left(f(R^*)\right)$ MIS nodes in distance $O\left(R^*\right)$ from $u$).

**Lemma 5.13.** For any $\varepsilon > 0$, the hop-stretch $\gamma$ of $G'$ is $O\left(f(R^*)\right) = O\left(1/\varepsilon^{O(1)}\right)$. Further, if $G$ is a Unit Disk graph, then the geometric stretch $\lambda$ of $G'$ is also $O\left(1/\varepsilon^{O(1)}\right)$.

**Proof.** Consider any source $s$ and destination $d$ in $V'$. Let $\mathcal{P} = \langle p_1, p_2, \ldots, p_k \rangle$ be the sequence of nodes in a shortest path in $G$ from $s = p_1$ to $d = p_k$. We define a new path $\mathcal{Q}$ going through nodes
⟨q₁, q₂, . . . , qₖ⟩ as follows: qᵢ := pᵢ if pᵢ ∈ D. Otherwise, let qᵢ be any node in D ∩ N(pᵢ) (such a node exists because D is dominating). Note that qᵢ ∈ D, for all 1 ≤ i ≤ k. From the proof of Lemma 5.5 we can conclude that between any pair (qᵢ, qᵢ₊₁), 1 ≤ i ≤ k − 1, there is a path in D of length at most 3p(R* + 4) − 1 = O(f(R*)). Hence there is a path Q of length ≤ k(3p(R* + 4) − 1) = k · O(f(R*)) = k · O(1/εO(1)) from q₁ to qₖ solely consisting of nodes in D.
For the geometric stretch in Unit Disk graphs, note that in the path R := ⟨t₁ = s, t₂, . . . tₖ = d⟩ of shortest geometric length, the outer two of any three consecutive nodes tᵢ, tᵢ₊₁, tᵢ₊₂ must have distance at least 1: ∥tᵢ − tᵢ₊₂∥₂ ≥ 1, for all i ∈ {1, . . . , k − 2}. So R with k hops has length at least (k − 1)/2. On the other hand, the path with the fewest number of hops (at most k) has length at most k. Since we have shown just before that G’ includes a path with hop-stretch γ = O(1/εO(1)) between any pair of nodes, it follows that the geometric stretch λ of G’ is at most 2k · γ/(k − 1) ≤ 4γ = O(1/εO(1)), k ≥ 2. For k = 1, the path with fewest hops has length at most γ = O(1/εO(1)), which completes the claim.

Summarizing, we have the following.

**Theorem 5.14.** The CDS computed by Algorithm 4 / Algorithm 5 is well-connected.

### 5.8 Open Problems

The distributed algorithm we presented computes a (1 + ε)-approximate Minimum Connected Dominating Set with O(1/εO(1)) maximum degree and stretch in time O(1/εO(1) · log* n). Any approach that requires a MIS for constructing a CDS must have a running time of Ω(1/εO(1) · log* n) because of the lower bound by Linial [78]. It is an intriguing open question whether a constant approximation for the MCDS problem can be achieved in o(1/εO(1) · log* n) rounds using a fundamentally different approach.
Chapter 6

The Small Connected Spanning Subgraph Problem – Algorithms and Lower Bounds

As the poet said, “Only God can make a tree” - probably because it’s so hard to figure out how to get the bark on. – Woody Allen (born December 1, 1935), three-time Academy Award-winning American film director, writer, actor, jazz musician, comedian and playwright.

6.1 Introduction and Motivation

A Spanning Tree is a basic structure in graph theory with a stunning amount of applications in computer science and algorithmics. The distributed computation of Minimum Spanning Trees in weighted graphs has triggered a considerable amount of research – see for instance [34, 35, 46, 70, 71, 102]. There are many reasons for this
interest. On one side, even though the problem of computing a minimum spanning tree is trivial in the LOCAL model, the locality of the problem in the CONGEST model is hard to capture. Spanning Trees are of central interest in distributed computation, because many other problems can be easily related to them. An example are Graph Spanners or Tree Spanners where the aim is to compute a spanning subgraph with limited stretch factor (for a survey see [37]).

In Chapter 5, we have modelled the virtual backbone of a wireless ad hoc network as a Connected Dominating Set to ease, for instance, the broadcast operation. If we measure the energy cost of a broadcast operation by the number of links used and we neglect the duration of the broadcast, a sparse Spanning Subgraph of the network is better suited to serve as a routing backbone. Maintenance applications in a network (such as the control of the status of the nodes in the network) have the requirement to be possibly inexpensive to the network.

As mentioned above, the problem of computing a Spanning Tree is trivial in the LOCAL model and requires a number of rounds linear in the network diameter, which might be unacceptable. This fact triggers a natural question. If we do not insist on the optimality of the subgraph (with respect to the number of edges), is it possible to compute a sparse Spanning Subgraph efficiently? Formally, we are interested in the following problem, called the Small Connected Spanning Subgraph Problem (or SCSSP). Given a connected network \( G = (V, E) \) and a parameter \( |V| - 1 \leq k \leq |E| \) find a connected subgraph \( S = (V, E') \) where \( E' \subset E \) such that \( |E'| \leq k \). Clearly when \( k = |V| - 1 \), the subgraph of interest is a Spanning Tree.

### 6.2 Related Work

Computing Spanning Trees is one of the fundamental problems in the theory of networks. Naturally, in the distributed setting it assumes even more importance. Several asymptotic near-optimal distributed protocols are known for the problem [71, 34, 35] in the CONGEST model. For example, Kutten and Peleg [71] design a protocol for the problem with a running time of \( O(d(G) + \sqrt{n} \cdot \log^* n) \). This running time is to be compared to the lower bound of \( \Omega(d(G) + \sqrt{n} / \log n) \) for computing a Minimum Spanning Tree due to Peleg and Rubinovich [102]. Distributed algorithms for the exact computa-
tion of an exact Spanning Tree in the \textit{LOCAL} model are uninteresting because the naive algorithm running in $d(G) + 1$ rounds is optimal. In fact a Spanning Tree cannot be found in general in $o(d(G))$ rounds where $d(G)$ is the diameter of the graph $G$. One can deduce this result from the impossibility of electing a leader in a ring (a cycle of $n$ nodes) within $n$ rounds of synchronous computation (see [9]). Therefore, we consider the problem of finding Connected Spanning Subgraphs with few more edges.

In the non-distributed setting the focus is more directed towards Minimum-weight Spanning Trees and there efficient algorithms do exist (see for example [27]). The $O(m \log m)$-textbook-algorithms were successively improved resulting in an algorithm with a running time of $O(m \log \beta(m, n))$ \cite{44} where $\beta(m, n)$ is the minimum integer $i$ such that the value obtained by iterated logarithm on $n$ is less than $\frac{m}{n}$ with $n = |V|, m = |E|$. Fourteen years later, the running time was further improved by Chazelle \cite{22}, up to the current optimal algorithm due to Pettie and Ramachandran \cite{106}, who showed a deterministic comparison-based algorithm that finds a solution in $O(T^*(m, n))$ time, where $T^*(m, n)$ is the minimum number of edge-weight comparisons needed to determine the solution.

\section{Summary of Results}

We design a simple distributed algorithm that finds a Connected Spanning Subgraph with $(1 + \varepsilon)n$ edges in $\min\{d(G), O(\log n)\}$ rounds for any approximation ratio $\varepsilon > 0$ in the \textit{LOCAL} model. We reduce the number of required active nodes (nodes that perform computations) of this algorithm by means of a preprocessing phase that asymptotically takes the same amount of rounds as the previous algorithm. Moreover, we show that any synchronous distributed algorithm running for $o(\log n)$ rounds computes a Connected Spanning Subgraph with $\omega(n)$ edges in the worst case.

Algorithms exploiting the same ideas for other problems are already known in the literature \cite{33, 76}. However in this thesis we get much easier proofs by employing results from the field of Extremal Graph Theory.

The results of this chapter have been established in a joint work with Shankar L. Ram \cite{108}.
6.4 Results from Extremal Graph Theory

Turán’s theory deals with the question of establishing the maximal number of edges that a graph $G$ on $n$ vertices can possibly have under the condition that an isomorphic copy of a given graph $H$ is not a subgraph of $G$. We denote this quantity by $\text{ex}(n, H)$. An interesting theorem proved by Mantel in 1907 states that $\text{ex}(n, K_3) = \lfloor n^2 / 4 \rfloor$. The result was further generalized by Turán in 1941 to $\text{ex}(n, K_s) = \left(1 - \frac{1}{s-1}\right) \frac{n^2}{2}$ (see [16]). A celebrated result by Erdős and Stone [38] solves the problem meaningfully for all non-bipartite graphs $H$ by proving

$$\text{ex}(n, H) = \left(1 - \frac{1}{\chi(H) - 1}\right) \frac{n^2}{2} + o(n^2)$$

where $\chi(H)$ denotes the chromatic number of $H$. Several results are known when the subgraph to be avoided belongs to other classes of graphs. If we have to avoid more than one type of subgraph, the extremal number is denoted by $\text{ex}(n, H_1, H_2, \ldots, H_l)$. In other words the maximal number of edges a graph $G$ on $n$ vertices can have without containing any of $H_1, H_2, \ldots, H_l$ as a subgraph is given by $\text{ex}(n, H_1, H_2, \ldots, H_l)$.

With a method due to Moore and improved by Alon et al. [3] one can prove the upper bound of the following theorem. The lower bound is due to Margulis [84] using the famous Ramanujan graphs and to Lubotzky et al. [80].

**Theorem 6.1 ([3, 80, 84]).**

$$n^{1 + \frac{2}{3k+3}} \leq \text{ex}(n, C_3, C_4, \ldots, C_{2k+1}) \leq \frac{1}{2} n^{1 + \frac{1}{k}} + \frac{1}{2} n$$

6.5 The Distributed Algorithm

As the first step, we use the node identifiers to introduce an ordering of the edges. An edge $e$ is said to be bigger than an edge $e'$, if the identifiers of the endpoints of $e$ are lexicographically bigger than the identifiers of the endpoints of $e'$.

We assume that every node knows $n$, the number of nodes in the graph. Otherwise, our algorithms can be seen as anytime algorithms,
where the solution continuously improves along with the execution time.

In this section we describe a distributed algorithm for the SC-SSP where all nodes are active, i.e., the nodes participate actively in the algorithm execution (as opposed to only forward messages). We reduce the number of active nodes in the subsequent section. Our algorithm works as follows: the nodes explore the neighborhood $N_v(c(\varepsilon) \log n)$, for all vertices $v \in V$. In such a neighborhood, every vertex detects locally all the cycles and removes the lexicographically biggest edge of every cycle. We refer to this process of removing the biggest edge of a cycle as destroying a cycle. When this phase finishes, the graph looks locally like a tree. Note that by our assumption of the model, all these local computations are not measured in the time complexity of the algorithm. The algorithm is compactly described by LOCALTREE.

![Figure 6.1: The picture depicts a graph (left) and the same graph after removing the cycles of length four (right), as prescribed by the algorithm LOCALTREE. The dotted edges are removed from the graph.](image)

Now we analyze Algorithm LOCALTREE showing that it builds a Connected Spanning Subgraph with few edges.

### 6.5.1 The Algorithm

LOCALTREE returns a Connected Spanning Subgraph with large girth. This is our motivation to use the above theorem. From Theorem 6.1, we can prove a linear upper bound on the number of edges of the subgraph returned by Algorithm LOCALTREE. Note that this algorithm does not necessarily requires the model to be synchronous.
Algorithm 6: Small Spanning Subgraph

1 Algorithm: LOCALTREE

Input: A connected graph $G = (V, E)$, an approximation $\varepsilon > 0$.

Output: A Connected Spanning Subgraph of $G$ with $(1 + \varepsilon)n$ edges.

2 Set $c := 6/\log(1 + 2\varepsilon)$

3 Every vertex concurrently deletes the biggest (lexicographical) edge in every cycle $N_v(c \log n)$.

4 return Remaining edges

Theorem 6.2. For every $\varepsilon > 0$, LOCALTREE selects a Connected Spanning Subgraph with at most $(1 + \varepsilon)n$ edges in

$$\min\{d(G), O\left(\frac{1}{\log \varepsilon} \cdot \log n\right)\}$$

rounds in the LOCAL model.

Proof. Let $H = (V, E')$ denote the subgraph returned by the algorithm. Note that since we run the algorithm on all vertices, the subgraph is spanning. The claim about the running time is immediate, since the nodes require only the information about the $c \log n$-neighborhood. We need to verify that $H$ is connected and that $|E'| \leq (1 + \varepsilon)n$.

Connectivity. Order lexicographically the edges of the graph $G$ and set the weights $1, \ldots, m$ to the edges of $G$ according to this order. We show that the solution $H$ computed by LOCALTREE on the graph $G$ contains the Minimum Spanning Tree $T$ as a subgraph of the corresponding weighted graph and is hence connected. Suppose that it is not the case and let $e$ be an edge of $T$ which is deleted by the algorithm (note that all the nodes that find that cycle in their view remove precisely $e$). Thus, $e$ belongs to a short cycle of $G$ and has the biggest weight among the edges of that cycle. If we remove $e$ from the $T$ there has to be another edge of the cycle – not previously part of $T$ – that reconnects the spanning tree and has a smaller weight than $T$. This is a contradiction.

Number of edges. $H$ does not contain any cycle of length less than $c \log n$ ($c$ chosen as in the Algorithm description) since all the
cycles within the \( c \log n \)-neighborhoods of the vertices have been destroyed. Thus the remaining cycles of \( H \) have to be of length greater than \( c \log n \). Therefore by using Theorem 6.1 \( H \) cannot contain more than

\[
|E'| \leq \text{ex}(n, C_3, \ldots, C_{c \log n}) \leq \frac{1}{2} n^{1 + \frac{2 \log n - 1}{c \log n}} + \frac{1}{2} n
\]

edges.

\[
\leq \frac{1}{2} n^{2^{6/c}} + \frac{1}{2} n = (1 + \varepsilon)n
\]

Unfortunately, we have no guarantee about the stretch factor: if one considers the graph constituted by cycles of length \( O(\log n) \), pairwise merged on a common edge, one can define the labels in such a way, that two neighboring nodes get to distance \( n \) after the process.

### 6.5.2 The Weighted Case

Consider the case where the graph is provided with a weight function \( \ell : E \to \mathbb{R}^+ \) on the edges. \( \ell(e) \) could represent the cost to use the edge \( e \) during a broadcasting transmission or the latency one incurs by sending a message along \( e \). We are interested in a weighted version of SCSSP denoted \( \text{w-SCSSP} \). Here the goal is to find a Connected Spanning Subgraph of minimum total weight. Unfortunately for \( \text{w-SCSSP} \), one cannot design a fast distributed algorithm that has a provable approximation guarantee. Consider for example a cycle of length \( n \), where every edge has weight 1 except for one which has weight \( \vartheta \geq 1 \). The optimal connected spanning subgraph has a weight of \( n - 1 \) given by a minimum weight spanning tree (MST), but every algorithm working for \( o(d(G)) \) rounds necessarily selects all the edges of the cycle, yielding a connected spanning subgraph of total weight \( n - 1 + \vartheta \). This can be arbitrarily bad, since \( \vartheta \) can be chosen big.

On the other hand, if the weight function is bounded and the instance is large enough, our algorithm provides immediately a \((1 + \varepsilon)\)-approximation for this problem, by choosing an appropriately large \( c \) (refer to the Algorithm \textsc{LocalTree}). We can suitably modify the deletion rule by choosing the heaviest edge (with respect to the weight function) in the cycle, breaking ties as before.


Algorithm 7: Small Spanning Subgraph with few active nodes

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Algorithm: \textsc{SpanningSubgraphWithFewActiveNodes}</td>
</tr>
<tr>
<td>2</td>
<td>Input: A graph $G = (V, E)$, an approximation ration $\varepsilon &gt; 0$.</td>
</tr>
<tr>
<td>3</td>
<td>Output: A Connected Spanning Subgraph of $G$ with $(1 + \varepsilon)n$ edges.</td>
</tr>
<tr>
<td>4</td>
<td>Set $c := 6/\log(1 + 2\varepsilon)$</td>
</tr>
<tr>
<td>5</td>
<td>Construct a $\log n/\log \star n$-ruling set $S$</td>
</tr>
<tr>
<td>6</td>
<td>Every $v \in S$ concurrently deletes the biggest (lexicographical) edge in every cycle $N_v(2c \log n)$.</td>
</tr>
<tr>
<td>7</td>
<td>return Remaining edges</td>
</tr>
</tbody>
</table>

6.6 Reducing the Number of Active Nodes

LocalTree, although being time-optimal, requires that all the nodes in the network compute their $O(\log n)$-neighborhoods in order to destroy cycles. Here, we describe an algorithm that overcomes this issue by requiring only $o(n)$ nodes to be active. This is desirable because active nodes consume more power than passive nodes being involved in computational tasks as well.

The first phase of the algorithm is a preprocessing phase where a subset of size $o(n)$ of the nodes are chosen to work in the cycle destruction phase. At the end of this phase, the set of nodes will be the active nodes. In the literature there are different algorithms that find a $\kappa$-ruling set of size no greater than $\lfloor n/(\kappa + 1) \rfloor$ in $O(\kappa \cdot \log^* n)$ synchronous rounds [71, 104]. By choosing $\kappa = \log n/\log^* n$ we select a set of active nodes of size $o(n)$ in $O(\log n)$ rounds. Now it suffices to let the active nodes run the Algorithm LocalTree on their neighborhood with doubled radius $2c \log n$ with respect to Section 6.5.1. The ruling property ensures that every cycle of size $O(c \log n)$ is eventually destroyed, so the analysis follows as easy as before.

Theorem 6.3. There exists a distributed synchronous algorithm that computes for every $\varepsilon > 0$ a Connected Spanning Subgraph with $(1 + \varepsilon)n$ edges in $\min\{d(G), O(1/\log \varepsilon \cdot \log n)\}$ rounds, where at most $o(n)$ active nodes destroy the cycles of $G$. 
6.7 Lower Bound

A well-known result states that it is impossible to elect a leader in a ring (a cycle on \( n \) nodes) in less than \( \mathcal{O}(n) \) rounds of synchronous computation. This implies that building a Spanning Tree on a ring network also takes \( d(C_n) \) time where \( d(C_n) \) is the diameter of the ring. In fact, it is straightforward to obtain an efficient distributed algorithm for the Leader Election Problem from an efficient algorithm for computing a Spanning Tree.

In this section, we show a lower bound on the number of edges of the Connected Spanning Graph produced in the worst case by any distributed algorithm for the SCSSP running for \( o(\log n) \) rounds.

**Theorem 6.4.** Every distributed synchronous algorithm running for \( o(\log n) \) rounds with the aim of building a Connected Spanning Subgraph picks at least \( \omega(n) \) edges in the worst case.

**Proof.** Let us consider such an algorithm \( \mathcal{A} \). Since \( \mathcal{A} \) runs for \( o(\log n) \) rounds, every node explores at most its \( o(\log n) \)-neighborhood that leads to the \( o(\log n) \)-view. Thus every node has to base its decision only according its \( o(\log n) \)-view and it cannot therefore prevent cycles of length at least \( \Omega(\log n) \).

Let \( G = (V,E) \) be an extremal graph where the length of a shortest cycle is \( \Omega(\log n) \) and \( |E| \) is maximum among all such graphs. If this graph is given as input to \( \mathcal{A} \) then every node needs to choose all edges in its \( o(\log n) \)-neighborhood (which looks like a tree), in order to ensure connectivity. Indeed the leaves of this local tree might or not be connected within the rest of the original graph and this cannot be established from the respective view of the node. But this means that all the edges of the graph have to be output. Using the lower bound given in Theorem 6.1, this yields

\[
\begin{align*}
\text{ex}(n, C_3, C_4, \ldots, C_{o(\log n)}) & \geq n^{1 + \frac{2}{3/2\cdot o(\log n) + 1}} \\
& = n \cdot 2^{\frac{\log n}{o(\log n)}} = \omega(n)
\end{align*}
\]

edges. \( \square \)
6.8 Open Problems

The lower bound that we presented in the previous section trivially holds for the \textit{CONGEST} model as well, since it is weaker than the \textit{LOCAL} model. However the algorithms require to efficiently send the whole information about a neighborhood to the corresponding node. This can be achieved only in the \textit{LOCAL} model. How does the situation in the \textit{CONGEST} model look like? Is it still possible to construct a Connected Spanning Subgraph with \((1 + \varepsilon)n\) edges in \(\mathcal{O}(\log n)\) rounds for some \(\varepsilon > 0\)?
Part III

Exploring

Simple Robots in Polygonal Environments
Prologue

Florence, 1505 – Mona Lisa is looking around in Leonardo’s workshop. She looks interested in some of Leonardo’s projects.

Mona Lisa: “Leonardo, what does that drawing up there represent?”

Leonardo: “It’s a map of the Arno river that flows through Florence in Tuscany. I drew it for Cesare Borgia.”

Mona Lisa: “I know the place, but I don’t know what a map is.”

Leonardo: “Imagine to be a bird that flies over a particular area. Basically, a map is a description of what you see.”

Mona Lisa: “I understand. Why should Cesare Borgia want something like a map?”

Leonardo: “In fact, I drew many maps for Cesare and for different reasons. The one you are looking at serves as a plan to figure out where to divert the river behind Pisa to deprive the city of access to the sea. This would give an important advantage for Florence which is keeping Pisa under siege. I did other maps to provide him with an advantage for defending Florence from its enemies in case of a war.”

Mona Lisa: “Right, the advantages of a map are clearly important. Though, I don’t see wings attached to your back and your project for the helicopter has not been realized yet. How did you manage to have a bird-eye view?”

Leonardo: “The process takes quite a long time. You need to achieve an overall understandings of the environment by looking only at small portions of the area at once.”

Mona Lisa: “This is clearly an hard task. Let alone the fact that you don’t have many measuring tools at hand.”

Leonardo: “Correct. What I did was to go on place personally. For the maps of a city, I measured every single path by the number of steps it takes to walk it. For larger environments this is not possible and I was roughly estimating the lengths.”

Mona Lisa: “And this is probably not going to be very precise.”
Leonardo: “Yes, it’s only an approximation and it took me no less than ten months to complete.”

Mona Lisa: “But the artistic beauty of your sketches is incontrovertible.”

Leonardo: “You should tell that to the thousands of men that are working on the project of changing the path of the Arno river. So far it doesn’t seem that the plan is going to be successful. However using the maps I drew, I also figured out a way to connect Florence to the sea by water. It’s only a matter of bypassing a short unnavigable stretch. I hope that this plan will be reality.”

Mona Lisa: “I’m sure that one day, when the measuring tools will be precise enough, it will.”

Note from the author. The express highway that connects Florence to the sea flows over the exact route that Leonardo indicated for his canal. For further information, see [36].
Chapter 7

Introduction

*It’s not what you look at that matters, it’s what you see.* – Henry David Thoreau (July 12, 1817 – May 6, 1862) American author, naturalist, transcendentalist, tax resister, and philosopher.

7.1 Motivation

The study of simple robot systems, with minimalistic sensory input, is of fundamental interest in both theory and practice. In theory, a minimalistic model provides a clean conceptual framework for performance analysis and lends insights into the inherent complexity of various tasks: the positive results identify the easy problems, while the negative results help isolate the difficult problems that necessitate richer functionality and sensing. Models with simple sensing are also robust against noise and help simplify the information (belief) space of the robot systems, whose complexity is often a source of much difficulty in planning [73]. On the practical side as well, robots with a simple sensing architecture have many advantages: they are inexpensive, less susceptible to failures, robust against sensing uncertainty, and useful for different applications. With the emergence of wireless sensor networks [107], a group of simple microrobots also offers an attractive and scalable architecture for large-scale collaborative exploration of unknown environments.

Nowadays, rapid technological innovation gives rise to new hopes
and exciting possibilities for microrobots. For instance, camera sensors continuously become smaller, cheaper and provide images of higher quality. It is natural for robotics engineers to build sophisticated robots that use many of the available features. It is, however, not always clear that a more complex information gained from more sophisticated sensor is what a robot needs in order to solve a task at hand. For instance, the project SToMP (Sensors, Topology, and Minimalist Planning) of the University of Illinois investigates questions of this nature. The current understanding of robotics is not united in answering the question of which sensor is better for certain tasks. Giving all possible sensors/features to the robot may spare scientists from asking such questions, but it does not come for free, and one has to justify the increased cost of building, memory usage, energy consumption, etc. It is therefore an important research question to understand the limitations and strengths of available features/sensors.

We take a step in this direction and try to understand what is the information (gained via sensors) that a robot necessarily needs to solve a certain task. We compare several sensors within the framework of simple robots in polygonal environments. Since the robot’s information is local (the visibility region can be a small subset of the entire workspace), we investigate tasks that are seemingly global. The sensors can be, e.g., the capability to measure distances between visible vertices, or the capability to measure angles between visible vertices. We compare the resulting robot models with the aim to determine their computational strength. At this stage of our investigation we do not want to compare how effective the robots may be in dealing with a massive amount of data. Instead we base our comparison on the notions of “possibility” – we are interested whether the considered robots can solve the same task, and we do not compare how much time, memory, energy, etc. the robots need to complete the task. In some cases, we shortly speak about efficiency as well.

To give a concrete example, consider a polygonal maze with one exit, where every wall has a different color. There are three different simple robots placed in the maze. All robots can see the vertices and the walls of the polygonal maze. Robot A can additionally measure the distance between its position and a visible vertex, robot B can see the color of the wall, and knows in advance the order of the colors as they appear on the boundary, and robot C can decide whether an angle between two visible vertices and its position is convex or reflex.
These robots have seemingly very different level of sophistication. The task for the robots is to find the exit. This is of course an easy task as a robot can walk along the boundary and it eventually finds the exit. In this thesis we prove that the three robots can do more: they can determine a Euclidean shortest path between their position and the exit of the maze. Even though this sounds plausible for robot A, we think it is very surprising for the other robots that have no distance measuring capability. This motivates the need to understand the true strengths (and weaknesses) of idealized robots.

7.2 Definitions and Models

7.2.1 Polygons and Visibility

The workspace of the robots is modeled as a polygon $P$, which might be simply or multiply-connected (with holes). A polygon is a connected compact set $P$ of the plane with the property that the topological boundary of $P$ is composed by a disjoint union of non-intersecting piecewise-linear curves, called boundary components of $P$. If a polygon has a single boundary component, then it is called a simply-connected polygon, otherwise it is called a multiply-connected polygon (with holes). For short, we call a simply-connected polygon $P$ simple and its boundary component the boundary of $P$. Many of the presented results hold for simple polygons only, so we tacitly assume to deal with simply-connected polygons, unless stated otherwise. The size of a polygon is the number of its vertices (including the vertices of the holes, if the polygon is multiply-connected). In a simple polygon we denote by $V(P) = \{v_0, \ldots, v_{n-1}\}$ the set of vertices of the boundary, which we assume are ordered counterclockwise (ccw). Every vertex $v_i$, $0 \leq i \leq n - 1$, is a point in the Euclidean plane. We assume, for simplicity of exposition only, that no three vertices lie on a line. Let $E(P) = \{e_0, \ldots, e_{n-1}\}$ be the edge-set of a polygon $P$, where an edge is the open segment $e_i = (v_i, v_{i+1})$, $i = 0, \ldots, n - 1$. Occasionally, we denote an edge $e_i$ by $v_i v_{i+1}$. All arithmetic operations on the indices of the vertices are to be understood modulo $n$. Given two points $p_1$ and $p_2$ in the polygon, we denote by $p_1 p_2$ the line segment between the two points. We say that $p_2$ is visible to $p_1$ in $P$ if $p_1 p_2 \cap P = p_1 p_2$ (note that the relation visible is symmetric). Two (mutually) visible vertices $v_i, v_j \in V(P), j \notin \{i - 1, i + 1\}$
form a *diagonal* of $P$.

A labeled set of points $p_0, \ldots, p_{n-1}$ in the plane with the property that no two open segments $(p_i, p_{i+1})$, $(p_j, p_{j+1})$, $0 \leq i < j \leq n-1$, intersect induces a polygonal boundary (and hence a polygon) in a natural way by connecting every point $p_i$ to its successor $p_{i+1}$ by a line segment, $i = 0, \ldots, n-1$. Given a polygon $P$, a sequence of vertices $v_{i_0}, \ldots, v_{i_k} \in V(P)$, $i_0 < \ldots < i_k$, with the property that $v_{i_l}$ and $v_{i_{l+1}}$ are visible in $P$, $0 \leq l \leq k$, induce a *subpolygon* $P'$ of $P$. The vertex set of $P'$ is $\{v_{i_0}, \ldots, v_{i_k}\}$ and the edge set is $\{e'_{i_0}, \ldots, e'_{k-1}\}$, where $e'_{i_l} = (v_{i_l}, v_{i_{l+1}})$, $0 \leq l \leq k$ (indices modulo $k$). Let three mutually visible vertices $v_i, v_j, v_k$ of $P$, ordered according to the boundary of the polygon, be given. If the vertices $v_{j+1}, v_{j+2}, \ldots, v_{k-1}$ are not visible to $v_i$, then we call the subpolygon induced by $\{v_j, v_{j+1}, \ldots, v_{k-1}, v_k\}$ a *pocket* of $P$ with respect to $v_i$. A *triangulation* of a polygon $P$ is a partition of $P$ into non-overlapping triangles (besides their edges and vertices) whose union is $P$. These triangles may have vertices only at the vertices of $P$ and are therefore specified by edges and diagonals of $P$ in this case.

![Figure 7.1: The shaded area in the figure depicts the pocket with respect to $v_0$ behind the diagonal $(v_2, v_6)$.](image)

Let $\mathcal{P}_n$ be the set of all simply-connected polygons on $n$ vertices in the Euclidean plane, and let $\mathcal{P}$ be the set of all simply-connected polygons, i.e., $\mathcal{P} = \bigcup_{n \geq 3} \mathcal{P}_n$.

The *visibility graph* of a polygon $P$ is the (labelled) graph $G = (V, E)$ with $V = V(P)$, where two vertices are adjacent iff they are (mutually) visible in $P$. Due to the special labelling of the polygon, the cycle $v_0, v_1, \ldots, v_{n-1}, v_0$ represents the boundary of $P$. The *vertex-edge visibility graph* is a bipartite graph $G = (\hat{V}', \hat{E}')$ with
7.2. Definitions and Models

\[ V' = V(P) \sqcup E(P) \] where two vertices \( v \in V(P) \) and \( e \in E(P) \) are adjacent in \( G \) iff \( v \) is visible to at least a point of the edge \( e \) in \( P \) (considered as an open segment).

![Visibility Graph](image)

**Figure 7.2:** The figure depicts the visibility graph (left) and the vertex-edge visibility graph (right) of the polygon of Figure 7.1.

### 7.2.2 Sensing Model

We consider robot systems with a simple model of “visual” sensing. We refer to the model that we describe as the **combinatorial sensing model**. Whenever the robot is standing still at \( p \), the sensory input of the robot is a **combinatorial visibility vector** \( \text{cvv}(p) \) (or visibility vector, for short) and a **point identification vector** \( \text{piv}(p) \). As long as the robot moves, it cannot sense. The \( \text{cvv} \) of a vertex is a cyclically ordered vector of zeroes and ones. This vector is defined by the vertices of the polygonal environment that are visible from the point \( p \), and the binary bits encode whether or not the consecutive vertices form an edge of the polygon or not, i.e., they form a diagonal. As mentioned earlier, we assume the visible vertices are always given in a consistent, say, counterclockwise, order around the robot’s position. Thus, the visibility vector tells the robot the number of vertices visible to it, and a cyclic order of the **edge types** (edge or diagonal) defined by consecutive vertices. The edge types are also called 1-edges or 0-edges, respectively. Occasionally, we replace for brevity a sequence of 1s in the \( \text{cvv} \) of a vertex by the number of consecutive ones. For instance \((1, 1, 1, 0, 1, 1)\) becomes \((3, 0, 2)\). Figure 7.3 shows an example, with the combinatorial visibility vector of vertex \( v \). The consistent ordering enables us to speak about the **right** or **left** neighbor of a vertex \( v \).
which correspond to the first (not including \( v \)) and to the last visible vertex from \( v \), respectively. We abuse the notation by denoting by \( u \in \text{cvv}(v) \) a vertex \( u \) visible from \( v \). Analogously, we say that \( u \) is the \( i \)th vertex of the \( \text{cvv}(v) \), if \( u \) is the \( i \)th vertex in the cyclical ordering of the visible vertices from \( v \).

The piv of a vertex \( p \) characterizes the potential extra information (e.g., the visual properties) of the vertices that are visible from \( p \) (including \( p \)). The visible vertices are ordered as they are in the \( \text{cvv} \). In our model, a robot can see whether a visible vertex is unoccupied, or has another robot sitting on it, or a pebble (of a specific color), or a combination of these. We encode this accordingly with help of the set \( \mathcal{V} \) of \textit{visually distinguishable features}. The \( i \)th-component of the piv is a feasible (non-contradictory) subset of \( \mathcal{V} \). For instance, \( \mathcal{V} := \{\text{empty\_vertex}, \text{robot}, \text{pebble}_1, \text{pebble}_2, \ldots \} \), where \text{empty\_vertex} denotes a vertex with no visual characterization, \text{robot} denotes a vertex with a robot on it and \text{pebble}_i denotes a vertex with a pebble of color \( i \) on it. The elements of the set \( \mathcal{V} \) depend on the sensors of the considered robot.

![Figure 7.3: Illustration of the combinatorial visibility vector. In cyclic (counterclockwise) order, the vertices visible from \( p \) are \( p, a, j, l, c, d, h, i \), and its visibility vector is \( \text{cvv}(p) = (1, 0, 1, 0, 1, 0, 1, 1) \).](image)

We emphasize that the polygon induced by the vertices of the visibility vector is different from the classical definition of the visibility polygon \cite{52} – the former only uses the vertices of the original polygon, and is typically a strict subset of the visibility polygon. We believe that the visibility vector, despite being less informative than
the corresponding visibility polygon, is better suited for our simple sensing model: in our coordinate-free, combinatorial sensing model, there is no obvious way to represent or communicate entities other than polygon vertices or edges. Even polygon vertices and edges have no “names” or labels for the robot, and as such they can only be described in relative terms: e.g., $i^{th}$ vertex in the cyclic order of the visibility vector of vertex $q$. In this regard, our sensing model is more basic and weaker than the minimalism assumed by [118], who require presence of labeled features and distinguishable landmarks in the environment.

### 7.2.3 Robot Models

All considered robot models are extensions of a basic robot model – the *simple combinatorial robot* (or *simple robot*, for short), which provides elementary motion and vision capabilities. We model additional sensing capabilities by *devices* – in many cases, *sensors* – which can be “mounted” onto the basic robot. Any set of devices derives a new robot model from the basic robot. We define the considered devices after the introduction of the basic robot.

The *simple combinatorial robot* has very basic sensing and motion capabilities. When placed on a vertex of a polygon $P$, the robot possesses a sense of *handedness*. It can locally distinguish its immediate “left” and “right”: a robot at vertex $v_i$ distinguishes $v_{i-1}$ as the “left” neighbor and $v_{i+1}$ as the “right” neighbor.

The simple robot senses the environment as described in Section 7.2.2, i.e., via the combinatorial visibility vector and the point identification vector. The motion ability is likewise very limited. The robot picks a destination among the visible vertices and moves on a straight line – along a polygonal edge or diagonal – until the destination is reached. Thus, if the robot does not move, it stands on a vertex of $P$. A movement instruction has the form “move to the $i^{th}$ visible vertex”. During the motion no sensing is possible.

Some problems we investigate involve more robots moving in the polygon. We assume that robots are visually indistinguishable but possess distinct identifiers (IDs), which can be learnt through communication. Different models of communication are conceivable, such as *point* communication, *(limited) line-of-sight* communication or *global* communication. The point communication allows
two robots to communicate only if they are sitting at a common vertex, the (limited) line-of-sight communication enables the communication between two (close enough) visible robots and, finally, the global communication enables the communication between any two robots in the polygon. Since we are interested in a minimalistic model, we adopt mainly the point communication for the algorithms. In a communication round, robots might send their IDs, motion orders (by specifying the index of the destination vertex in the cvv), content of memory, etc. Observe that the uniqueness of the IDs, enables the election of a leader and to count the number of robots among the robots sitting at a common vertex. Further we assume that messages are sent without failures or interference. The robots act in an asynchronous way: they wake up at any time and move with different speeds, but they receive messages at any time. Moving robots cannot be sensed by the other robots. However a robot can sense if another robot is steady on a visible vertex.

Our focus (mainly in the next chapter) is to study the possibility issues of the simple robots equipped with various devices. Therefore we assume that the robot has an unbounded computational power and an infinitely large memory with arbitrary precision. Occasionally, we give bounds on the required memory.

We consider some of the most common devices in robotics.

**Pebble** A *pebble* is a device that is used for marking the vertices. A robot can drop the pebble at the vertex of the robot’s position, and can recollect it again for further use. If the vertex with the pebble is visible to the robot’s position, the robot sees the vertex as marked, and distinguishes this from the other vertices. The most important implication of this is the following. Suppose that the robot is on vertex \( v_i \), leaves the pebble at this vertex and moves to vertex \( v_j \). There, at vertex \( v_j \), the robot can determine the relative position of \( v_i \) in the robot’s cvv (see Lemma 8.8).

**Angle-Measuring Device, Length-Measuring Device** Suppose that a robot is on vertex \( v_i \) and that two vertices \( v_j \) and \( v_k \) are visible from \( v_i \). Let us assume, without loss of generality, that the positions of \( v_j \) and \( v_k \) in the cvv of \( v_i \) are \( j' \) and \( k' \), respectively, \( j' < k' \). Upon a request for the angle between the visible vertices \( j' \) and \( k' \) of the cvv, the angle-measuring device returns the exact angle \( \angle v_j v_i v_k \), i.e. the angle that “lies” entirely in-
side the polygon. Analogously, the length-measuring device measures the distance between the robot’s current location and a chosen visible vertex (identified by a position in the cvv).

**Reflexity Detector** The reflexity detector is similar (but weaker) to the angle-measuring device but instead of providing the size of the angle it merely decides whether the angle is convex or reflex, i.e., whether the angle is smaller, respectively bigger, than 180 degrees. We call this property the type of the vertex (or of an angle, in general).

**Compass** The north-direction is a consistent reference direction in the polygon and is parallel to the \(y\)-axis in the coordinate system of the polygon. The compass enables the robot to measure the angle formed by the north-direction, the robot’s current location and a chosen visible vertex.

**Oracle** An oracle \(\Omega\) is a (possibly non-deterministic) “black-box” device that can answer pre-specified questions posed by the robot. An oracle has perfect knowledge of the universe (the polygon, the robot’s position, history of movements, etc.). Note that the other devices can all be seen as oracles.

The labelling oracle \(\Omega_l\) answers the following type of questions. The query to the oracle is a component \(i\) of the cvv of \(v\). If the component is 1 then the oracle reports the (global) label \(k\) of the corresponding edge \(e_k\), i.e., the edge between the \((i-1)\)st and \(i\)th visible vertex from \(v\). If the component is 0, then the oracle reports the label of the edge that is partially visible behind the corresponding diagonal. This is the first edge that is intersected by the line that emanates from the vertex \(v\) and passes through the \((i-1)\)st visible vertex of \(v\) (see Figure 7.4). Note that the line that emanates from \(v\) and passes through the \(i\)th visible vertex intersects the same edge.

We investigate some interesting combination of devices. In the following table we name the considered robot models and the devices that specifies them.

<table>
<thead>
<tr>
<th>Explorer</th>
<th>Pebble(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflexer</td>
<td>Pebble(s), reflexity detector</td>
</tr>
<tr>
<td>Labeller</td>
<td>Pebble(s), labelling oracle (\Omega_l)</td>
</tr>
<tr>
<td>Surveyor</td>
<td>Pebble(s), angle-measuring device</td>
</tr>
</tbody>
</table>
7.2.4 A Robot Hierarchy

The set of problems that we are interested in is defined by the universe of mappings \( f : \mathcal{P} \times \mathcal{I} \rightarrow \mathcal{O} \), where \( \mathcal{I} \) denotes the set of inputs (besides the polygon) and \( \mathcal{O} \) the set of outputs, respectively. In some cases, we allow the polygons to be multiply-connected. Instead of using a more classical Turing machine as a computational model to solve the aforementioned problems, we use robots. An instance of a problem \( f \) is defined by a pair \((P, I)\), where \( P \in \mathcal{P} \) and \( I \in \mathcal{I} \). Given a problem \( f \), a robot is said to be able to solve \( f \) if, given any instance \((P, I)\) of the problem \( f \), a robot \( R \) that moves according to its deterministic specifications in the polygon \( P \) returns a solution \( O \in \mathcal{O} \) which (1) is independent of the vertex where the robot is initially located, (2) fulfills \( O = f(P, I) \), and (3) needs a finite number of movements of the robot. For example, consider the problem of determining the Euclidean shortest path between two vertices \( v_s \) and \( v_t \) of a polygon \( P \) (on \( n \) vertices), which lies entirely inside \( P \). We set \( \mathcal{I} = \{(v_i, v_j)\mid (i, j) \in \mathbb{N}^2\} \), which represents the choice of the vertices \( v_s \) and \( v_t \), \( \mathcal{O} = \bigcup_{P \in \mathcal{P}} \{\text{set of all polygonal paths in } P\} \) and \( f \) is the map that given a polygon \( P \) and two vertices of \( P \) returns a shortest path inside \( P \) between those two vertices.

In this thesis we consider a robot as a computational model for problems in polygons. Robots are characterized by the given features and limitations, which define a robot model\(^3\) – we denote by \( \mathcal{R} \) the

---

\(^1\)Reflexer is the robot C of the previous section.

\(^2\)Using colors to label edges, we see that labeller is precisely the robot B of the previous section.

\(^3\)When this is not misleading, we often do not distinguish the concepts of a robot
set of all robot models that operate in polygons. For a robot model \( R \in \mathcal{R} \) we want to identify the set of problems that a robot can solve. Accordingly we define a partial order on the set of robot models. We say that a robot \( R_1 \) is at least as strong as the robot \( R_2 \) – denoted \( R_2 \preceq R_1 \) – if \( R_1 \) can solve all the problems that \( R_2 \) can solve. We say that \( R_1 \) is strictly stronger (\( R_2 \prec R_1 \)) if additionally there is a problem that \( R_1 \) can solve but \( R_2 \) cannot. \( R_1 \) and \( R_2 \) are equivalent, if \( R_1 \preceq R_2 \) and \( R_2 \preceq R_1 \).

7.2.5 Measure of Complexity

Movements are costly for robots, mainly in terms of energy. Therefore we analyze some of the algorithms with respect to the total number of movements from a vertex to another necessary to terminate an algorithm in a polygon with \( n \) vertices. The worst-case analysis is performed with respect to the polygonal shape, the starting position of the robots and their identifiers. We refer to this measure of complexity as the number of steps of an algorithm.

However, we point out that the principal goal of this part is to identify the problems that can be solved in the combinatorial sensing model and not to optimize the algorithm efficiency. For this reason, the number of steps of the algorithms are stated only for the sake of completeness.

7.3 Related Work

Combinatorial geometric reasoning is key to many motion planning and exploration tasks in robotics [72, 73, 124]. Our work is similar in spirit to that of [118, 124], in that we aim to explore the power and limitations of a minimal model of robot systems. Our model is different from the one in [118] because they assume that important features of the environment are uniquely labeled, allowing sensors to distinguish these landmarks. The combinatorial sensing model is similar to the one used in [124] but the nature of problems investigated in our paper is quite different from those studied in [55, 110, 124]: their main focus is navigation and pursuit evasion, while we are concerned with the geometric and topological structure of the environment and a robot model.
collaborative and distributed self-deployment, which do not seem to have been addressed in the past.

In the literature there have been other efforts to introduce a hierarchical description of robot models – see for example [96], which summarizes some of the recent work. The present analysis differentiates from the previous work in two main aspects. Firstly, instead of considering the universe of robots we focus on a specific class of robots. Each robot that we consider is derived from a single basic robot by equipping it with new devices of different sophistication. Such a restriction allows more precise and formal reasoning. Further, along with a hierarchy which represents the relative strengths of the robots, we study the absolute strengths of the robots. To do so, we try to understand which information about the environment can be extracted by each robot.

The idea of understanding and comparing robots’ sensor equipment and the underlying strength of resulting robots is not new. Donald [32] defines and studies among the firsts general concepts of hierarchies among robots. His work deals, analogously to ours, with the problem of determining the information requirement of a robot to solve a task. He also introduces a general concept for comparing different robot models. In our work we consider a particular abstract robot model that enables us to clearly and formally argue about relative strengths of robots equipped with certain sensors. In the considered model we also apply results of computational geometry and get to the information-theoretical core of sensors.

7.4 Useful Facts

Fact 7.1 ([30]). Every polygon of size $n$ admits a triangulation. Every triangulation of a simply-connected polygon is composed by $n - 2$ triangles and $n - 3$ diagonals.

Fact 7.2 ([30]). The triangulation of a simply-connected polygon is an outerplanar graph. As such the vertices of the polygon admit a coloring with three colors in such a way that every triangle of the triangulation has all its vertices of different colors.
Chapter 8

Combinatorial Structures of a Polygon

One geometry cannot be more true than another; it can only be more convenient. Geometry is not true, it is advantageous.
– Robert M. Pirsig (born September 6, 1928) American writer and philosopher.

8.1 Introduction and Motivation

We introduce the topic of the chapter with an example. Suppose to be a tourist that is visiting a beautiful city for the first time – say Paris. You leave the hotel early in the morning and you walk around the city for visiting the most important places in Paris: the Tour Eiffel, the Louvre Museum, the cathedral Notre Dame and so on. In the evening, you reach back the hotel to have some rest. Naturally, a person that is located in a new environment – like a city – builds a map of that environment. The map enables the tourist to establish the current position and to move between different locations, for instance to find the hotel again. Studies have shown that women and men typically create this map in very different ways: women tend to remember specific locations – landmarks – and maintain an orientation among these. On the other hand, men have a stronger sense of direction and lengths: they maintain an orientation by approximating a Cartesian coordinate.
system \cite{59}. This shows that different skills create different representations of the same environment – some representations might be better than others for certain tasks. In this chapter, we ask the question of how a robot builds a representation of the polygon it is placed in. Clearly, the representation depends on the sensors and devices it is equipped with. We would like to characterize, where possible, the information that a robot can extract from a polygon – its map and to compare different maps.

All robots of our interest have limitations in the motion and sensing capabilities, but no limits on their computational power or memory size. In that respect an ability of a robot to solve a problem is closely related to the ability of building a good representation of the environment. To characterize the map of a polygon, we compare the robot models with the so called delphic robot\footnote{Delphi is known for the oracle at the sanctuary that became dedicated to Apollo during the classical period.} according to the aforementioned robot hierarchy. The delphic robot has the conceptually unbounded computational unit and the memory of the simple combinatorial robot but it can neither sense the environment nor move. For enabling the comparison we characterize the delphic robot by means of a specific oracle $\Omega$ it is equipped with. The nature of the oracle will help us understand the type of information that the compared robot can extract from the polygon.

In many cases we are able to precisely characterize the map that a robot is able to build. Surprisingly, robots equipped with significantly different sensors eventually produce the same representation of the polygon they operate in.

### 8.2 Summary of Results

In this chapter we prove both intuitive and counterintuitive results. We show that the simple robot is very limited, for instance it cannot compute the visibility graph of a polygon nor establish its size in general. Equipping a simple robot with a pebble enriches essentially the information that a robot can derive from the map. More precisely, the explorer can compute the visibility graph even of a multiply-connected polygon. On the negative side, it cannot decide whether an angle specified by three polygonal vertices is convex or reflex, or...
which boundary component of a multiply-connected polygon is the outer one. The former problem is easy for the reflexer, because its reflexity detector is designed exactly for this task. As it turns out, the labeller can decide the type of an angle as well because we will show that it is equivalent to the reflexer, i.e., they build the same map. We will exploit this equivalence to show that both robot models can determine the Euclidean shortest path between two vertices of a polygon. To show this we delineate a relationship between a shortest path \( L \) and the type of the angles induced by \( L \). This result also implies that the explorer cannot compute a Euclidean shortest path in general. Finally we show that the surveyor can build a very accurate map of the polygon.

The results of this chapter have been established in a joint work with Jan Brunner, Matúš Mihalák, Subhash Suri and Peter Widmayer [116, 20, 19].

8.3 The Map of a Polygon

8.3.1 The Case of Simply-Connected Polygons

The natural candidate for the map of a polygon is its visibility graph. Suppose that a simple robot (1) has access to the visibility graph of the polygon, (2) labels successively the vertices in the visibility graph according to the boundary of the polygon (i.e., the Hamiltonian cycle that corresponds to the polygonal boundary induces a natural ordering) and (3) knows its current position in the polygon with respect to the visibility graph. Under these conditions, the robot operates with labelled features as in [118]. This is possible, because the query “What is the label of the \( i \)th visible vertex?” can be easily answered. The consequences of this fact are manifold. For instance, it becomes easy to navigate from any given vertex to another. If the robot needs to move from vertex \( v \) to vertex \( w \) in a polygon \( P \), it computes a path \( v = v_{i_1}, \ldots, v_{i_k} = w \) between the corresponding nodes in the visibility graph of \( P \). The path at hand can now be encoded as follows: while sitting at the vertex \( v_{i_l} \), the vertex \( v_{i_{l+1}} \) is the \( j_l \)th visible vertex. Hence the robot is able to execute the corresponding instruction and eventually reach \( w \).

We note that the only information given by the cvv of the ver-
tices of a polygon around the boundary does not enable the construction of the corresponding visibility graph. More precisely, there are two polygons with different visibility graphs defined by the vertices \(v_0, \ldots, v_{n-1}\) and \(w_0, \ldots, w_{n-1}\), respectively, with the property that \(\text{cvv}(v_i) = \text{cvv}(w_i)\) for every \(i\) (see Figure 8.1).

![Figure 8.1](image-url)

**Figure 8.1:** The figure depicts two symmetric polygons with identical sequence of \(\text{cvv}\)'s around the boundary. The sets of vertices visible to the vertex \(v_0\) (marked by a cross in the two polygons) witness that the two visibility graphs are different. The dotted lines are drawn to ease the recognition of what is visible and what is not.

This observation alone does not imply that a simple robot cannot build the visibility graph of a polygon in a more involved manner than by just walking around the boundary. Nevertheless, we can prove that already the easier problem of counting the number of vertices is already non-solvable in general.

Suppose to have a problem \(f : P \times I \to O\). The general technique to prove the impossibility for a robot model to solve the problem \(f\) is the following. We exhibit two polygons \(P_1, P_2 \in P\) and two inputs \(I_1, I_2 \in I\) with the property that (1) \(f(P_1, I_1) \neq f(P_2, I_2)\) and (2) the robot (when initially placed on some proper vertices of the polygons) cannot resolve with its equipment in a finite number of steps whether it is placed in \(P_1\) with input \(I_1\) or in \(P_2\) with input \(I_2\). Hence the problem cannot be solved in this case.

**Theorem 8.1.** The simple robot cannot determine the size of a polygon in general.

**Proof.** Consider the two polygons depicted in Figure 8.2. The poly-
The Map of a Polygon

gon $P_1$ is only partially drawn. We first informally argue that $P_1$ can be extended to an arbitrary size with the property that vertices with \( \text{cvv} \ (1,0,0,1) \) and \( (1,1,0,1,0,1,1) \) alternate along the boundary (on both sides, with the exception of few vertices at the sides). $P_1$ is constituted by spikes that alternate on two sides. The tip of a spike (like the vertex $v_2$ in the figure) sees only its two neighbors and the extremity of an opposite spike (like the vertex $v_1$). A vertex which is not the tip of a spike sees only the vertices of the two spikes it is adjacent to and the vertices of the opposite spike. We show how to add a new spike (that is two new vertices, one of which is a tip) to make $P_1$ larger in such a way that more vertices with the given cvv’s alternate. Without loss of generality, we place a spike on the upper side of the polygon (as depicted in Figure 8.2). The vertices are placed along some equally spaced vertical lines. First, we make sure that the extremity of the new spike does not see the extremities of the spikes it is not supposed to see. We do this by placing the vertex on the appropriate vertical line, above the hitting point of the line $l_1$. Further we need to hide the tip of the last opposite spike. To do so, we consider the extension of the left edge of that tip (line $l_2$). If we place the extremity below the hitting point with the vertical line, the corresponding tip is hidden. In other words, if the hitting point of the line $l_1$ with the vertical line is below the hitting point of the line $l_2$, we can place the new vertex in-between and the visibility constraints are fulfilled (see the cross in the picture). We are left with placing the tip of the new spike. We set it on the vertical line where the right extremity of the spike lies. It has to be high enough to guarantee that the line $l_2$ drawn for a next spike intersects the corresponding line $l_1$ appropriately. It is straightforward to check that the tip sees only the claimed vertices.

If $P_1$ is made large enough, then its middle part, i.e., the portion of the polygon specified by all the vertices except for those with maximal and minimal $x$-coordinate, “looks like” the polygon $P_2$. Let us make this precise. $P_2$ and the middle part of $P_1$ are composed by vertices of two different cvv’s: \( c_1 = (1,0,0,1) \) and \( c_2 = (1,1,0,1,0,1,1) \). Moreover, take two vertices $v \in V(P_1)$ and $w \in V(P_2)$ with the same cvv. If we move to the $i$th visible vertex in both cases, we get to two vertices $v', w'$, respectively, such that again \( \text{cvv}(v') = \text{cvv}(w') \) holds. Provided that $P_1$ is large enough and $v$ is chosen “in the middle” of $P_1$, this observation can be repeated by a desired number of times.
Suppose that there is an algorithm $A$ for a simple combinatorial robot to count the vertices of a polygon. Let the robot sitting on a vertex of type $c_i, i = 1, 2$, of $P_2$ execute $A$. The robot will report 10 after moving a finite number of times – say $t$ times. We now construct $P_1$ with a number of vertices that depends on $t$. By the above argument, if we place the robot on a middle vertex of type $c_i$ of $P_1$, it will visit the same sequence of vertices as in the execution in $P_2$, because it makes decisions only according to the sequence of cvv’s of the visited vertices and $P_1$ is large enough. Thus it wrongly reports 10 after exactly $t$ steps – this is a contradiction and establishes the claim.

\[\blacksquare\]

**Figure 8.2:** $P_1$ and $P_2$ are the polygons used in the proof of Theorem 8.1

**Corollary 8.2.** In general, the simple robot cannot build the visibility graph of a polygon.

Non-surprisingly, the explorer can build the visibility graph of an unknown polygon using a pebble. Hence the explorer is strictly stronger than the simple robot. We now describe the algorithm that allows the explorer to build the visibility graph of a polygon $P$. The algorithm labels the nodes of the visibility graph in such a way, that the cycle $v_0, v_1, \ldots, v_{n-1}, v_0$ corresponds to the boundary of $P$ and $v_0$ is the node that corresponds to the vertex where the robot is initially on.
8.3. The Map of a Polygon

**BUILD THE MAP**

1. Start at any vertex and set $i := 0$. Leave the pebble on the current vertex and walk ccw along the boundary until the vertex with the pebble on it is reached. This established the size $n$ of the polygon. Add the visited vertices $v_0, \ldots, v_{n-1}$ to the visibility graph.

2. The current vertex is $v_i$. Set $j := i$.

3. Leave the pebble on the current vertex and walk ccw along the boundary. For every visited vertex, set $j := j + 1 \mod n$ and check:

   (a) If the pebble is on the current vertex and if $i < n - 1$, set $i := i + 1$, collect the pebble, move to right neighbor and repeat from point 2. If $i = n - 1$ stop.

   (b) If the pebble is visible from the current vertex, add the edge $v_i v_j$ to the visibility graph.

**Lemma 8.3.** The algorithm **BUILD THE MAP** can be executed by the explorer and returns the visibility graph of a polygon in $O(n^2)$ steps.

**Proof.** The feasibility and correctness of the algorithm follow immediately.

**Theorem 8.4.** The explorer is equivalent to the delphic robot with the oracle that returns the visibility graph of the polygon.

**Proof.** Lemma 8.3 shows that the explorer can build the visibility graph of a polygon and hence it is at least as strong as the delphic robot with the described oracle. For the other direction, note that the cvv of every vertex can be read from the visibility graph, and also the information about a placed pebble (since the visibility graph is labelled) and thus any information gathered by the explorer can also be obtained (by a simulation) by the delphic robot with the described oracle.

The visibility graph encodes much information about a polygon, yet some seemingly easy problems cannot be solved only by looking at the visibility graph. Consequently, by Theorem 8.4 neither the explorer can solve such problems. One such example is to resolve
whether the angle defined by three consecutive vertices \( v_{i-1}, v_i, v_{i+1} \) is convex or reflex, i.e., if it is smaller than 180 degrees or not. To see this consider the symmetric polygon shown in Figure 8.3: the visibility vectors of the pairs of vertices \( \{v_0, v_2\} \) and \( \{v_1, v_3\} \) are identical. Suppose that an algorithm for deciding the type of an angle of a given vertex does exist, and apply it to the vertex \( v_0 \), specified in some way to the robot (for instance by a pebble of distinguished color). Hence, the robot starts the algorithm execution from some vertex \( v \in \{v_0, v_1, v_2, v_3\} \) of the polygon and concludes correctly that \( v_0 \) is reflex. The decision is established by looking at the sequence of cvv and piv generated during the algorithm execution. We now ask the robot to determine the type of \( v_2 \), which is marked as was \( v_0 \) before. By letting the robot start at the “twin” vertex of vertex \( v \) (i.e. at \( v_2 \) if \( v = v_0 \), or at \( v_3 \) if \( v = v_1 \), or vice versa), it follows inductively that the sequence of cvv and piv generated is the same as before. This time the robot concludes that \( v_2 \) is reflex as well, which is false.

Figure 8.3: An example showing that the explorer in our model cannot resolve whether a vertex is convex or not.

This may seem surprising in light of the fact that even the simple robot can decide whether the entire polygon is convex or not. Observe that a necessary condition for the polygon to be convex is that the visibility vector of any vertex \( v \) must be all 1’s – the presence of a 0 bit implies that there exists a pocket in the polygon not seen by \( v \) and thus the polygon is not convex. If the visibility vectors of all vertices are all 1’s, then the polygon is convex – this follows because every non-convex polygon must contain a reflex vertex, and if \( r \) is a reflex vertex then the visibility vector of either neighbor of \( r \) cannot be all 1’s. Thus, an algorithm to decide the convexity of a polygon \( P \) is the following.
8.3. The Map of a Polygon

**Check convexity**

1. Start at any vertex, say, $v_0$, and consider the visibility vector $cvv(v_0)$. If this vector is not all 1’s, stop – the polygon is ostensibly non-convex. Otherwise, compute the size of the polygon $n$ by counting the number of 1’s in $cvv(v_0)$.

2. Move ccw along the boundary to the remaining $n - 1$ vertices. If and only if the visibility vectors of all the vertices are all 1’s, the polygon is convex.

**Theorem 8.5.** A simple robot can decide whether a given polygon is convex or not in $O(n)$ steps. However, it is not possible in general for the (stronger) explorer to decide if a given vertex is convex in a non-convex polygon.

It is evident that additional devices provided to define the other robot models enrich the information that such a robot can gather from a polygon. One can prove the following theorem in a similar way as Theorem 8.4.

**Theorem 8.6.** The labeller is equivalent to the delphic robot with the oracle that returns the visibility graph and the vertex-edge visibility graph of the polygon.

[97] shows that if no three vertices of the polygon are on a line, the visibility graph of a polygon can be computed from the vertex-edge visibility graph. This is not true in general otherwise.

To study the strengths of the surveyor, which seems to be a very strong robot, we introduce an oracle $\Omega_s$. Let $\mathcal{A}$ be the set of orientation-preserving similarity transformations of the plane. When queried, $\Omega_s$ returns the sequence $(v'_0, \ldots, v'_{n-1})$, where $v'_i$ is the image (i.e. the coordinates) of the vertex $v_i$ according to a mapping $m \in \mathcal{A}$, where $m$ is fixed (independent of the query) and unknown to the robot.

**Theorem 8.7.** The surveyor is equivalent to the delphic robot that is equipped with the oracle $\Omega_s$.

**Proof.** First observe that the surveyor can build the visibility graph of the polygon because it is an extension of the explorer and hence

---

2 An orientation-preserving similarity transformation is a point-to-point mapping of the plane which is a composition of scalings and rotations followed by a translation.
can easily navigate in the polygon \( P \) (move from a desired vertex to another).

Let the delphic robot have an instance of the oracle \( \Omega_s \) that uses a map \( m \in \mathcal{A} \). Consider now the surveyor. It picks a triangulation \( T \) of \( P \) and it takes an edge \( e \) of \( T \) which corresponds to an edge or a diagonal of \( P \) with endpoints \( u \) and \( v \). The robot assigns arbitrary (distinct) coordinates to \( u \) and \( v \) to start building a representation of \( P \). Further, it takes a vertex \( w \) that induces a triangle with \( e \) in \( T \). The robot can measure perfectly the angles \( \angle(u, v, w) \), \( \angle(v, w, u) \), \( \angle(w, u, v) \). This leaves exactly two points of the plane where \( w \) can be placed to represents correctly the angles of \( P \). The handedness of the robot cuts down the number of options to one. It is easy to see that by further following the triangulation \( T \) the remaining vertices of \( P \) are placed uniquely. Hence the coordinates of the representation of \( P \) of the robot are equal to the image of the true coordinates of \( P \) under a (unknown) map \( m' \in \mathcal{A} \). Note that this map is fully specified by the coordinates of the vertices \( u, v \). This shows that the surveyor can simulate the oracle \( \Omega_s \). For the other direction observe that the angles that the delphic robot measures in the representation generated by the oracle are equal to the respective angles measured by the surveyor.

Suppose that we enrich the capabilities of the surveyor. In addition to the angle-measuring device, we endow the surveyor with either a length-measuring device or a compass (or both). By literally repeating the previous proof, we can easily see that such a robot builds a more accurate representation of the polygon. In other words, the enriched surveyor is equivalent to the delphic robot endowed with the same oracle \( \Omega_s \), but the set of geometric transformations which may be used by the oracle is smaller – the length-measuring device prevents the oracle to pick a transformation that involves a scaling while the compass prevents rotations. Further, by again arguing with a triangulation of the polygon, it becomes evident that the length-measuring device can simulate the angle-measuring device (since the lengths of the three sides of a triangle uniquely specify the angles). Obviously, the opposite direction does not hold.

We note that the angle-measuring device can be simulated by the compass and that the pebble can be simulated by the combination of the compass and the length-measuring device. More precisely, we shall say that a robot equipped with the compass or with the com-
pass and the length-measuring device is at least as strong as a robot
equipped with the angle-measuring device or the pebble, respectively.
To see the latter, notice that a robot equipped with the compass and
the length-measuring device can simulate a polar coordinate system
and hence give coordinates to the vertices that coincide with the true
coordinates up to a translation. If coordinates are given, the robot
can operate with labelled features and hence a pebble can be easily
simulated.

8.3.2 The Case of Multiply-Connected Polygons

Next, we consider another fundamental task related to the geometry
of a polygon: is the workspace of the robot simply-connected, or does
it have holes? If the polygon is multiply-connected, then how many
holes does it have? We show that the explorer can build the visibil-
ity graph of a multiply-connected polygon along with the information
about which vertices lie on a common (polygonal or hole) boundary.
However, the explorer is not able to decide which vertices belong to
the polygonal boundary and which vertices belong to the hole bound-
daries. Non-surprisingly, the simple robot is not capable of resolving
in general whether a polygon is simple or not.

Before we describe how the explorer can figure out the visibility
graph of a polygon, we point out a very basic difficulty that a simple
robot has (even in simple polygons).

Lemma 8.8. In general if a simple robot, initially placed on the ver-
tex \( v \), moves to vertex \( w \), it cannot recover the position of the vertex
\( v \) in the \( cvv \) of vertex \( w \). It can recover this position if the robot has
a pebble of a new color with respect to the pebbles already present in
the polygon.

Proof. Consider Figure 8.4 and suppose that a robot without pebbles
can always figure out the origin of a movement from its destination.
Note that for the depicted polygon, we have that \( piv(v_1) = piv(v_2) \)
and \( cvv(v_1) = cvv(v_2) = (1, 0, 0, 1) \). Consider a robot without peb-
bles that moves from \( v_1 \) to \( w \) in one instance and from \( v_2 \) to \( w \) in a
second instance. In both cases it starts with the same visual informa-
tion at the origin and has the same visual information at the destina-
tion, since it does not have a pebble to influence the piv of \( w \). Hence
it concludes that the origin vertices are the same in both cases, which
is false.
On the other hand, if the robot has a pebble that uniquely marks the origin vertex, the task is easy.

![Diagram](image.png)

**Figure 8.4:** A robot cannot recognize the origin of a movement from its destination.

A general concern for the explorer is that it might need to use its pebble to perform a movement that needs to be backtracked while the pebble has been placed somewhere else for another purpose. In the case of simply-connected polygons, we have already observed (Theorem 8.4) that the explorer can build the visibility graph. Since it cannot solve any other problem that cannot be assessed by the visibility graph, it is not faced to the problem of using a single pebble for two different purposes. In multiply-connected polygons though, we encounter this issue to establish the visibility graph. Hence, we use the following easy lemma.

**Lemma 8.9.** Let the explorer be on a vertex $v$ of a (multiply-connected) polygon $P$. Suppose that it has previously placed its pebble on the vertex $w$. The explorer can move to some vertex $u$ of $P$ in such a way, that it can recover the position of the vertex $v$ from the $cvv$ of $u$ and possibly backtrack to $v$. At the end of the operation the pebble is on $w$ again.

**Proof.** Let $v_{i_1}, \ldots, v_{i_r}$ be the sequence of vertices visited by the robot since it dropped the pebble at $v_{i_1} = w$ to its current position $v_{i_r} = v$. We prove the claim by induction on $r$. If $r = 1$ the claim is trivial as $v = w$. Hence the robot recognizes $v$ from $u$ by the presence of the pebble.
Suppose that the claim holds for $r - 1$. By induction hypothesis,
the robot can backtrack every single step done and move to \( w \) via
\( v_{i_r-1}, v_{i_r-2}, \ldots, v_{i_1} \). From that position, it recollects the pebble, moves back to \( v \) and drops the pebble to \( v \). Using the pebble moves to \( u \), recovers the position of the pebble and moves back to \( v \). Finally it recollects the pebble and it places it back at \( w \). At this point it is free to move to \( u \) and has enough information to backtrack to \( v \) without the pebble.

Lemma 8.9 states that the explorer is able to move inside of a multiply-connected polygon without worrying about using the pebble for other purposes other than backtracking. As a corollary, the robot can use its memory to move from its current position to any vertex it has encountered in the past. Hence, from now on we shall simply say “the robot moves back to a previous position” without arguing how. Note that it is easier to recollect the pebble in case of a simply-connected polygon: it suffices to move along the boundary and count the number of steps performed between the current position and the vertex with a pebble on it. By reverting the motion direction and counting the same amount of vertices, the robot easily returns to the starting position.

In our task to establish the visibility graph of a multiply-connected polygon, we begin with a simple geometric fact that plays an important role in our algorithm. The hole graph of a polygon \( P \) is defined as follows: the boundary components of the polygon are the vertices of the graph and two vertices are connected by an edge, if and only if there are two mutually visible vertices of the polygon in the corresponding boundary components.

**Lemma 8.10.** For any multiply-connected polygon, the corresponding hole graph is connected.

*Proof.* The proof follows from the fact that any polygonal domain admits a triangulation (using only polygonal edges and diagonals that connect vertices). Such a triangulation is a connected graph. By contracting each boundary component to a single node and identifying all the diagonals between the same boundary components, the connectivity is preserved and we obtain the hole graph.

The previous lemma states in particular, that for any boundary component \( C \) of a multiply-connected polygon \( P \), there must be a
vertex $u \in C$ whose visibility vector includes a vertex from a different component of $P$.

Intuitively, the task at hand is the following. The robot has to assign labels to vertices in such a way, that when the robot is placed at a vertex, it is able to figure out the labels of the visible vertices. To do so, it places the pebble on a visible vertex $u$. Lemma 8.9 ensures that the robot can navigate through the boundary components that it already knows. If the pebble is on one of such boundaries, it is easy to figure out the label of the vertex $u$ by backtracking and checking if a visited vertex has a pebble on it. Otherwise a new label can be assigned. If we repeat this procedure for every vertex of the boundary component that the robot is currently inspecting, we eventually find a new boundary component (if the polygon is multiply-connected). By repeating this procedure for the vertices of every boundary component encountered so far, we will complete the visibility graph thanks to Lemma 8.10. The label of a vertex $u$ is of the form $(l, k)$, where $l$ is an index that denotes a boundary component of the polygon and $k$ denotes the distance ccw along the boundary component between $u$ and a distinguished vertex of the boundary component, e.g., the first vertex of a boundary component visited. The following pseudo-code describes the algorithm for explorer.

**BUILD THE MAP**

1. Start on an arbitrary boundary component and compute its size. Let $n_1$ be the size of the current component. Set the component counter $i := 1$.

2. Give the labels $(1, 1), \ldots, (1, n_1)$ to the vertices of the current boundary component and add them to the visibility graph. Add the vertices of the current component to the queue $Q$.

3. As long as the queue is not empty, pick a vertex $v$ in the queue and move to $v$. For every vertex $u$ in $cvv(v)$ move the pebble to $u$ and proceed with 4.

4. Move to every vertex $(j, 1), 1 \leq j < i$ and check if the pebble is on the corresponding boundary component $j$ by moving for $n_j$ steps around the boundary component.

   (a) If the pebble is found for some $j$, compute the label of $u$ and add the corresponding edge between $u$ and $v$ to the visibility graph.
8.3. The Map of a Polygon

(b) If the pebble is not found, move to \( u \), set \( i := i + 1 \) and determine the size \( n_i \) of the boundary component. Label the vertices of the boundary component by \( (i, 1), \ldots, (i, n_i) \). Add the vertices to the queue and to the visibility graph. Add the corresponding edge between \( u \) and \( v \) to the visibility graph.

5. When all the vertices visible from \( v \) have been settled, remove \( v \) from the queue and repeat from point 3.

The processing time of the algorithm can be improved in several ways, though that is not our main intent here.

Theorem 8.11. The explorer can compute the visibility graph of a multiply-connected polygon. Further it can partition the vertices according to the boundary component they belong to.

Proof. First remark that explorer can execute the algorithm BUILD THE MAP 2 due to Lemma 8.9. By construction of the labels, the nodes of the visibility graph are partitioned according to the boundary component they belong to. The graph \( G \) returned by the algorithm is certainly a subgraph of the visibility graph. We now argue that \( G \) is the complete visibility graph. Due to the point 3. of the algorithm, \( G \) is an induced subgraph of the visibility graph. The vertices of a boundary component are added simultaneously when it is visited for this first time. So it is enough to argue that every boundary component is visited. Due to Lemma 8.10 this is trivial, because otherwise the hole graph would be disconnected.

The use of a pebble to mark a vertex in our algorithm seems critical. The next theorem states that a simple robot is in general not even able to decide whether a given polygon is simply- or multiply-connected.

Theorem 8.12. The simple robot cannot decide in general if a given polygon is simply- or multiply-connected.

Proof. Suppose that an algorithm \( A \) that resolves the topological connectedness without requiring a pebble exists. Assume that \( A \) correctly resolves after \( t \) steps that the polygon on the left in Figure 8.5 is non-simple. If we run the same algorithm from a suitable vertex
(in the middle) of the polygon depicted on the right (whose size is proportional to $t$), $A$ necessarily concludes that this polygon also is non-simple – a contradiction. In fact the two polygons yield identical visibility information at every vertex visited within $t$ steps.

Finally, as another example of a simple task that is involved in the combinatorial sensing model, we argue that even though the explorer can discover all the components (holes) in the environment, it cannot decide which one is the outer boundary component. Consider the centrally symmetric polygon shown in Figure 8.6. We observe that every vertex in this polygon has the same combinatorial visibility vector, namely, $(1, 0, 1, 1, 0, 1)$, irrespective of whether it is on the outer boundary component or the inner boundary component. Because the robot cannot sense or measure angles or distances, its world view looks the same whether it is on the outer boundary component or the inner one, implying that it cannot resolve between those two boundaries. Again, the proof exploits the fact that a robot executing an algorithm starting at a vertex of the inner boundary component generates the same sequence of $cvv$ as a robot executing the same algorithm but starting at a vertex of the outer boundary component.

**Theorem 8.13.** The explorer cannot resolve in general which is the outer boundary component of a multiply-connected polygon.

### 8.4 Comparing the Robot Models

Now that we understand how polygons “look like” to the different robots, we start comparing the different robots models. As already
8.4. Comparing the Robot Models

hinted at in the introduction, comparing robot models lead to surprising results. As already established in the previous section, we have the following result.

**Theorem 8.14.** It holds: “simple robot $\preccurlyeq$ explorer”.

The labeller is obviously at least as strong as the explorer. To see that it is strictly stronger take for instance the problem of deciding if a given vertex is convex or reflex. Theorem 8.5 shows that the explorer cannot solve this problem in general, whereas [97] shows that the vertex-edge visibility graph provides this information. Hence we have the following theorem.

**Theorem 8.15.** We have “explorer $\preccurlyeq$ labeller”.

The next result is more surprising.

**Theorem 8.16.** Labeller and reflexer are equivalent.

*Proof.* As both robots are at least as strong as the explorer, both can build the visibility graph of the polygon.

We first prove that “labeller $\preceq$ reflexer”. To do so, we show that reflexer can use its capabilities to simulate labeller’s oracle, i.e., we want to show that reflexer can identify the endpoints of every (partially) visible edge of the polygon. This is obvious if an edge $e_i$ is *totally visible*, i.e., if both the vertices $v_i$, $v_{i+1}$ are visible from the robot’s current position – without loss of generality $v_0$. In this case the reflexer sees the endpoints, too, and thus can read the labels of the vertices, and hence the label of the edge, from the visibility graph.
Thus we concentrate on the case where the reflexer sees only a portion of the edge $e_i$. The edge $e_i$ thus corresponds to a diagonal in the robot’s cvv – let $v_jv_{j'}$ be this diagonal (the robot knows the labels $j$ and $j'$ from the visibility graph). The visible part of $e_i$ is determined by the cone defined by the lines that go through $v_0, v_j$ and $v_0, v_{j'}$, respectively. See Figure 8.7 for illustration. The robot first checks whether it sees one endpoint of $e_i$, i.e., whether $v_i = v_j$ or $v_{i+1} = v_{j'}$. The case $v_i = v_j$ happens iff the vertex $v_{j+1}$ lies left of the line $v_0, v_j$, i.e., iff the angle $\angle v_0v_jv_{j+1}$ is convex. Similarly, $v_{i+1} = v_{j'}$ happens iff the vertex $v_{j'-1}$ lies right of the line $v_0, v_{j'}$, i.e., iff the angle $\angle v_{j'-1}v_{j'}v_0$ is convex. In such a case reflexer can easily identify the labels of the endpoints of $e_i$.

It remains to consider the case when the robot does not see $v_i$ and $v_{i+1}$ from $v_0$. Thus, $v_i$ lies to the right of the line $v_0, v_j$ and $v_{i+1}$ lies to the left of the line $v_0, v_{j'}$ (see Figure 8.7). Observe that this is the only visible edge between $v_j$ and $v_{j'}$ with this property. Thus, it follows trivially, $e_i$ is the only visible edge which has one endpoint to the right of $v_0, v_j$ and the second endpoint to the left of $v_0, v_{j'}$. The robot then moves to $v_j$ and checks whether it sees from $v_j$ an edge where one endpoint $v_k$ is to the right of $v_0, v_j$ (again, this can be done by deciding whether the angle $\angle v_0v_jv_k$ is reflex) and the second endpoint $v_{k+1}$ is to the left of $v_0, v_j$. If it cannot find such an edge, the edge $e_i$ (which is still partially visible from $v_j$; actually, a bigger portion is visible) forms the background of some diagonal in the robot’s cvv. It is easy to identify which diagonal it is – the only diagonal formed by vertices $v_k$ and $v_{k'}$ for which $v_k (j' \leq k \leq j)$ lies to the right of the line $v_0, v_j$ and $v_{k'} (j' \leq k' \leq j)$ lies to the left of the line $v_0, v_j$. We have been in our analysis in this situation already – the robot now checks whether it sees one endpoint of $e_i$, i.e., whether $v_i = v_k$ or $v_{i+1} = v_{k'}$. If yes, we are done, otherwise we proceed recursively: the edge $e_i$ has its endpoints on the two sides of the cone formed by the lines $v_j, v_k$ and $v_{j'}, v_{k'}$, and it is the only visible edge between $v_k$ and $v_{k'}$ with this property. Thus, the robot can move to $v_k$ and perform the whole procedure again. This recursive approach has to stop, as the distance between $k$ and $k'$ strictly decreases in every step, and eventually $v_k = v_i$.

We now prove that “reflexer $\subseteq$ labeller”. Recall that by Theorem 8.6 and [97], labeller is already able to decide the type of an angle specified by three consecutive vertices. Consider the angle
8.4. Comparing the Robot Models

![Diagram](image)

**Figure 8.7:** Partially visible edge $e_i = (v_i, v_{i+1})$ from the vertex $v_0$.

![Diagram](image)

**Figure 8.8:** The angle $\vartheta$ is convex, because $v_i$ can see an edge among $e_j, \ldots, e_{k-1}$.

$\vartheta = \angle v_i v_j v_k$, where $v_i$ and $v_k$ are visible from $v_j$, and for simplicity assume that $0 \leq i < j < k \leq n - 1$. $\vartheta$ is convex if and only if one of the edges $e_j, e_{j+1}, \ldots, e_{k-1}$ are partially visible to $v_i$ (see Figure 8.8). To see this, assume first that $k - j = 1$ and that $\vartheta$ is reflex. Then $v_i$ cannot see $e_j$ because it is hidden by $v_j$. Conversely suppose that $\vartheta$ is convex. Then by the assumption that the boundary component is non-selfintersecting, $v_i$ must be able to see a portion of $e_j$. The claim for $k - j > 1$ follows by noting that $v_i$ can see a portion of the diagonal $v_j v_k$ if and only if it can see a portion of one of the edges $e_j, e_{j+1}, \ldots, e_{k-1}$.

Finally, we want to argue that a group of explorers, or labellers, are not stronger than a single explorer, or labeller, respectively. This claim is not clear in case of a group of simple robots vs a single simple robot. If two simple robots are initially placed on a vertex, a robot can simulate the algorithms designed for an explorer and the other robot
Figure 8.9: The Rendezvous Problem is in general impossible to solve in multiply-connected polygons. Every robot (represented by a dot) has the same visibility information and perform the same movement. If the movements are chosen by an adversary to be synchronized, no two robots are going to be visible to each other.

Even though a group of robots do not solve new problems with respect to a single robot, in certain cases they are more efficient. We briefly deal with such efficiency questions in Chapter 9.

**Theorem 8.17.** If a group of explorers (labellers) can solve every instance of a problem $f$ (in any communication model), then a single explorer (labeller) can solve every instance of a problem $f$.

**Proof.** We show how a single explorer can simulate an algorithm designed for a group of explorers. The claim for the labeller is proved analogously. To fully specify an instance, we feed the single explorer with the information about the starting position of the group of explorers of the instance to simulate. As a first step, the single explorer builds the visibility graph of the polygon to enable a simple navigation and recognition of visible vertices. The algorithm of the explorers has to comply with the asynchronous model. Hence we further
By noting that an explorer can simulate the role of a pebble, it follows from the previous theorem that providing explorer with more pebbles does not help.

8.5 Equivalent Robots: An Example

O’Rourke and Streinu showed in [97] that the vertex-edge visibility graph provides enough information to determine the Euclidean shortest path between two points in a polygon (and also a shortest path tree from a given vertex). In the previous sections we have proved that the reflexer is equivalent to the labeller and that the labeller in turn is equivalent to the delphic robot that has access to the vertex-edge visibility graph. Hence we know that the reflexer can compute a shortest path between two vertices of the polygon as well. A way for the reflexer to do this is to follow the reduction to the labeller and simulate the labeller’s algorithm. This does not lead to an intuitive algorithm for the problem. In this section we solve the shortest-path problem for the reflexer directly, and present an algorithm that exploits naturally reflexer’s features – deciding the convexity of any angle induced by two visible vertices and the robot’s position.

To this end, we prove and exploit a structural theorem about paths in polygons, for which we define the following terms. A polygonal path is a path induced by a sequence $l_1, l_2, \ldots, l_k$ of points in the plane – it starts at a node $l_1$ and always connects by a straight line to
the next node of the sequence. We look only at polygonal paths that are entirely included in the polygon \( P \) and with \( \{l_1, \ldots, l_k\} \subset V(P) \). An internal angle of a polygonal path at vertex \( l_i \), \( 1 < i < k \) (like the ones depicted in Figure 8.10), is the angle between the lines \( l_il_{i-1} \) and \( l_il_{i+1} \), that lies entirely inside \( P \).

**Theorem 8.18.** Let \( P \) be a simply-connected polygon and \( v_s, v_t \in V(P) \). Then there is a unique polygonal \( v_s - v_t \)-path that turns at vertices of the polygon for which every internal angle is reflex. This path is the unique shortest \( v_s - v_t \)-path in \( P \).

![Figure 8.10: The picture depicts a polygon with a shortest \( v_s - v_t \)-path. Note that all internal angles are reflex.](image)

Lee and Preparata [74] showed that the shortest \( v_s - v_t \)-path in a polygon is unique and has the claimed property. Nevertheless, we present a new proof because it is slightly simpler and additionally shows that no \( v_s - v_t \)-path that turns at vertices of the polygon with reflex internal angles other than the Euclidean shortest path exists.

**Proof.** Figure 8.10 illustrates the situation of the theorem. It has already been proved in the literature that a shortest path between any two points in a polygon is a polygonal path that turns at vertices of the polygon [85]. Furthermore, it is not difficult to see that every shortest path has only reflex internal angles (i.e., angles bigger than 180 degrees): suppose that an internal angle \( \angle l_{i-1}l_il_{i+1} \) of a shortest (polygonal) path is convex. Then the general position assumption of the polygon implies that \( l_i \) can be moved slightly in the direction of the bisector of the angle \( \angle l_{i-1}l_il_{i+1} \) such that the new path does not cross the polygonal boundary component. Obviously the newly
created path is shorter (see Figure 8.11), which contradicts our assumption. A more careful analysis shows that the general position assumption is not necessary.

![Figure 8.11: The shortest path in a simple polygon has all its internal angles reflex.](image)

We now show that there is a unique polygonal $v_s - v_t$-path with all internal angles being reflex. This then shows that it has to be a shortest $v_s - v_t$-path.

Suppose for contradiction that we can find two distinct polygonal $v_s - v_t$-paths $L_1, L_2$ such that all their internal angles are reflex. Let $p$ be the first vertex on $L_1$ from which the two paths differ, and let $q$ be the first point on $L_1$ after $p$, where the two paths meet again (notice that $q$ does not have to be a vertex of $P$). Let $L'_1$ and $L'_2$ be the induced sub-paths of $L_1$ and $L_2$, respectively, between $p$ and $q$. Observe that $L'_1, L'_2$ induce a closed curve $C$. Observe also, that the region enclosed by $C$ is completely inside the polygon.

Let $Q$ be the polygon which is defined by the convex hull of the nodes of $C$. Note that $Q$ has at least three vertices and that all vertices other than $p$ and $q$ are vertices of $P$. The situation is depicted in Figure 8.12.

As $L'_1$ and $L'_2$ lie in $Q$, the internal angle of any vertex of $L'_1$ or $L'_2$ is at most the respective angle of the polygon $Q$. However, the angles of $Q$ are all convex, so every internal angle of a vertex of $L'_1$ and $L'_2$ must be convex. Thus there is a vertex $w$ of a path $L'_1$ or $L'_2$ not equal to $p$ and $q$ such that the internal angle of $w$ is convex. This vertex induces the same convex angle in the whole path ($L_1$ or $L_2$), which means there is a convex internal angle in one of those paths, a contradiction.

The new insight of Theorem 8.18 is that a polygonal $v_s - v_t$-path that turns only at polygonal vertices with reflex internal angles
is unique. To find a shortest \( v_s - v_t \)-path in a polygon \( P \), the reflexer can just find the unique polygonal \( v_s - v_t \)-path with only reflex internal angles. We describe an algorithm for the reflexer that builds a shortest-path tree of \( P \) rooted at \( v_s \). The robot starts at \( v_s \) and connects \( v_s \) with all visible vertices to begin the construction of the shortest-path tree \( T \). The robot then proceeds iteratively: it goes to every leaf \( l \) of \( T \) and for every visible vertex \( v \) from the leaf the robot checks whether the angle between the predecessor of the leaf and \( v \) is reflex. If yes, it connects \( v \) to \( l \) in \( T \). The following invariant follows easily from Theorem 8.18 and is maintained at any time of the algorithm: (i) The path in \( T \) from \( v_s \) to any vertex of \( T \) is the shortest path between the vertices in \( P \), (ii) and \( T \) is a tree. It is easy to see that (i) is always fulfilled (Theorem 8.18). To see that \( T \) is always a tree, we need to show that any vertex \( w \) in \( T \) is connected to at most one predecessor. This is easy to see, as if this is not the case, then there would be two different shortest paths from \( v_s \) to \( w \), a contradiction to Theorem 8.18. It is not difficult to see that at the end \( T \) contains all vertices of \( P \), and thus \( T \) is a shortest-path tree of polygon \( P \).

**Corollary 8.19.** The reflexer can compute the shortest-path tree of each vertex of a polygon.

Notice that Theorem 8.18 implies that the explorer is not capable of determining the Euclidean shortest path between two vertices in a polygon. If an algorithm for this problem would exist, then the explorer could easily exploit it to determine whether a given angle is convex or reflex. In fact, an angle \( \angle v_i v_j v_k \) is reflex if and only if the path \( v_i, v_j, v_k \) is the shortest path between \( v_i \) and \( v_k \). Recall
that the explorer cannot determine the type of an angle in general (Theorem 8.5).

**Corollary 8.20.** The explorer cannot determine in general the Euclidean shortest path between two vertices of a polygon.

The surprising fact is that to determine the Euclidean shortest path in a polygon we do not measure lengths. We remark that in multiply-connected polygons paths composed by reflex vertices are not unique in general (and have not the same length). Theorem 8.18 settles the intriguing example in Chapter 7 about robot A, robot B and robot C.

## 8.6 Open Problems

The robot model for which we fail to provide a simple characterization in terms of the map it can produce is the simple robot. We have seen that it is not able to compute the visibility graph in general. On the other hand it can easily decide if a polygon is convex or not. Is there a well-defined structure that encodes this information? One of the consequences that a result to this problem might have is that it could shed light to a problem that we call Lemma 2 for historical and fondness reasons. Informally, Lemma 2 conjectures that if a polygon displays a symmetry in the sequence of $cvv$’s of the vertices around the boundary, then the visibility graph displays the same symmetry. More precisely, suppose that a polygon $P$ has the property that $cvv(v_i) = cvv(v_{i+\kappa})$ holds for every $i = 0, \ldots, n - 1$, and a $\kappa \in \mathbb{N}$. Then the graph homomorphism $\varphi : V(P) \to V(P)$, $\varphi(v_i) = v_{i+j\cdot\kappa}$, $0 \leq i \leq n - 1$, (operations modulo $n$) operating on the visibility graph of $P$ is an automorphism for every $j \in \mathbb{N}$.

Lemma 2 has many surprising applications. For instance, suppose that a polygon $P$ satisfies the conditions of Lemma 2 for a certain $\kappa$. Then it is possible to show that there are $n/\kappa$ vertices $v_i, v_{i+\kappa}, \ldots, v_{i+(n/\kappa)-1}\kappa$, $0 \leq i \leq n - 1$, that are mutually visible and thus induce a convex subpolygon of $P$. In turn, this can be exploited to yield a positive answer to the question whether the Rendezvous Problem is solvable by a group of identical explorers. In the Rendezvous Problem, a group of explorers are placed on different vertices of the polygon. Their task is to move in some position in such a way, that eventually all explorers are visible to each other. Symmet-
ric polygons make the problem intuitively harder. On the other hand, Lemma 2 enables us to exploit this symmetry.
Chapter 9

The Art Gallery Problem

Which painting in the National Gallery would I save if there was a fire? The one nearest the door of course. – George Bernard Shaw (July 26, 1856 - November 2, 1950), Irish playwright, critic and Nobel Prize winner in literature.

9.1 Introduction and Motivation

Back in 1973, Victor Klee posed a question about visibility in polygons to Václav Chvátal, a young mathematician. The question became known as the Art Gallery Problem and has been deeply studied under many facets since then. The classical version asks about the minimum number of vertices – the guarding positions – from which it is possible to view every point in the interior of a given polygon $P$. The well-known Art Gallery Theorem [24, 42] asserts that every simple $n$-vertex polygon can be “guarded” by placing $\lceil n/3 \rceil$ guards at vertices of the polygon, and this bound is the best possible in the worst-case. The classical setting of the Art Gallery Theorem assumes full knowledge of the polygon, including vertex coordinates.

Chvátal’s solution [24], although being conceptually simple, entails a few special cases. An almost trivial but extremely beautiful proof was given by Fisk three years later [42]. The proof uses the fact that a triangulation of the polygon always exists and can be 3-colored: three colors can be assigned to vertices so that no edge of a triangu-
lation has the same color at both endpoints. Then, placing the guards at the vertices of the least frequent color solves the Art Gallery Problem: there are \( n \) vertices, so the least frequent color occurs at most \( \lfloor n/3 \rfloor \) times, and since each triangle must have vertices of all three colors, every triangle is visible to some guard. For an illustration see Figure 9.1.

![Figure 9.1](image)

**Figure 9.1:** The picture illustrates Fisk’s idea. 3 is the least used color. Since every triangle has a vertex colored with 3, two guards collectively guard the depicted polygon.

In a recent paper [45], Ganguli et al. showed that, given an unknown polygon, \( \lfloor n/2 \rfloor \) mobile guards – different from the ones studied here but also with only local views – can “self-deploy” at vertices to achieve the Art Gallery coverage. Their result raises the interesting question whether the gap between \( \lfloor n/2 \rfloor \) and \( \lfloor n/3 \rfloor \) is inherently due to the lack of global geometry. We show that this is not so, and in fact \( \lfloor n/3 \rfloor \) guards (explorers) in our combinatorial sensing model can self-deploy to guard the polygon. Intuitively, this claim results immediately by the fact that the explorer can build the visibility graph of the polygon. One can operate on the visibility graph by mimicking the original proof of Fisk [42]. To make the problem interesting, we take into consideration efficiency questions. More precisely, we would like to bound the memory storage of the robots and look at the total number of steps required to completion. We make the standard assumption that each memory cell of a robot can store a word of \( \Theta(\log n) \) bits. Unfortunately, our algorithms fail to be asymptot-
9.2 Summary of Results

The focus of this chapter is on the Art Gallery Problem. We show how a group of simple robots can compute a triangulation of a polygon. Successively, we use the triangulation to determine at most \( \lfloor n/3 \rfloor \) guarding positions, which is optimal in the worst case. The presented algorithm is executed by simple robots and requires each robot to have only \( \Theta(1) \) word memory. Further, we present an improved version that reduces the total number of steps in many instances. Finally, we argue that a group of strong labellers (or reflexers) cannot solve in general the decision version of the Art Gallery Problem.

The results of this chapter have been established in a joint work with Mattia Bergomi, Matúš Mihalák, Subhash Suri and Peter Widmayer. [116, 117, 13].

9.3 Polygon Triangulation

The triangulation algorithm In this section, we describe our algorithm for triangulating a simple polygon, which is the key step in solving the Art Gallery Problem as well as a geometric structure with broad applicability [30]. We describe our algorithm for an explorer, with the assumption that this robot has \( \Theta(n) \) memory to store the triangulation – we note that storing a visibility graph requires \( \Theta(n^2) \) memory in the worst case. Later we show that the same result can be obtained by a collaborative group of \( m \) explorers, each with \( \Theta(n/m) \) memory that communicate with the point communication model. As a corollary of this distributed triangulation, we obtain that a group of \( \lfloor n/3 \rfloor \) explorers, each with \( \Theta(1) \) memory, can collectively build the triangulation and solve the Art Gallery Problem.

The polygon triangulation algorithm is recursive in nature, and works as follows (see Figure 9.2). The explorer places a pebble at a vertex, call it \( v_0 \). The combinatorial visibility vector of \( v_0 \), \( cvv(v_0) \),

ically faster in the worst case than the algorithm BUILD THE MAP that computes the whole visibility graph (making the problem easy), however they are faster in many instances, require less memory, and try to exploit collaboration between robots.
consists of a sequence of 0-edges intermixed with 1-edges. Each 0-edge is a diagonal that separates a pocket from \( v_0 \), and the pockets defined by different 0-edges are pairwise disjoint. The explorer will \emph{recursively} triangulate the pockets formed by the 0-edges (say, by visiting them in cyclic order), and then complete the triangulation by drawing diagonals from \( v_0 \) to all the vertices in its visibility vector. Thus, in high-level pseudo-code, the triangulation algorithm can be described as follows:

**TRIANGULATION**

1. Start at an arbitrary vertex \( v_0 \). Let \( e_1, e_2, \ldots, e_k \) denote the 0-edges in the combinatorial visibility vector of \( v_0 \) (in ccw order), and let \( P_i \) denote the pocket of the polygon defined by the edge \( e_i \).

2. Compute recursively the triangulation of \( P_i \), for \( i = 1, 2, \ldots, k \).

3. Finish the triangulation by adding diagonals from \( v_0 \) to all the vertices in its combinatorial visibility vector (endpoints of both 0- and 1-edges).

![Figure 9.2: The triangulation algorithm. Starting at \( v_0 \), the algorithm triangulates the pockets \( P_1, P_2, P_3 \), in that order, and finally completes the triangulation by drawing edges from \( v_0 \) to all vertices visible from it.](image)

To better understand the structure of the algorithm, we introduce the concept of the \emph{exploration tree} \( T = T(v) \) of a polygon with respect to a vertex \( v \). The exploration tree \( T \) is recursively defined starting from the vertex \( v_0 \), which is the root of \( T \). Given a current
9.3. Polygon Triangulation

leaf \( w \) of \( T \) consider the 0-edges of the cvv of \( w \) within the pocket defined by the parent of \( w \) and the 0-edge having \( w \) as the first endpoint and order them ccw with respect to their endpoints. Add the first endpoint \( u \) of every 0-edge to the vertex-set \( V(T) \) of \( T \) and the edge \( uw \) to the edge-set \( E(T) \) (see Figure 9.3 for an example).

\[ \text{Figure 9.3: The exploration tree of the polygon } P \text{ with respect to } v. \text{ The dashed diagonals are the 0-edges with respect to the vertices of } T. \]

\( T \) represents the explorer’s navigation scheme of the algorithm (if we neglect the movements required by Lemma 9.1 see later). Consequently, we can express the total number of steps performed by the explorer by \( O(|E(T)|) \). Unfortunately, the size of \( T \) strongly depends in general on the choice of the root, as Figure 9.4 shows. From the vertex \( u \) the whole polygon is visible, hence the exploration tree consists of only one point. On the other hand, the exploration tree is a long path, if the root is taken to be \( v \) (the diagonals defining each pocket are represented by a dotted line). The same example shows that \( O(|E(T)|) \) can be as bad as \( \Theta(n) \).

Turning this high level description into a correct algorithm that fits in our combinatorial sensing model, however, requires several careful steps. We use the illustration of Figure 9.2 to explain the key steps of the algorithm.

At the top level of the recursion, the visibility vector is defined for vertex \( v_0 \). The robot consistently scans the visible edges (the elements of the cvv\((v_0)\)) in cyclic order. Let the first 0-edge in this visibility vector be \( v_iv_{i+1} \), and let \( P_i \) denote the pocket defined by this edge. The vertices \( v_i \) and \( v_{i+1} \) are identified by the robot as the \( i \)th and the \( (i+1) \)st vertices in cvv\((v_0)\). However, as the robot enters the pocket \( P_i \) for the recursive triangulation, it no longer “sees” the polygon from \( v_0 \), and needs a way to distinguish (identify) \( v_{i+1} \) from the “new base”
Figure 9.4: The size of the exploration tree strongly depends on the choice of the root.

\(v_i\) (see Lemma 8.8). Similarly, the explorer needs a way to recognize \(v_0\) from \(v_i\), where it must return after the recursion ends in the pocket \(P_i\). The robot does this using two pebbles as follows.

**Lemma 9.1.** The explorer can compute in a constant number of steps two indices \(j_1\) and \(j_2\) such that \(v_{i+1}\) and \(v_0\), respectively, are the \(j_1\)th and \(j_2\)th vertices in the cyclic ordering of the combinatorial visibility vector \(cvv(v_i)\).

**Proof.** First, we prove the lemma for the explorer with two pebbles. From \(v_0\), which is marked by the first pebble, the robot heads straight towards \(v_{i+1}\), drops a second pebble there, and returns to \(v_0\). It then, heads to \(v_i\), and identifies \(v_{i+1}\) as the first vertex in the cyclic order of \(cvv(v_i)\) that has a pebble—its index is the value \(j_1\). Similarly, the index of the second vertex with a pebble serves as the value \(j_2\). Having identified both \(v_0\) and \(v_{i+1}\) in the local visibility of \(v_i\), the robot can then recollect these pebbles, ensuring that we only use at most two pebbles throughout the algorithm.

The procedure can be easily simulated by the explorer with only one pebble, at the cost of an increased amount of total movement (and of verbosity for the description – we leave this as an exercise for the reader).
Once the robot can identify $v_{i+1}$ from its local view at $v_i$, it knows the extent of the pocket $P_i$: the edge $v_i v_{i+1}$ marks the end of the pocket and the recursive call to the triangulation.

Secondly, by always visiting the pockets in a cyclic order, the robot can consistently compute a “vertex labeling” that serves to identify and store the triangulation globally. In particular, while at position $v_0$, the robot can see all the vertices in its visibility vector, that “view” is entirely local—the $j$th ccw vertex in $cvv(v_0)$ has no meaning to the robot when it is located at another visible vertex $v_k$. Thus, during the triangulation algorithm, the robot computes a global labeling, which is a cyclic ordering of the vertices in $P$, starting from the base vertex $v_0$. This labeling is computed easily as follows. The robot assigns the label 0 to the vertex $v_0$; that is, $\ell(v_0) = 0$. It then assigns increasing labels to all the vertices in $cvv(v_0)$ until it comes to the first 0-edge, say, $e_i = (v_i, v_{i+1})$. As the robot recursively computes the triangulation of $P_i$, it assigned labels in the pocket, starting with $\ell(v_i)$, and ending with the label $\ell(v_{i+1})$. At this point, the robot continues the labeling in $cvv(v_0)$ until the next 0-edge, and so on. In the end, the triangulation is stored in the robot’s memory as a collection of diagonals, where diagonal $(i, j)$ means the presence of a triangulation edge between the vertices that have indices $i$ and $j$ in the ccw walk along the polygon starting at $v_0$.

So far we have described the triangulation algorithm using a single explorer with $\Theta(n)$ memory (necessary to store the triangulation). But it is easy to convert this into a distributed implementation, using a group of $m$ robots, each with $\Theta(n/m)$ memory. The single-robot algorithm can be simulated easily in this new setting: one robot acts as a leader and executes the triangulation algorithm, while the others follow passively the leader, acting as storage devices and simulating the role of the pebble, when required. This shows that $\lceil \frac{n}{3} \rceil$ robots, each with $O(1)$ memory, can achieve the triangulation.

**Theorem 9.2.** The explorer can compute a labelling of the vertices of a polygon $P$ of size $n$ and compute a triangulation of $P$ according to this labelling. The algorithm is easily turned into a distributed implementation using $m$ simple robots, each with $\Theta(n/m)$ memory. The number of steps is $O(m \cdot |E(T(v))|)$, where $v$ is the initial vertex.

**Proof.** Let $P$ be a simple polygon. If we draw the 0-edges in $P$ picked by the explorer, we trivially partition $P$ into a set of disjoint
Chapter 9. The Art Gallery Problem

star-shaped subpolygons\footnote{A star-shaped polygon is a polygon which has a center, i.e., a point from which the whole polygon is visible.} The point from which a single subpolygon is triangulated is by construction precisely the first endpoint of a 0-edge. It follows immediately that no two diagonals intersect and that the whole polygon is eventually triangulated. Lemma 9.1 takes care that the explorer can perform the algorithm. The claims about the memory requirements and number of steps are immediate. \hfill \Box

Notice that what the algorithm actually does is to compute a small subgraph on \( n \) vertices of the visibility graph of \( P \). This subgraph has a linear number of edges only. This information is already enough to allow the construction of a triangulation (and, later, to solve the Art Gallery Problem).

One might be tempted to simply place a guard on the vertices that correspond to the nodes of \( T \), without computing a triangulation and a coloring. While this obviously solves the Art Gallery Problem, the number of guards required in the worst case is equal to \( n - 1 \) (see for instance the polygon in Figure 9.4 when starting the algorithm at the node \( v \)).

**Geometrically close triangles** In the algorithm as described, we have made no effort to ensure that the triangles stored by each robot are *local* in the sense that they are geometrically near to each other. However a slight modification of the ordering in which the triangles are picked can easily achieve that each of the \( \lfloor n/3 \rfloor \) robots stores at most three triangles, where one of them shares a vertex with each of the other two, when \( n = O(m) \). To do so, it is enough that the leader triangulates the pocket \( P_i \) it has just entered by adding ccw every diagonal originating at the robot’s position in \( P_i \), starting from the last selected one during the last visit of \( P_i \) (or from the right neighbor, if \( P_i \) is visited for the first time) until the second endpoint of the next unvisited 0-edge. This is then visited recursively. The sequence of such triangles has the property that every triangle has a vertex in common with its predecessor and its successor. To see this, we pick two successive triangles \( T_1, T_2 \) of this sequence. If \( T_1 \) and \( T_2 \) are picked in the same recursion call, the claim follows trivially. If \( T_1 \) and \( T_2 \) are not picked in the same recursive call, then \( T_2 \) must have been picked either (1) in a deeper recursion call than \( T_1 \) or (2) after exiting the
9.4. Guarding an Art Gallery

The algorithm for the Art Gallery Problem To solve the Art Gallery Problem in our sensing model, we just need to show that the triangula-

recursion call where \( T_1 \) was picked. By construction, at least a triangle is immediately picked right after calling a recursive call, hence \( T_1 \) and \( T_2 \) share the first vertex of the 0-edge responsible for the recursive call for \( T_2 \) and the claim is proved if (1) occurs. The case (2) requires a more careful argument, because it is not guaranteed that the triangle \( T_2 \) is picked immediately after exiting the recursive call of \( T_1 \). Suppose that \( T_1 \) is specified by the vertices \( a, b, c \), where \( a \) is the vertex where the robot is sitting on to triangulate the current region and \( b, c \) are two consecutive vertices of the cvv of \( a \). We claim that the next triangle picked by the algorithm shares the vertex \( c \) with \( T_1 \). Call \( d \) the vertex where the previous recursion was called. Thus \( a \) and \( c \) form a 0-edge in the cvv of \( d \). If \( c \) is not the last vertex in the cvv of \( d \) in the current pocket, the claim follows trivially. Otherwise \( c \) and \( d \) form a 0-edge of another node \( e \) where the recursion was previously called. The same argument applies again. In this way, we are navigating back a branch of an exploration tree. Since \( T_2 \) eventually is going to be picked, this chain of exiting recursive calls stops and a triangle with the vertex \( c \) is picked. Figure 9.5 illustrates the situation. Thus, it is enough for the robots to store three contiguous triangles from this list to achieve the required property.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure9_5}
\caption{This picture illustrates the case (2) of the aforementioned argument. The triangles have been drawn without using the polygonal edges and diagonals to ease the understanding. \((a, c)\) is a 0-edge of \( d \) and \((c, d)\) is a 0-edge of \( e \).}
\end{figure}

9.4 Guarding an Art Gallery

The algorithm for the Art Gallery Problem To solve the Art Gallery Problem in our sensing model, we just need to show that the triangula-
tion computed in the previous section can be 3-colored by our robots. Recall that the vertices have no coordinates, so we assign colors to the computed labels of the vertices. This is trivial, because a robot can greedily compute the 3-coloring of the triangulation that they have in memory. Nevertheless, we show how this can be carried out during the construction of the triangulation.

**Lemma 9.3.** The explorer can online 3-color the triangulation of a polygon during the execution of the Algorithm TRIANGULATION.

*Proof.* See Figure 9.6 for illustration. First note that a 3-coloring exists due to Fact 7.2. The explorer begins by coloring the initial vertex \( v_0 \), from which the triangulation algorithm begins, as 1. It then colors all the vertices in \( cvv(v_0) \) alternately as 2 and 3. Thus, each 0-edge in the visibility vector of \( v_0 \) is colored \((2, 3)\). When the explorer now visits the pockets \( P_i \), it propagates the coloring. If the pocket \( P_i \) is triangulated by the explorer from the vertex \( v_i \), and the color of \( v_i \) is 2 then the diagonals incident to \( v_i \) can be colored by alternating between colors 3 and 1, and so on. It is easy to see that this coloring succeeds, because both the diagonals picked by the robot and the polygonal edges receive different colors by construction. □

![Figure 9.6: 3-coloring the triangulation.](image-url)
Art Gallery Problem – since the robot labels the nodes sequentially (in cyclic order) around the boundary, finding those nodes is trivial.

As before, we can implement this algorithm using a collaborative group of \( m \) simple robots, each with \( \Theta(n/m) \) memory. In fact, once the triangulation (along with the coloring that can be performed simultaneously) is in place (Theorem 9.2), the leader assigns every vertex of the least used color to a robot, which then can autonomously deploy.

**Theorem 9.4.** The explorer with \( \Theta(n) \) memory can solve the Art Gallery Problem in a polygon of size \( n \) (determine the guarding positions). Alternatively, a group of \( \lfloor n/3 \rfloor \) explorers, each with \( \Theta(1) \) memory, can solve the Art Gallery Problem and self-deploy to guard the polygon. The number of steps is \( \mathcal{O}(n \cdot |E(T(v))|) \), where \( v \) is the initial vertex.

**A faster algorithm** The algorithm above has been deliberately chosen for its description ease. Variants that improve on the total number of steps performed do exist. We now show how to reduce the number of steps to \( \mathcal{O}(n \cdot \text{depth}(T)) \). The depth of a tree is the length of the longest path between the root and a leaf. As we will see, a further advantage of this new approach is that the collaboration among the robots is really crucial. First, recall that in this section we argued how an explorer can construct and navigate in \( T \) (Lemma 9.1). Hence we just assume this capability without further comments to allow an easier presentation.

To design a better algorithm, observe that the above presented coloring schemes works without having the triangulation at hand. Every recursion pocket can be colored online with only the information about the colors assigned to the endpoints of the diagonal defining the pocket (Lemma 9.3) – this is a local property. To illustrate the idea of the improved algorithm, we consider an explorer with \( \mathcal{O}(n) \) memory to explore the polygon and determine the guarding vertices, and we show later how this can be simulated by \( \lfloor n/3 \rfloor \) robots with \( \Theta(1) \)-memory. Again, the robot uses the exploration tree \( T \) as the navigation scheme: it performs a Depth-First Search in \( T \) and executes the coloring algorithm discussed above. For every node \( w \) of \( T \) it maintains the following state information:

- The indices of the cvv to locate the parent of \( w \) in \( T \) and the
vertex of \( P \) that specifies the current pocket with \( w \).

- The depth of \( w \) in \( T \).

- The total number of robots located on any node of \( T^*(w) \) (for the distributed version), whereby \( T^*(w) \) is the subtree of \( T \) rooted at \( w \).

- A triple \( \alpha(w) = (\alpha_1, \alpha_2, \alpha_3) \), where \( \alpha_i \) represents the number of vertices of color \( i \) in \( T^*(w) \), \( i = 1, 2, 3 \).

Note that \( \alpha(w) \) is obtained recursively: \( \alpha(w) = (\sum_{u \text{ child of } w} \alpha(u)) + (k_1, k_2, k_3) \), where \( k_i \) is the number of vertices colored \( i \) in the current pocket (not including \( w \)) and the sum is taken component-wise. Observe that \( k_j = 0 \), for at least one color \( j \).

When \( \alpha(v_0) \) is known (recall that \( v_0 \) is the root of \( T \)), the least frequently used color can be determined – suppose that this is color \( i \). The recursive deployment of the guards can begin: retrieve the \( i \)th \( \alpha \)-components of the children of \( v_0 \) and send this amount of robots in the corresponding pocket. The coloring algorithm is simulated again to recover the position of the vertices of the current pocket and some robots deploy to them. This procedure is called recursively for every further pocket. The coloring algorithm only depends on the current depth and on the color of the current vertex. Obviously, the guarding positions established by this algorithm are the same as the original algorithm, hence the correctness follows.

We show next how this algorithm can be simulated by \( \lfloor n/3 \rfloor \) robots with \( \Theta(1) \) memory each. The idea behind this is to send in the first phase only a minimal number of robots into a particular pocket, such that (1) every robot in the pocket will eventually guard a vertex in this pocket and (2) these robots maintain a communication chain between the vertex where the algorithm was executed (the “robot’s reservoir”), and the robot deeper in a pocket. In case that the number of robots inside a pocket is not sufficient for a complete exploration, this communication chain permits to efficiently request a new robot to continue the task. We will make this precise in the following.

We subdivide the tree \( T \) into trees of depth three, four or five in the following way: start with the root of \( T \) and call \( T' \) the subgraph induced by all the vertices of \( T \) at distance at most three to the root. Remove \( T' \) from \( T \) (i.e., take the graph induced by \( V(T) \setminus V(T') \), which is the union of disjoint trees) and proceed recursively with all
the remaining trees. If one of the remaining trees has not depth at least three, we merge it with the corresponding subtree $T'$. We make one robot responsible for such a subtree $T'$ of $T$, for which it maintains the state information of its root. Note that since the depth of $T'$ is constant, the robot can always maintain with constant memory the information to reach the root of $T'$ from any vertex of $T'$ it is currently on. Hence it is possible to compute the state information of every node of $T'$ from the state information of the leaves of $T'$. Moreover observe that $\min_i |\alpha_i(u) - \alpha_i(v)| \geq 1$, if $u$ is a grand-parent of $v$. So, the robot responsible for $T'$ can always find a guarding position within $T'$.

We are ready to describe the algorithm. Initially the explorers are sitting at a common vertex $v$ of $P$. A first explorer is designed to be responsible of the maximal subtree $T'$ of depth at most 5 rooted at $v$. Every son of a leaf of $T'$ which is not a leaf of $T$ becomes a root of a new subtree and a new robot moves from $v$ to it. Then the procedure continues recursively. Any request for new explorers, or any update for the information state that has to flow from a node of $T$ to the root is transmitted via the explorers responsible for the subtrees in-between on the path to the global root. Since every branch is visited sequentially, this is always possible. Once the root of $T$ knows its state information, the final deployment of the explorers to their guarding position starts. The set of explorers waiting at a root are sent in the right amount into every branch of the corresponding subtree by the responsible explorer. By an easy modification similarly as before, we can make the algorithm feasible to an equally sized group of simple robots (rather than explorers) without an asymptotic increase on the number of steps.

To compute the total number of steps performed, note that every responsible robot has to walk along the edges of its subtree only twice (not including the steps required to deliver messages) and the union of each edge-set is $E(T)$. Further, every time a message has to be delivered $O(\text{depth}(T))$ steps are sufficient. We crudely bound the number of messages required by the number of nodes of $T$. Finally, $n$ robots require $O(\text{depth}(T))$ steps to reach the guarding position. Hence, the algorithm terminates within at most $O(n \cdot \text{depth}(T))$ steps.

Summarizing, we have the following result.

**Theorem 9.5.** A group of $\lfloor n/3 \rfloor$ simple robots, each with $\Theta(1)$ memory, can solve the Art Gallery Problem and self-deploy to guard the
Chapter 9. The Art Gallery Problem

The number of steps is $\mathcal{O}(n \cdot \text{depth}(\mathcal{T}(v_0)))$, where $v_0$ is the initial vertex.

**The decision problem** Even though a group of explorers are able to solve the Art Gallery Problem, it is interesting to note that even a group of stronger labellers cannot resolve the *decision version* of the problem. Suppose that a group of labellers are initially placed on a set of vertices of a polygon $P$. The question for the robots is to decide whether the set of vertices they were assigned to correspond to a guarding configuration or not. To make the impossibility result stronger we adopt the model of global communication, i.e., each robot can communicate to every other robot in the polygon.

**Theorem 9.6.** The decision version of the Art Gallery Problem cannot be solved in general by a group of labellers (or reflexers).

*Proof.* Recall that the labeller (and the reflexer) are equivalent (Theorem 8.16). In turn, the labeller (or a group of labellers, Theorem 8.17) is equivalent to the delphic robot with access to visibility graph and the vertex-edge visibility graph. Figure 9.7 depicts a situation with two polygons with identical visibility graph and vertex-edge visibility graph. However, the vertex configuration on the left side is guarding, while the vertex configuration on the right is not.

![Figure 9.7](image)

*Figure 9.7:* The two polygons have the same visibility graph and vertex-edge visibility graph, but in one case (left) the marked vertices are in guarding configuration and in the other (right) are not (the shaded area is not guarded).

On the other hand a group of surveyors can solve the decision
version of the Art Gallery Problem. To see this, we need the following lemma.

**Lemma 9.7.** Let \( v_0, \ldots, v_{n-1} \) be the vertices of a polygon \( P \). Suppose that \( \{v_{i_1}, \ldots, v_{i_k}\} \) is a guarding configuration of \( P \). Let \( m(x) = A \cdot x + b \) be an affine and invertible mapping in \( \mathbb{R}^2 \) (i.e., \( A \) is invertible). Consider the points \( v'_i = m(v_i) \), \( i = 0, \ldots, n - 1 \). Then \( v'_0, \ldots, v'_{n-1} \) induce a polygon \( P' \) and \( \{v'_{i_1}, \ldots, v'_{i_k}\} \) is a guarding configuration of \( P' \).

**Proof.** An elementary argument from linear algebra shows that if two segments of the plane do not cross, then their image under an affine and invertible mapping do not cross either. Hence \( \{v'_0, \ldots, v'_{n-1}\} \) induce a polygon.

Further suppose that \( \{v'_{i_1}, \ldots, v'_{i_k}\} \) is not a guarding configuration in \( P' \). Hence there is a point \( x' \) in \( P' \) that is not visible from any of the vertices \( \{v'_{i_1}, \ldots, v'_{i_k}\} \). However the corresponding point \( x \) in \( P \) (which is the preimage of \( x' \) according to \( m \)) was visible by at least a vertex in \( \{v_{i_1}, \ldots, v_{i_k}\} \) – assume without loss of generality that it was \( v_{i_1} \). In other words, the segment \( v_{i_1} x \) does not intersect any edge \( e \) of \( P \), while the segment \( v'_{i_1} x' \) intersects an edge \( e'_{j} \) of \( P' \). Analogously as before this is impossible and the lemma is proved. \( \square \)

Lemma 9.7 states in particular that the solution of the decision version of the Art Gallery Problem is invariant under the class of orientation-preserving similarity transformations. Hence Theorem 8.7 implies the following result.

**Theorem 9.8.** The decision version of the Art Gallery Problem can be solved by a group of surveyors.

### 9.5 Open Problems

Numerous variants of the Art Gallery Problem have been investigated in the domain of computational geometry. For instance, one might be interested to solve the same problem with patrolling guards that move along an edge. Many of these modifications might be of interest to the engineering community and an analysis in the combinatorial sensing model is definitely appealing.
Chapter 10

The Target Counting Problem

People who count their chickens before they are hatched, act very wisely, because chickens run about so absurdly that it is impossible to count them accurately. – Oscar Wilde (October 16, 1854 – November 30, 1900) Irish playwright, novelist, poet.

10.1 Introduction and Motivation

We clarify the title of the chapter by an example. Imagine to be a museum custodian at the Louvre Museum. You are assigned the task to count the number of visitors in a certain wing of the museum. Luckily for you, the visitors are delighted by the displayed masterpieces, so they take their time to admire the paintings and they do not move very fast. On the other hand, you need to walk among the different rooms and corridors to come across every visitor. The main difficulty is that there are many guests, so if you look at the same room from two different perspectives to cover every angle, you might count a visitor more than once without noticing. Is it possible to obtain a good estimate of the number of visitors?

This chapter investigates the power of our robot models on an elementary yet natural problem of counting objects of interest in the robots’ environment. This is called the Target Counting Problem.
Chapter 10. The Target Counting Problem

As before, the environment is a polygon $P$ (simply- or multiply-connected) and the objects of interest, namely targets are modeled as a set of points inside $P$. By $n$ we denote the number of vertices of $P$ and by $m$ the number of targets therein. For simplicity we assume that the targets and polygon vertices are in a general position, i.e., no three points are collinear. We consider two different scenarios to model two basic classes of applications. In the friendly environment, the robot is allowed to walk to any target. In the hostile environment, the robot is not allowed to walk to targets. This scenario models the situation where a target represents an unsafe entity and coming into an imminent closeness to targets is dangerous.

Since we introduce a new object in the robot’s workspace, we slightly extend the definition of a simple robot for the purpose of this chapter. Most importantly, we add the capability to see a target in a polygon (by adding the corresponding element to the set $\mathcal{V}$ of visually distinguishable features, see page 126), and to see if there is a pebble or a robot on it. The targets are otherwise, similarly to the vertices, visually indistinguishable. The simple robot can however distinguish a vertex from a target – the information is provided by the piv. Formally, the robot’s sensing system provides the robot as before with an ordered list of points, that can be vertices or targets. The piv encodes in the corresponding component the visual features of a point, e.g., if it is a vertex, a target, a target with a pebble on it, etc. In particular, the piv tells how many targets lie in the cone induced by two consecutive visible vertices (see Figure 10.1). Sitting on a vertex of $P$, we assume that the cyclic order of the visible points (vertices and targets) starts with the neighboring vertex on the right. For a robot located on a target, we make no assumption about the first visible point – it is chosen by an adversary. This difficulty has the consequence that if a robot sitting on the vertex $v$ wants to place a pebble on a target and move back to $v$, it needs two pebbles. We place the question whether this is possible with only one pebble (thus with access to the visibility graph of the polygon) as an open problem. See Figure 10.1 for an illustration of the cvv and the piv in a polygon with targets. To deal with targets, we work with word-memory units where one word of memory has $\Theta(\log(\max\{m, n\}))$ bits.

We are interested in deterministic algorithms that provide the exact count of targets in all instances. However this is not always possible. Hence we investigate approximation algorithms as well,
10.1. Introduction and Motivation

Figure 10.1: An illustration of a point identification vector (piv) and a combinatorial visibility vector (cvv) in the polygon $P$ (with 4 targets); the piv is $(v, t, v, v, t, v, v)$ (where $t$ stands for target and $v$ for vertex) and the cvv is $(1, 1, 0, 1, 0, 1)$. The dotted lines specify the cones induced by consecutive vertices.

i.e., algorithms that deliver a (provably good) estimate on the number of targets. Further, we look for estimates that are never smaller than the actual number of targets. We say that an algorithm is a $\rho$-approximation for the Target Counting Problem if for the setting with $m$ targets, $m \in \mathbb{N}$, the algorithm estimates the number of targets by $z$, for which $m \leq z \leq \rho \cdot m$.

To demonstrate the notion of approximation and to justify the sensing model for the targets we illustrate that for a weaker sensing model no non-trivial approximation exists. Consider the sensing of the vertices in the same way as we defined before, but consider the sensing of the targets only by their presence, i.e., not interleaved with the vertices. Thus, the only information the robot gets is the number of visible targets (but not their ordering within the vertices of $P$). Figure 10.2 depicts two different scenarios, one scenario with $m = 1$ target and the second scenario with $m = n/3$ targets. In both scenarios the robot senses from every vertex exactly one target and therefore cannot distinguish the two scenarios. Hence, for this simple sensing model no approximation algorithm can guarantee a ratio better than $n/3$. 

Chapter 10. The Target Counting Problem

Figure 10.2: If a robot only senses the number of targets then the number of targets cannot be approximated within $o(n)$.

10.2 Summary of Results

For the friendly scenario we show that the explorer with two pebbles can count the number of targets in any polygon $P$. In contrast, we show that in the hostile scenario, not even labeller (or reflexer) can count the targets in general. However, we provide a 2-approximation algorithm for reflexer which is tight in sight of the corresponding inapproximability result.

Requiring the robots to count targets only from afar in general polygons is a more complicated problem, and we must endow the robots with some additional capabilities as anticipated before. Surprisingly, we show that these additional capabilities are quite minor, yet subtle. In fact, we consider two possible models, and show their implications to our problem. We consider robots that can walk along edge or diagonal extensions, i.e., if a robot picks a visible vertex $u$ as the direction of its walk, the robot can continue its walk in the same direction after it reaches $u$, if there is no polygonal edge to prevent it. In the second model we consider one additional global direction (think of the north direction) in which the robot can walk from any vertex of $P$. This types of motions enable a robot to specify consistently new points on the boundary of the polygon. Consequently we can show that a robot can solve the Target Counting Problem in both models.

The results of this chapter have been established in a joint work with Beat Gfeller, Matúš Mihalák, Subhash Suri and Peter Widmayer [48].
10.3 The Friendly Environment

In this section we show that in a friendly environment the explorer with two pebbles can count the targets in any simply- or multiply-connected polygon.

We consider simply-connected polygons first. In the beginning the explorer counts \( n \), the number of vertices of the polygon. The idea of the algorithm is to go to every vertex \( v_i \), \( i = 0, 1, \ldots, n - 1 \), and count the targets that are visible from \( v_i \) and that are not visible from any vertex \( v_j \), \( j < i \). We call these targets *newly visible* at vertex \( v_i \). Thus, the explorer can go through vertices \( v_i \), \( i = 0, 1, \ldots, n - 1 \), and sum up all newly visible targets. Clearly, no target will be counted twice, and therefore the resulting sum is the total number of targets.

We now describe how the robot can identify whether a target is newly visible. Being at vertex \( v_i \), the robot wants to identify whether the \( k \)th target in its visibility vector is newly visible. The robot goes to the target, leaves a pebble there, and checks for every vertex \( v_j \), \( j < i \), whether the pebble is visible from \( v_j \). The navigation from the target back to the vertex \( v_i \) can be done by leaving the second pebble at \( v_i \) and checking the position of \( v_i \) in the cvv of the target. Obviously, the target is newly visible if and only if the pebble is not visible from any vertex \( v_j \), \( j < i \). Overall, the robot needs two pebbles and a constant number of memory words (to remember the number of vertices, the current position \( v_i \), the position of \( v_j \) and the position \( k \) of the considered target of \( v_i \), and to mark the newly visible targets in the cvv of vertex \( v_i \)). Hence, in \( 2i \) steps we can check whether a target visible from the \( i \)-th vertex is newly visible. To check all targets at position \( i \) we need at most \( 2mi \) steps. Thus, the robot needs \( \mathcal{O}(mn^2) \) steps to count the targets in \( P \).

If the time is crucial, one can achieve a \( \mathcal{O}(mn) \) number of steps at the expense of used memory. For each vertex \( v_i \) the robot maintains the piv with the additional information stating whether a given target is newly visible. At the beginning, every target in the piv is marked as newly visible. Then for every vertex \( v_i \) the robot marks each newly visible target with a pebble and walks around the boundary towards vertex \( n \) and at every vertex \( v_j \), if the robot sees the pebble, it marks the corresponding bit in the bit array of vertex \( v_j \) as not newly visible. Thus, the robot walks \( m \) times around the boundary (for each target it walks exactly once and at most \( n \) steps), resulting in \( \mathcal{O}(mn) \) steps.
of the robot. The robot needs $O(mn)$ memory.

**Theorem 10.1.** In the friendly environment the explorer with two pebbles can count the targets in a simply-connected polygon in $O(mn^2)$ steps and with $O(1)$ memory, or in $O(mn)$ steps and with $O(mn)$ memory.

The result can be easily extended to multiply-connected polygons because the explorer can navigate through the vertices in a consistent way (Theorem 8.11), i.e., it can walk along a tour that visits all the vertices. Note that to do this, it is sufficient to have a representation of the hole graph and know the number of vertices of every boundary component. It is straightforward to see that this representation can be constructed and stored with $O(n)$ memory.

**Corollary 10.2.** In the friendly environment the explorer with two pebbles can count the targets in any polygon in $O(mn^2)$ steps and with $O(n)$ memory.

### 10.4 The Hostile Environment

After considering the Target Counting Problem in the scenario where the explorer can walk to targets, we consider now the scenario where the explorer walks only to vertices of $P$.

#### 10.4.1 Partition of the Polygon and Counting

**General Idea**

The algorithms are based on the idea of partitioning the polygon into triangles and counting the targets in these triangles exactly. To illustrate the idea, consider a partition of $P$ into triangles having their vertices on the boundary of $P$ with the property that every triangle has at least one side $e$ on the boundary of $P$. We call such a triangle a *baseline* triangle, and each edge of the triangle that lies on the boundary of $P$ a *baseline edge*. A partition of a polygon into baseline triangles is called a *baseline triangulation*. We might want to require

---

1. Here, the triangles are not necessarily meant in the strict sense of the previous chapter.
10.4. The Hostile Environment

that the baseline triangles of a baseline triangulation are triangles in the classical sense, i.e., specified only by vertices of $P$ – these are called baseline vertex triangles (or only vertex triangles, if we do not insist on the baseline property). A triangulation into baseline vertex triangles is also called a Hamiltonian triangulation \(^6\), as its dual is a path.\(^2\) Unfortunately, a Hamiltonian triangulation does not always exist, see Figure 10.3. We call a polygon that admits a Hamiltonian triangulation a Hamiltonian polygon.

In the case of a baseline triangulation the explorer can count the targets with the following algorithm. For every baseline triangle the explorer moves to a vertex of the triangle (recall that this might not be a vertex of the polygon, so a pebble must be there) opposite to a baseline edge, and counts the targets that are visible between the two vertices of the edge. Clearly, in this way every target is counted exactly once. Hence, a general algorithm that allows a robot to solve the Target Counting Problem is composed of a procedure to produce a baseline triangulation and of a navigation scheme to account for every baseline triangle exactly once.

---

\(^2\)A dual of a triangulation is a graph, where each triangle corresponds to one vertex and there is an edge between two vertices iff the two corresponding triangles share an edge in the triangulation.
10.4.2 A Tight Approximation

Inapproximability

We show that the Target Counting Problem cannot be approximated to within a factor $2 - \varepsilon$, for any $\varepsilon > 0$, even if the polygon $P$ is simply-connected. We start with a warm-up example to illustrate the idea. Consider the polygon in Figure 10.4. The polygon consists of four spikes attached to the four sides of a square. It depicts two scenarios with a different number of targets. In the first scenario there are 6 targets and in the second scenario there are 4 targets. Considering any vertex of the polygon, the cvv and piv of the vectors are the same in both scenarios. Hence, the robot cannot distinguish the two scenarios, which shows a lower-bound of $6/4 = 3/2$ for the approximation ratio.

![Figure 10.4: The Target Counting Problem cannot be approximated to within a factor 3/2.](image)

This construction can be extended to a general-sized polygon, where $2k$ spikes are attached to a regular $2k$-gon, using $2k$ and $4k - 2$ targets in two different scenarios, thus giving the desired inapproximability lower-bound of $2 - \varepsilon$.

**Theorem 10.3.** The Target Counting Problem cannot be approximated by the reflexer within a factor $2 - \varepsilon$, for any $\varepsilon > 0$, even in a simply-connected polygon.

Note that this inapproximability result relies only on the visibility limitations of the robots and not on their limited navigation capabil-
10.4. The Hostile Environment

Proof. We assume \( n \) is even, i.e., \( n = 2k \). The shape of the polygon is depicted in Figure 10.5 and 10.6. The polygon consists of \( n \) outer vertices \( y_0, y_1, \ldots, y_{n-1} \) and \( n \) inner vertices \( x_0, x_1, \ldots, x_{n-1} \). It can be viewed as an \( n \)-gon, a regular polygon formed by vertices \( x_i, \), \( i = 0, \ldots, n - 1 \), connected on each side \( x_i, x_{i+1} \) to a triangular spike \( x_i, y_i, x_{i+1} \). Here and further in the text, the indices are to be understood in a cyclic fashion. The line \( y_i x_i \) intersects the segment \( x_{i+1}, x_{i+2} \) in the middle. Thus, the visibility region of \( y_i \), i.e., the cone of \( y_i \) defined by lines \( y_i, x_i \) and \( y_i, x_{i+1} \), intersects the visibility regions of vertices \( y_{i-1} \) and \( y_{i+1} \), but not the visibility regions of other \( y_j \)s.

Observe first that a robot at vertex \( y_i \) sees only two vertices of \( P \), namely vertex \( x_i \) and vertex \( x_{i+1} \). Further, a robot sitting at vertex \( x_i \) sees all vertices \( x_j \), \( j = 0, 1, \ldots, n - 1 \), and vertices \( y_{i-1} \) and \( y_i \).

The aim is to place the targets in a way that a robot sitting at vertex \( y_{2l+1} \) sees one target (the piv is \( (v, t, v) \)), and a robot sitting at vertex \( y_{2l} \) sees two targets (the piv is \( (v, t, t, v) \)). For a robot at vertex \( x_i \), \( i = 0, \ldots, n - 1 \), we want the robot to see exactly one target between each two consecutive vertices of its piv, i.e., we want the piv to be \( (v, t, v, t, v, t, \ldots, v, t, v) \). Observe that the consecutive vertices of in the cvv of the vertex \( x_i \) are \( y_i, x_{i+1}, \ldots, x_n, x_1, \ldots, x_{i-1}, y_{i-1} \). We show how to achieve such visibility with two different number of targets. First we use only \( n \) targets and then we use \( 2n - 2 \) targets.

To place the \( n \) targets we proceed as follows. We place one target into each triangle \( y_i, x_i, x_{i+1} \). Observe that the triangle is divided into three parts by the lines \( y_{i-1}, x_i \) and \( y_{i-1}, x_{i-1} \). Let us label the parts \( P_1, P_2 \) and \( P_3 \), starting at a part containing the vertex \( x_{i+1} \). Figure 10.5 illustrates the partition. For odd \( i \), we place the target into part \( P_2 \). For even \( i \), we place the target into \( P_1 \). Observe now that a robot indeed sees one target from every vertex \( y_{2l+1} \) and two targets from every vertex \( y_{2l} \). Observe also that any vertex \( x_j \) sees exactly one target between two consecutive vertices \( x_i, x_{i+1} \), \( i, i + 1 \neq j \), because the parts \( P_1 \) and \( P_2 \) of triangle \( y_i, x_i, x_{i+1} \) contain exactly one target and the parts are completely visible from \( x_j \) within the segment \( x_i, x_{i+1} \). There is also one target visible in the segment.
Chapter 10. The Target Counting Problem

Figure 10.5: The partition of the triangle $y_i, x_i, x_{i+1}$ into three parts $P_1, P_2$ and $P_3$ by the lines $y_{i-1}, x_{i-1}$ and $y_i, x_i$.

We now use $2n - 2$ targets in $P$ to achieve the same visibility configuration. First, we place one target into every triangle $x_i, y_i, x_{i+1}$ such that the target is visible only from vertices $x_i, y_i$ and $x_{i+1}$. This can be easily achieved when the target is placed very close to $y_i$. This leads to the piv being $(v, t, v)$ at vertices $y_i$ and the piv being $(v, t, v, \ldots, v, v, t, v)$ at vertices $x_i$. The remaining $n - 2$ targets are placed in the following way. For the presentation purposes we label the targets $t_1, \ldots, t_{n-2}$. Each target $t_i$ is placed close to vertex $x_i$ and in the cone $C_i$ of $x_i$ defined by the vertices $x_{n-1}, x_n$. More precisely, by placing $t_i$ close to $x_i$ we mean to place the target $t_i$ into the triangle $T_i := x_{i-1}, x_i, x_{i+1}$. Observe now that for any placement of target $t_i$ into $C_i \cap T_i$ the piv of vertex $x_i$ is as desired, i.e., $(v, t, v, \ldots, v, t, v)$. Indeed, for vertex $x_i, i \leq n - 2$, the cone $C_i$ contains $t_i$ and thus the target is visible between $x_{n-1}$ and $x_n$. For every other cone of $x_i$ defined by vertices $x_j$ and $x_{j+1}$, the target $t_j$ lies in that cone. Also, for vertex $x_{n-1}$ the cone of $x_{n-1}$ defined by vertices $x_i$ and $x_{i+1}$ contains exactly one target, namely $t_{i+1}$. Similarly, the cone of vertex $x_n$ defined by vertices $x_i$ and $x_{i+1}$ contains
10.4. The Hostile Environment

exactly one target, namely $t_i$. To achieve the desired piv from the vertices $y_i$, we place each target $t_i$ within $T_i$ either to the left or to the right of line $y_{i-1}, x_{i-1}$. For $i - 1 = 2l$ we place $t_i$ to the right of the line $y_{i-1}, x_{i-1}$, so that $t_i$ is visible from $y_{i-1}$ (i.e., into the cone of $y_{i-1}$ defined by vertices $x_{i-1}$ and $x_i$). For $i - 1 = 2l + 1$ we place $t_i$ to the left of line $y_{i-1}, x_{i-1}$, so that $t_i$ is not visible from $y_{i-1}$. It is easy to observe that for every vertex $y_i$, its piv is $(v, t, v)$ if $i = 2l + 1$, and $(v, t, t, v)$ if $i = 2l$. A placement of $2n - 2$ targets into the polygon $P$ with $2n$ vertices, where $n = 12$, is depicted in Figure 10.6. This ends the proof.

Figure 10.6: A placement of $2n - 2$ targets into the polygon $P$. The arrows indicate the position of the targets.
Approximation

A Naive Approximation  Since the Target Counting Problem cannot be solved exactly in general, it is natural to look for approximate solutions, i.e., for good estimates of \( m \), the number of targets. Observe first that \( m \) is at least the number of targets visible from any vertex of \( P \). Let \( m_i \) denote the number of targets that are visible from vertex \( i \). We have \( m \geq \max_i m_i \). On the other hand, clearly, \( m \leq \sum_i m_i \).

Since every target is visible from at least three vertices of \( P \) (consider a triangulation of \( P \) and the vertices of the triangle, in which the target lies), we have \( m \leq \frac{1}{3} \sum_i m_i \). The explorer can compute the sum \( z = \sum_i m_i \) by navigating through all vertices of \( P \). Obviously, reporting \( \frac{1}{3} z \) as the estimate for the number of targets yields an \( \frac{n}{3} \)-approximation. Alternatively, if we denote by \( k \) the number of vertices with non-zero \( m_i \), the value \( \frac{1}{3} z \) becomes a \( \frac{k}{3} \)-approximation.

Although the approximation is not sound at first sight (consider a convex polygon with a single target in it), it gives some insight into the complexity of the Target Counting Problem. Notice that the derived approximation ratio depends solely on the number of vertices \( n \) (or on \( k \), the number of vertices with a view on at least one target) and not on the number of targets. Hence, if \( m \) grows in comparison to \( n \) or \( k \), the approximation ratio gets better. In other words, the approximation ratio does not grow with the number of targets, but is determined by the structure of the polygon (i.e., by \( n \)) and by the way the targets are placed in this structure (i.e., by \( k \)).

A 2-approximation Algorithm  The previous result can be refined. We present in the following a 2-approximation algorithm for the Target Counting Problem for labeller. In light of the previous section, this approximation is the best possible for this robot model (and for reflexer, since the two are equivalent).

The idea of the algorithm is to (recursively) partition a simply-connected polygon into discretely straight walkable subpolygons by adding suitable diagonals. In every subpolygon a Hamiltonian triangulation is built under the assumption that the added diagonals are actually edges in the subpolygon. By removing these diagonals we are left with the original polygon divided into baseline triangles, some of them extending into more than one of the mentioned subpolygons and hence overlapping. We will prove that every point of the polyg-
10.4. The Hostile Environment

Onal area is covered by at most two baseline triangles, leading to the claimed approximation guarantee. To ease the description, we present the algorithm with two labellers, but it will be easy to see that a single robot can simulate the algorithm.

First, a pebble is used to mark an origin vertex $v_s$, which introduces a localization system in the polygon $P$ as before. The two labellers start a feasible walk from $v_s$ towards a destination vertex $v_t$ that is not yet determined, one on each path ($L$ and $R$). The labellers move according to an arbitrary deterministic rule that ensures the visibility between the two robots. In the case that the walk cannot be extended without violating the visibility condition, one of the robots walks along the diagonal that connects its current vertex to the next vertex visible to the other robot. We call such a diagonal a *gateway* (see Figure 10.7). They continue in this way until the robots meet at a common point. The gateways and edges that were traversed by the robots define a subpolygon $P_1$, which is discretely straight walkable by construction. The labellers count the targets in all baseline triangles that were defined by the robots’ walk.

All the gateways of $P_1$ define pockets of $P$: a pocket is specified by the gateway and the portion of the boundary of $P$ between the vertices of the gateway that does not include the vertex from which the gateway was built. The pockets are partitioned recursively, but now the starting points $v_{sL}, v_{sR}$ and the destinations $v_{tL}, v_{tR}$ of the two robots’ walks are fixed: given a pocket defined by a gateway $d$, the endpoints of $d$ represent $v_{sL}$ and $v_{sR}$, whereas $v_{tL}, v_{tR}$ are the endpoints of the edge $e$ partially visible between the endpoints of $d$ with respect to the robot’s location $v$ where the gateway $d$ was built. We say that $e$ is the *shadow* of $d$ with respect to $v$. It is easy for labeller to identify the endpoints of $e$, because this is exactly its ability. The recursive partition of every pocket leads to a set of discretely straight-walkable polygons $P_1, \ldots, P_k$. When a robot counts the number of targets in a baseline triangle defined by a gateway, it possibly counts targets that are going to be counted again in the recursion of the corresponding pocket. Nevertheless, we claim that the algorithm counts every target at most twice and hence yields a 2-approximation. Figure 10.8 shows an example of an execution of the algorithm.

**Theorem 10.4.** The labeller can approximate the number of targets

---

3Note that since the endpoints of $d$ are consecutive vertices in the cvv of $v$, $e$ is uniquely defined.
to within a factor two in any simply-connected polygon.

Proof. For simplicity, we prove the result for the original algorithm with two labellers - the scenario with a single labeller follows easily. The algorithm induces as described a partition $P$ of the polygon $P$ in discretely straight walkable subpolygons $P_i$, $1 \leq i \leq k$. During the algorithm execution the labellers build two types of triangles: vertex triangles (the baseline is an edge of $P$) and gateway triangles (the baseline is contained in an edge of $P$, the shadow of the corresponding gateway). The latter type are exactly the triangles that extend into more than one subpolygon $P_i$.

Consider a single subpolygon $P_i$ and the set of triangles generated exclusively by the recursion call in $P_i$. If we restrict the gateway triangles by cutting off their portion not included in $P_i$, the triangles form a Hamiltonian triangulation of $P_i$ because $P_i$ is discretely straight walkable. Hence, if we show that every subpolygon $P_i$ contains the portion of at most a single gateway triangle defined in another subpolygon $P_j$, we obtain that every point contained in $P$ is covered by
10.4. The Hostile Environment

one or two triangles (neglecting the boundaries by the general position assumption) and the claim follows.

So, consider the dual graph $T$ of the partition $P$. Obviously, $T$ is a tree if $P$ is a simply-connected polygon. The algorithm execution in $P$ induces an orientation on the edges of $T$: every oriented edge $(p_i p_j)$ corresponds to a gateway triangle that has been specified by three vertices of the subpolygon $P_i$, but which has the baseline in the subpolygon $P_j$. Recall that the baseline of this gateway triangle is guaranteed to be an edge of $P$. So, by construction every gateway triangle traverses at most two subpolygons. Since $T$ is a tree, $P_j$ has at most a predecessor $P_i$ in $T$, hence there is at most a gateway triangle that might enter $P_j$.

Since we use the labeller’s peculiar capabilities, the explorer cannot execute the presented approximation algorithm in general (Theorem 8.15). However, the explorer can guess the label of the shadow of a gateway. Note that if some guess is wrong, the explorer still completes the covering of the polygon into triangles. Hence, in such an execution the explorer sees all targets. However, more than two triangles can possibly overlap, degrading the approximation guarantee. Consequently, we let the explorer enumerate all the (exponentially many) possibilities to label the shadows. The output of the algorithm is the minimum number of targets counted among all executions. Clearly, during one execution all guesses are correct, hence the explorer is able to achieve the 2-approximation as well.

**Corollary 10.5** ([61]). The explorer can approximate the number of targets to within a factor two in any simply-connected polygon.

**10.4.3 More Power to the Robots**

We have seen in the previous section that the explorer cannot exactly count the targets in a simply-connected polygon in general. We therefore look at possible enhancements of capabilities, which keep the robots as simple as possible and at the same time enable the robots to count the targets. We consider two such enhancements.

---

4 Analogously as before, the dual graph of this partition is as follows: every subpolygon $P_i$ is a node $p_i$ of the graph $T$ and two nodes are connected iff the corresponding subpolygons share an edge (a gateway of $P$).
Chapter 10. The Target Counting Problem

In the first one we allow the robots to walk along edge-extensions and diagonal-extensions, i.e., if a robot at vertex $v$ picks a vertex $w$ as the direction of the robot’s walk, the robot is allowed to continue walking in the same direction after it reaches $w$, and it will stop only when it hits the boundary, at a point $w'$. Figure 10.9 illustrates this enhancement. The line $vw'$ is called an edge-extension (diagonal-extension) if $vw$ is an edge (diagonal) of $P$. If we do not need to distinguish whether $vw'$ is an edge- or diagonal-extension, we simply say that $vw'$ is an extension. If a pebble is placed at $w'$, it is then visible in the same way as a vertex of $P$, and therefore the robot can go there from any vertex visible to it. $w'$ is then visually distinguishable from the other regular vertices of $P$, because it is marked with a pebble.

Figure 10.9: At $v$, a robot chooses $vw$ as the direction of the robot’s walk. After reaching $w$, it can continue in the same direction until it hits the boundary at point $w'$.

In the second enhancement we endow the explorer with the compass described in Chapter 7. With a compass, a global direction is introduced, the North. Further we assume that the explorer with a compass can walk along this direction. For simplicity of presentation we assume that the polygon does not have vertical edges. Note that with a compass the explorer can tell whether a visible point (a vertex or a target) is to the left or right of the vertical line, and whether it is above or below the robot, i.e., above or below the horizontal line going through the robot’s position. Such an enhancement can be viewed as a navigation with a compass. If a robot walks from a vertex $v$ in the vertical direction we say that it walks along the vertical extension of $v$.

For both enhancements we present algorithms that allow robots to solve the Target Counting Problem inside a polygon $P$. 
10.4. The Hostile Environment

Walking along Edge- and Diagonal-Extensions

In this section we consider robots that can walk along edge- and diagonal-extensions. We show that such robots can partition any simply-connected polygon into baseline triangles, and thus can count the targets.

Consider the explorer at a vertex $v$ of the simple polygon $P$. Let $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ denote the visible vertices from $v$, cyclically ordered in the counterclockwise direction. Observe that the lines $vv_{i_j}$, $1 \leq j \leq k$, partition the visible part of $P$ into baseline triangles (all with a common point $v$), each with at least one baseline edge. See Figure 10.10 for an illustration, where the triangles $vw_1v_6$, $vv_6w_2$, $vv_{10}v_{11}$ and $vv_{11}v_{12}$ partition the visible part of $P$. Observe that the invisible part of $P$ is a set of disjoint simply-connected subpolygons. In the example from Figure 10.10 the subpolygons $P_1$ and $P_2$ form the invisible part of $P$. As customary, we call such a subpolygon a pocket of $P$ with respect to $v$. Observe that a pocket is created by a line which is an edge-extension or a diagonal-extension. Applying a recursive partitioning approach on the pockets, we create a partition of $P$ into triangles with at least one edge on the boundary of $P$ (see Figure 10.11). Let $T$ denote this triangulation.

![Figure 10.10: The extensions of a vertex $v$ define baseline triangles and pockets of $v$.](image1)

![Figure 10.11: The resulting baseline triangulation of the algorithm and the visited pockets (grey). The labeled dots show the order of the recursion calls.](image2)

The main idea of the algorithm is to count all targets from the robot’s position $v$ and then proceed recursively in the corresponding pockets of the polygon, thus navigating through $T$ and counting the targets in the triangles of $T$. We begin with a high-level description. For a vertex $v$ let $P_1, \ldots, P_\ell$ denote the pockets of $P$ defined by all
extensions originating at \( v \). Let \( p_i, i = 1, \ldots, \ell \), denote the visible vertex whose extension defines \( P_i \). Let \( w_i \) be the point of \( P \) for which \( vw_i \) is the extension of \( vp_i \).

**COUNTING IN SIMPLY-CONNECTED POLYGONS**

1. Count all the targets that are visible from the robot’s position at vertex \( v \).

2. Put a pebble at \( v \) and remember the position of \( v \) in the respective piv of every vertex \( p_i \) and of every point \( w_i \).

3. Recursively count the targets in \( P_i, i = 1, \ldots, \ell \), by marking the point \( w_i \) with a pebble and going to \( p_i \).

When the explorer walks to vertex \( p_i \) to start a recursive call for the pocket \( P_i \), it first checks the position of the pebble that marks the point \( w_i \). Next the explorer determines which vertices (and targets) visible from \( p_i \) belong to pocket \( P_i \). Let \( k \) be the number of vertices (including \( w_i \)) and targets visible from \( p_i \). Let \( h \) be the index of \( w_i \) in the piv of vertex \( p_i \). If pocket \( P_i \) lies to the right of \( p_iw_i \), then \( P_i \) contains the vertices and targets from the piv of \( p_i \) with index \( 1, 2, \ldots, h \). If pocket \( P_i \) lies to the left of \( p_iw_i \), then \( P_i \) contains the vertices and targets from the piv of \( p_i \) with index \( h, h + 1, \ldots, k \). Observe that \( P_i \) lies to the right of \( p_iw_i \) if and only if \( p_i \) is the first end-point of the diagonal in piv of vertex \( v \).

The explorer at vertex \( p_i \) knows which part of its piv represents the subpolygon \( P_i \) and it can therefore perform the same steps of the Algorithm Counting in Simply-connected Polygons on the pocket \( P_i \) only. Before that, the pebble from \( w_i \) is collected as it is no longer needed. When the robot finishes the counting in \( P_i \) it returns to the vertex \( v \) (using the stored navigation information) and continues there.

**Theorem 10.6.** The explorer, able to walk along extensions, can solve the Target Counting Problem in a simply-connected polygon of \( n \) vertices in \( \mathcal{O}(n) \) steps with \( \mathcal{O}(n) \) memory.

**Proof.** Let \( T \) be the baseline triangulation of \( P \) produced by the algorithm. The triangles of \( T \) are defined by vertices of \( P \) and intersection points between polygonal edges and extensions. Observe first that the algorithm provides a consistent navigation scheme through \( T \). Thus
every target is counted exactly once. Note that the dual of $T$ is a tree. The edges of every triangle of $T$ contain at least two vertices of $P$, which are then mutually visible and form a diagonal of $P$. Since any polygon contains exactly $n - 3$ disjoint diagonals and every diagonal can be used by at most two triangles of $P$, it follows that $T$ has at most $O(n)$ triangles. Hence the dual of $T$ has $O(n)$ vertices, which is also the number of steps of the algorithm (since the robot spends a constant number of steps in every triangle of $T$). The robot stores the necessary information to return from a recursive call – the predecessor $v$ of every vertex $p_i$. Hence, $O(n)$ memory is sufficient.

\[ \square \]

**Walking with a Compass**

In this section we consider in addition one fixed direction in which the robot can move. Without loss of generality, we assume that a robot sitting at a vertex can, additionally to moving to all visible vertices, move also along the vertical line going through the robot’s position.

We present an algorithm that computes a baseline triangulation in any simply- or multiply-connected polygon and navigates the explorer such that each triangle is considered for counting exactly once and thus it allows the robot to count the number of targets in the polygon. To simplify the presentation we first use an arbitrary number of pebbles – we show later how to use only a constant number of pebbles.

The key observation is that all the vertical extensions from vertices of a polygon $P$ partition the polygon into baseline triangles and convex quadrilaterals for which two opposite sides are on the boundary of $P$ (Figure 10.12). Each quadrilateral can be subsequently partitioned into two baseline triangles (by picking a diagonal as the common boundary of the triangles).

Hence, using at most $2n$ pebbles, the robot can mark every endpoint of every vertical extension which then imposes a baseline triangulation. This can be done by visiting every vertex of $P$ (Theorem 8.11). To count every target exactly once, the robot goes through every vertex or pebble $p$ and considers only triangles lying above $p$ and on its right (if any). Since every triangle has one vertical side, the robot can always reach the opposite vertex of the baseline side in one step and count the targets in the triangle, and return back.

We now show how to reduce the number of used pebbles at the
cost of an increased number of steps. The robot does not mark all the quadrilaterals at once, but one by one. Let us call an endpoint of a vertical extension a $q$-node. We show how to navigate through all the vertices and $q$-nodes in a consistent way. We begin by establishing the navigation instructions through vertices of $P$, where every edge of $P$ is visited exactly once. This is achieved as described after Theorem 10.1. If a robot moves in this navigation along a polygonal edge $uv$, we compute all the $q$-nodes lying on this edge and before the robot moves to $v$ it visits all the $q$-nodes in the order of increasing distance from $u$.

Let us consider the situation where the robot is at a point $p$ (a vertex $u$ or a $q$-node) of the edge $uv$ and it wants to move to the next $q$-node. The robot can find the next $q$-node by sequentially creating all $q$-nodes (by going to every vertex of $P$) and checking which one lies on the edge $uv$ and closest to $p$. Specifically, using a pebble the robot marks the initial position $p$. The next pebble is used to mark the so-far closest $q$-node on the edge $uv$. The robot goes through every vertex $w$ of $P$ and creates $q$-nodes lying on the vertical extensions of $w$. For every such $q$-node the robot checks whether it lies on the edge $uv$ and whether it is closer to $u$ than the current best. The two pebbles make this operation easy for the robot.

**Theorem 10.7.** The explorer with two pebbles, a compass, and able to walk along vertical extensions, can solve the Target Counting Problem in multiply-connected polygons in $O(n^3)$ steps and with $O(n)$ memory.
10.5 Open Problems

We raise the question whether the Target Counting Problem can be non-trivially approximated by a simple robot. This question seems non-trivial also due to the fact that the presented 2-approximation for the explorer requires an exponential number of steps. Intuitively, this problem is deeply related with the open problem of determining how the map of a simple robot looks like.
Index

asynchronous network, 64
bounding function, 62
combinatorial visibility vector, 123
communication complexity, 14
  multi-party, 46
  nondeterministic, 43
  one-way, 29
  two-way, 29
compass, 127
diagonal, 122
dominating set, 61
  connected, 61
exploration tree, 160
extension, 176
growth-bounded graph, 62
  p-, 62
handedness, 125
independent set, 61
  maximal, 61
  maximum, 61
map, 132
mechanism, 22
  strategyproof, 22
mechanism design, 20
  algorithmic, 21
  distributed algorithmic, 21
model
  CONGEST, 63
  LOCAL, 63
  combinatorial sensing, 123
  message-passing, 63
multicast routing protocol, 22
oracle, 127
  labelling, 127
pebble, 126
point identification vector, 123
polygon, 121
problem
  Art Gallery Problem, 157
  Set-Disjointness Problem, 37
  Maximal Independent Set Problem, 61
  Minimum Connected Dominating Set Problem, 61
  Rendezvous Problem, 155
  Multicast Cost Sharing Problem, 23
  Small Connected Spanning Subgraph Problem, 106
  Target Counting Problem, 173
robot model
  delphic, 132
  explorer, 128
  labeller, 128
  reflexor, 128
  simple, 125
  surveyor, 128
synchronous network, 64
triangulation, 122
unit disk graph, 62
visibility graph, 122
  vertex-edge, 122
Bibliography


Curriculum Vitae

Elias Vicari

Date of Birth    April 14, 1980
Place of Birth   Viganello, Switzerland
Citizenship      Switzerland

Education

2004 – 2008    ETH Zurich, Switzerland
    PhD Student at the Institute of
    Theoretical Computer Science
    with Prof. Dr. Peter Widmayer
    (2004 – 2005 with Prof. Dr. Markus Bläser)
    Academic title: Dr. Sc. ETH Zurich

1999 – 2004    ETH Zurich, Switzerland
    Student at the Department of Mathematics
    Academic title: Dipl. Math. ETH Zurich

1995 – 1999    Liceo di Mendrisio, Switzerland
    High School, scientific orientation
    Degree: Maturità Tipo C (high school diploma)
Publications


