Dissertation

Collective Graph Exploration

Mirosław Dynia

Universität Paderborn
Fakultät für Elektrotechnik, Informatik und Mathematik
Institut für Informatik & Heinz Nixdorf Institut (HNI) &
DFG-Graduiertenkolleg "Automatic Configuration in Open Systems"
Warburgerstraße 100, D-33098 Paderborn

Paderborn, August 2007

To my dear wife Ania and our precious daughter Magdalena

Acknowledgments

I would like to thank my wife for her support and being with me during all my doctoral studies. Also our little daughter Magdalena, for tearing apart the preliminary version of this work (it was quite messy anyway).

Special thanks to Prof. Friedhelm Meyer auf der Heide, who gave me this great opportunity to be a member of the research group "Algorithms and Complexity", and for the flexibility in choosing a research area to work on. It was an honor to work and develop myself in such an excellent environment of great people, ideas, and possibilities.

I also would like to thank Prof. Christian Schindelhauer for his professional support, openness and for all discussions, also those which did not end with interesting results. My great acknowledgments to my Polish friends Jaroslaw Kutylowski, Miroslaw Korzeniowski, and Jakub Lopuszanski, always ready to discuss the current ideas and problems, and to Bastian Degener who helped me to improve the readability of my dissertation. It all would not be possible without your supportive attitude.

Paderborn, August 2007 *Mirosław Dynia*

Contents

Chapter 1

Introduction

Years before computers were discovered, mankind was looking for new ways to do things faster, better, more accurate or just with less expense. Great navigators and explorers, like Christopher Columbus, driven by the lust for knowledge or power have built fast, robust ships which were able to cross the unbounded oceans horizon and discover the undiscovered. No man was that reckless to conduct the expedition with only one ship – nor was Columbus.

On the evening of August 3, 1492, he departs from Palos with three ships for his first voyage. Certainly, he faces the problem of an appropriate strategy for the ships under his command. If they stay together within eye-contact distance, it will make them maybe stronger against pirates, but the ability to explore will be decreased. On the other hand, the decision to split will increase these abilities, but unfortunately it will break their communication capability, and therefore the ability to make common decisions will be compromised. The time past, the technology made huge progress, but the question regarding the most efficient methods of group exploration is still open.

Although we focus on the exploration, it is just one example among a variety of problems, where the cooperation of team members can be observed. Always when many entities (people, robots, processors, software) participate in the accomplishment of some common task, we would like to observe profits from such a multiplication of "resources". Usually, we try to achieve an increased efficiency or a robustness, but unfortunately the additional coordination costs need to be taken into consideration.

A sufficient effort has to be invested in designing of appropriate methods, otherwise we will observe chaotic activities within the team, rather than a desired cooperation. The swarm intelligence shows that even from very simple rules a global solution might emerge, however, it can take much time for the desired properties to appear in the doings of the chaotic swarm. Therefore we believe that by making the rules slightly more complex, we obtain a solution with better quality and in a shorter time. Tailored

techniques allow to observe the team cooperation right from the beginning.

Nowadays, there are many unknown environments asking to be explored. It goes beyond the exploration of unknown terrains, ocean floor or surface of distant planets. The network is a classical environment asking to be explored. For instance, in case of Internet, which changes rapidly and nobody knows its complete structure, an efficient exploration would deliver information on its topology, allowing more effective routing of packets. To work on a certain abstract level, we create a model which suitably exposes the core problem, and on the other hand, is reasonable to analyze. We model the environment by *a graph* – an abstract structure consisting of nodes (locations to be visited) and edges (denoting accessibility between locations). Eventually, each node has to be visited by a simple entity (a robot) being a member of a larger homogeneous group of robots. We would like to observe how efficient the collective exploration can be, assuming various variants of the main problem.

The basic robot's ability is that it can move along edges of a graph. It cannot rapidly change its position within the graph and in order to reach some node, a robot has to traverse all edges along some path, between its current position and the destination. In terrain exploration, this is quite a natural assumption, since robots are physical entities related to their current positions. As we will later see, this physical-movement property is the main reason disabling existing classical algorithms. To emphasize this restriction, we call an entity a robot, however, here we do not pay too much attention to problems directly related to robotics (like sensor inaccuracy, odometry error or image processing), as this would unnecessarily make our problem more complicated and move the focus of the problem toward a specific area of application. We rather tend to study the problem on a higher abstraction level.

We are looking for robot's movement strategies which are simple, local, distributed, and deliver possibly minimal-cost solution. The locality of our strategies is reflected by the fact that each robot has a very restricted view of the overall state of the environment and the team, unless it exchanges information with other team members. As we often discuss the scenario where robots communicate only, if they are close to each other, the propagation of information is quite restricted. Each decision is based only on the partial knowledge possessed by a robot at a given point of time.

1.1 Motivation

One of the most obvious motivation is an exploration of a terrain by a large group of robots. Such a robotic swarm is deployed in an unknown terrain with the goal to recognize it. Typically, this scenario might be a result of a rescue expedition in dangerous

terrains (like regions touched by a natural disaster), or an exploration of distant planets or unaccessible terrains like the ocean's floor[1].

In a classical example, the team has to construct a topological map of the terrain. Additionally, information on some environmental data (e.g. temperature or salinity) can be collected and mapped on the topological map. Robots might also explore in order to find some interesting "objects" hidden in the terrain. Those might be some interesting minerals or rocks but maybe also bombs or land mines which has to be found and disarmed, or in case of a rescue expeditions, all people asking for help.

We expect that (as there are multitude of robots) the given goal will be fulfilled much more efficiently than in the one-robot scenario. As an additional factor a robustness of such a multi robot system is achieved – when faults occur, or some robots crash, there are still many team members ready to accomplish the given mission.

The usual technique for a one-robot scenario is that the human operator controlls the robots movement, basing on the pictures sent from the robot's camera, or on other type of information from another type of sensors. For instance, this is the case of robots disposing bombs, where the specially trained staff uses a control console (connected to the robot by a wire or per wireless channel) to undertake the dangerous task by the robot's manipulator. The clear advantage of this solution is a safe distance to the dangerous place. By the exploration of distant planets or the oceans floor, this solution is much more difficult to apply. It is unpracticable to connect the control console with the exploring robot by a wire, because of a large distance, and difficult, unpredictable conditions on the way between them. The quality of a wireless channel is often either to expensive or drastically weakened by the environmental conditions (poor propagation of radio waves in water) to the level disabling its usage for our purposes.

When we cannot take care of everything, we usually decide to delegate some parts of tasks. In this case, it is suitable to give more freedom to the robot and controll only some basic parameters of the mission. The technology is already there[2]. NASA's twin geologists Spirit and Opportunity, the Mars Exploration Rovers, that landed on Mars on January 4 and on January 25, 2004, are partially autonomous robots. They apply autonomous navigation detecting interesting locations (for instance a rock) and then drive toward them. In the first step, they use a stereo vision to construct the approximate 3D model of a terrain. After a target is selected, a traversability analysis is carried out. By the path selection process, dimensions of the robot, obstacles, and potential dangers are taken into account. Finally, the robot moves along the selected path toward the selected target. In this way, the robot sees the terrain as a collection of locations (nodes)

[1] NOAA Office of Ocean Exploration has 200 years history of mapping and characterizing the physical, biological, chemical, and archaeological aspects of the ocean.

[2] The goal of the RoboCup Project: "*By 2050, develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer*".

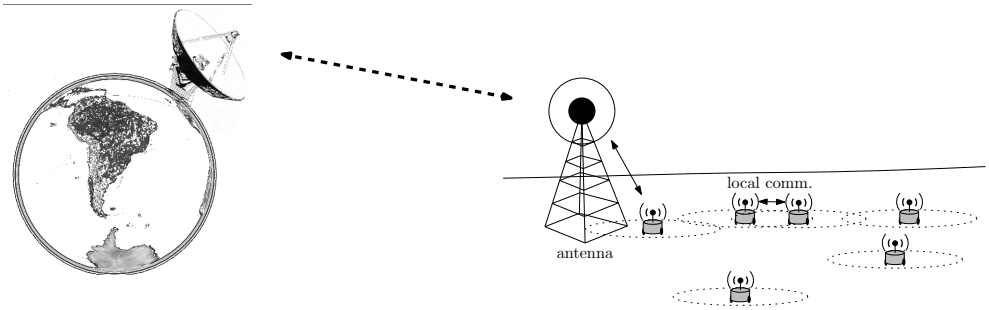

Figure 1.1: Collective Mars exploration scenario.

and paths between them (edges) and is able to traverse edges. Scientists and engineers on the Earth determine only the crucial mission parameters and command through a costly link, created by international network of antennas (NASA Deep Space Network), and shared also by many other space missions. Therefore, we might assume, that the robots of our team will be able to move within the terrain, without crashing or hitting against the obstacles, and they will need intervention of human controller only by some crucial decisions.

Suppose that a large team of robots lands at some location (called a base station) on a distant planet. The goal is to disperse in a terrain and explore everything within a 10km radius of the base station. As we have mentioned above, there are some first techniques enabling to model the terrain as a graph, were nodes denote interesting locations, and an edge models the accessibility between two locations. There is no central authority which controlls or coordinates the detailed movement of each robot separately, as it would require many costly communication channels. Therefore, each robot is autonomous and runs a predefined exploration algorithm designed in such a way, that the team self-organizes and reacts on the faced conditions and the terrain complexity. The human commander influences only some high-level mission parameters or decisions (start of the mission, termination ect.) and awaits the complete map of an unknown terrain, as a result of a robot exploration. We assume that there is one static communication antenna, placed in the base station, which establishes the communication link with the Earth. All robots, dispersed in the terrain, cannot communicate directly with the Earth as they are equipped with short-range communication devices. Only the robots sufficiently close to the antenna or other robot can exchange messages.

Unfortunately, the collective exploration is not well studied and neither humans nor robots know how to extract the potential power offered by a team of robots. As robots are autonomous, there is a need for tailored techniques which prove their efficiency in every possible terrain. The situation, where the method works fine for some easy terrains, but fails in a complicated one, is unacceptable. The distance, mission costs, and

unpredictability of the situation at a mission place are important factors here.

Our work delivers answers to the core questions in this setting, i.e. how to design an exploration method in such a way, that the robots efficiently cooperate without an intervention of a human controller. The methods should be adaptive and prove their efficiency, also in case of highly complex terrains.

1.2 General related work

The problem of exploration of an environment is widely studied in the literature in various contexts. We present a general overview of the results related to this problem. Later, in Chapter 3 and Chapter 4, we also make more detailed description of some selected publications, which are only mentioned here and are later presented from the perspective of our setting and the contribution they make to it.

It is quite often, that an environment to be explored is modeled as a graph or as a plane). In a simpler setting, a group of robots (agents) has to jointly visit all nodes or traverse all edges of a graph, **known in advance** to the algorithm. One of the measures, applying to this setting, is the length of the longest walk a robot makes during the exploration, where the goal is to minimize this value (MIN-MAX objective). Finding walks collectively visiting all nodes of a known graph and minimizing a MIN-MAX objective is an NP-hard problem (see e.g. [FGKP06]). Bellmore and Hong show in [BH74] reduction of multi-robot exploration problem to a classical traveling salesman problem. Frederickson, Hecht and Kim [FHK78] give $(5/2 - 1/k)$-approximation for the problem of collective visiting all nodes of a graph by k robots with a min-max objective function. They assume that robots start and finish in the same node of the graph (k-TSP). Their solution bases on the subdivision of a tour over all nodes (constructed by Cristofides algorithm). Authors present also constant approximations to some related problems, like the Chinese postman problem and stacker-crane problem for a directed graphs.

Arkin, Hassin and Levin consider in [AHL06] a variety (over 12) of vehicle routing problems. For instance, they present 4-approximation to the collective graph exploration (they refer to it as MIN-MAX PATH COVER) as well as methods covering a graph with stars, trees minimizing its size.

Even et al. present in [EGK+04] how to cover a graph by a collection of trees. Each tree defines a subgraph which has to be visited by a separate robot. They aim for minimizing a size of the maximal tree (assuming a bounded number of trees) and provide constant $(4 + \epsilon)$-approximation to this problem.

A similar problem is studied by Guttmann-Beck and Hassin in [GBH97]. They subdivide a graph into equal-sized subgraphs, minimizing the maximal length of a minimum spanning tree of subgraphs. They prove general bounds on the running time of any

algorithm (unless P=NP) as well as an algorithm with the approximation factor and the running time depending on the given input parameter.

Averbakh, Berman present in [AB97] a $2 - 2/(k+1)$-approximation for collective tree exploration. Unfortunately, the running time is exponential in k. It is only slightly improved by Namagochi, Okada in [NO04], still remaining exponential in the size of a team of exploring robots.

Averbakh and Berman show in [AB02] how to find the optimal initial locations of $k = 2, 3$ robots in a tree, without giving their routes. The same authors present in [AB96] a heuristic for $k = 2$ robots, but unfortunately the worst case approximation factor turns out to be 2, which is the worst possible result for the scenario with two robots.

1.2.1 Online algorithms for a single robot

An interesting area of research opens with the assumption that a graph is **not known in advance** to the algorithm. Robot has to learn it, sequentially traversing edge by edge, and exercising the cost of such a movement. In the worst case, the graph topology discovered in that way might be a highly unsuitable setting for the current distribution of robots. This could, for instance, enforce them to traverse the same edges many times or to do a costly relocation of robots. From the robot's perspective, the new edges appear as they were constructed by a malicious entity, often called the adversary in the online analysis.

The online analysis (see e.g. [ST85]) is a classical technique to measure a quality of such algorithms, i.e. those which have to deal with an incomplete input sequence, sequentially exposed to the algorithm. In our context, the sequence would be the configuration of edges the algorithm sequentially discovers. Certainly, the robot would construct a "better" walk (solution with a smaller cost), if it knew the graph beforehand (offline setting). Typically, bounds on the *competitive ratio* between the cost of an online and an optimal offline solution are studied. There are also different notions, like comparative ratio, penalty ect., used in the literature, but their essential idea matches the classical online analysis notion.

Rao et al. in [RKSI93] present an overview of one-robot algorithms navigating in an unknown environment with obstacles. The problem of reaching a given target by traveling through an unmapped terrain is also studied by Berman et al. in [BBF+96]. The goal is to minimize the ratio between the distance traveled by a robot and the minimal distance in a terrain in respect of the obstacles. They present a randomized algorithm that achieves a competitive ratio of $O(n^{4/9} \log n)$, which is less than the lower bound of $\Omega(\sqrt{n})$ for a deterministic algorithm (see Papadimitriou et al. [PY91]). The algorithm matching the deterministic lower bound is presented in [BRS91] by Blum, Raghavan and Schieber.

Unlabeled graphs

Exploration and navigation in graphs with unlabeled nodes and edges require different techniques and often expose much weaker efficiency results. By this class of problems there is much effort invested in obtaining a polynomial time exploration or just any bound on the exploration time. For instance, the classical exploration algorithms, like breadth-first search or depth-first search, are completely infeasible in this setting, as their run time turns out to be infinite. A robot cannot distinguish edges/nodes, and thus cannot "remember" visited places, which seriously weakens the progress of exploration. There are several techniques used, like homing sequences or artifacts put in the nodes, to recognize visited nodes.

Bender and Slonim present in [BS94] an algorithm for two robots which explores an unknown, strongly connected graph (with unlabeled nodes) in finite time (polynomial in number of nodes) using homing sequences, and furthermore, show how to achieve this by making the robots run the random walk. In the second solution, additional assumptions on the graph topology are made (high conductance of a graph) to assure the desired property of the random walk.

Bender et al. in [BFR+98] show that the graph can be explored in polynomial time using only one robot and a certain number of artifacts (pebbles), which can be placed or removed from a node in order to mark it. If the upper-bound on the total number of nodes n is known, one pebble suffices, otherwise $\Theta(\log\log n)$ pebbles are both necessary and suffice to completely explore the graph in a polynomial time. Comparing this to the result from [BS94], we might observe that an additional mobile robot is worth $\Theta(\log\log n)$ "static" pebbles. Fraigniaud et al. in [FIRT05] present the lower bound on the memory size needed by a robot with one pebble to explore an arbitrary graph on n nodes.

Even more restricted model is studied in [DFKP02], where storage capacity of a robot is bounded, i.e. it can store up to b bits in its local memory. While $b = O(\log\delta)$ bits are sufficient to traverse all edges of an arbitrary tree given to the algorithm, there are some trees for which an online algorithm (not knowing the tree beforehand) requires $\Omega(\log\log\log n)$ bits of memory. This also proves that no constant-size memory allows to explore (online) every tree. In more precise investigations, they show that this bound is even sharper, if we require a robot to stop after the tree is explored. It means that a robot has to be aware of the exploration status. In this setting, it requires $\Omega(\log n)$ bits of memory for some trees, and here the authors also conclude with the $O(\log^2 n)$-bit algorithm.

Labeled graphs

By a graph with labeled nodes, we enter another class of problems. In some settings, labels seem to be a very natural assumption – for instance, by the planar graphs. Betke, Rivest and Singh consider in [BRS95] an exploration of grid-like graphs populated by rectangular, axis-parallel obstacles obtained by removing some nodes from the grid. Each pair of obstacles cannot overlap or even touch one another. The so obtained graphs resemble a city map in appearance and thus are called city-block graphs. The exploring robot has to visit all edges, but it is required to return to the initial position every certain number of steps, say, for refuel (the piecemeal exploration). In [BRS95], authors observe regular structures (waves) the breadth-first search creates in this class of graphs. In the first presented algorithm, the robot mimics those waves without a large penalty for relocations. Their second algorithm is even simpler (dfs-like algorithm), but they claim it would be less suitable for generalization for arbitrary graphs. Both algorithms make $O(|E|)$ edge-traversals, and thus are optimal, up to a constant factor.

Beteke et al. in [ABRS99] continue investigations in a model, where a robot is enforced to periodical returns to its initial base station, but here in context of arbitrary, undirected graphs. Their algorithm traverses all edges in $O(|E|+|V|^{1.5})$ time or even $O(|E|+|V|^{o(1)})$ after some improvements. Although this still does not match the trivial lower bound of $\Omega(|E|)$, it improves a previous result of $O(|V|^2)$. Awerbuch and Kobourov in [AK98] use recursive framework of algorithm from [ABRS99], and use *sparse neighborhood covers* [ABCP93] to obtain a recursive algorithm exploring in time $O(|E| + |V| \log^2 |V|)$.

Finally, Duncan, Kobourov and Kumar in [DKK06] improve the result from [AK98], presenting an algorithm visiting all edges in time $O(|E|)$. This closes the open question from [ABRS99] and [AK98] about the linear time piecemeal exploration algorithm, even for the general graphs. Dessmak, Panaite and Pelc investigate in [PP98] and [DP02] an exploration of edges, but without the piecemeal assumption. They show an algorithm working in $O(|V| + |E|)$ steps, and claim that classical algorithms do not achieve this efficiency in the worst case.

An exploration of directed (strongly connected) graphs is studied by Deng and Papadimitriou in [DP99]. They observe that the time needed to traverse all edges depends on the number d of edges that have to be added in order to make the graph Eulerian (*deficiency*). The ratio between the number of traversals done by the algorithm and the optimal number of traversals is proved to be unbounded for an unbounded deficiency (ratio is $\Omega(d/\log d)$). They also consecutively show an algorithm which achieves the ratio of $d^{O(d)}$, and which does not require d to be known in advance. Albers and Henzinger in [AH00] improve this ratio and obtain the first sub-exponential result of $|V| \cdot d^{O(\log d)}$. However, this result is improved even further by Fleischer and Trippen in [FT05] who present an exploration that works in $O(d^8)$ steps.

Exploration of a continuous plane

A graph is not the only abstraction used to describe an environment in which robots can move. The exploration of a room bounded by rectilinear walls as a subset of a 2-dimensional plane is studied by Deng, Kameda and Papadimitriou in [DKP98] (see also Shermer [She92]). A robot enters a room by the *entry* point, and has to leave it using the *exit* point. The goal is to compute *a gallery tour* i.e. a path from which all sides of all obstacles can be seen (assuming an unbounded range of robot's vision). They study the ratio of the length of such a path to the length of the optimal path computed thanks to a map given in advance.

1.2.2 Multi-agent exploration

A cooperation of a team is studied as even more challenging than the single-robot scenario. A group of robots has potentially more exploration power, which is positively reflected in the efficiency of algorithms. As negative factors, there are additional coordination costs as well as difficulties by the cooperation, coming from a very local, and thus partial view of the current situation of each robot.

Cao, Fukanaga and Kahng in [CFK97] and Dudek et al. in [DJMW96] present a survey as well as a framework for the multi-agent systems. This includes models of communication topology, bandwidth, technological constraints as well as an overview of some multi-agent solutions. Halpern and Moses study in [HM84] the restriction on the propagation of local knowledge in a distributed system.

In the context of multi-agent systems, there are many classical *pattern formation* algorithms studied in the literature, where a team, dispersed in an environment, has to either form a circle [Kat05], polygons [SS96], gather in one point [SY96, SY99, FPSW01, KTI+07], or form a minimum path between two points [DKMS07, DKLM06, SS96]. The main problem here, is to break the symmetry coming from the model assumption that robots are not distinguishable (do not have IDs). There are various additional assumptions that can be made about the basic model. Katayama et al. [KTI+07] assume uncertain compasses, and Suzuki and Yamashita [SY96] or Prencipe in [Pre01b, Pre01a]) study asynchronous/semi-synchronous models.

A multi-robot graph exploration is studied by Fraigniaud et al. in [FIRT05], where a robot is modeled as a finite automaton. They prove that there exists a graph of size $O(k \cdot K)$, such that k robots (each modeled by a K-state automaton) fail to explore it. Das et al. in [DFNS05] consider the problem of obtaining a consistent labeling of a graph by a set of asynchronous, identical robots. The important aspect of this problem is to break the symmetry among robots.

Hsiang et al. present in [HAB+02] an algorithm for a large number of robots entering a graph through a selected node/nodes (a door), and filling the entire grid (populated by obstacles). The interesting assumption there is, that only one robot can occupy a node at a time. Also here, robots are modeled by a simple finite automate additionally equipped with local communication devices. The authors show the optimal algorithm for one-door case and also $O(\log(k+1))$-competitive algorithm for k-door case.

Fraigniaud et al. study in [FGKP06] an exploration of trees with uniform edges by a team of k synchronous robots starting in one node, aiming at the minimization of the total exploration time i.e. time for robots to jointly visit all nodes of a tree and return to the initial position. They prove (by reduction to 3-partition problem) that finding optimal walks of robots is an NP-hard problem. They make interesting observations on the influence of the communication patterns on the efficiency of the online exploration – its impact on the competitive ratio. They prove that if no communication granted, there is no cooperation at all. In the worst case, the whole team explores with the same efficiency as a single robot (competitive factor of $\Omega(k)$). By allowing global communication (or letting robots leave messages in the nodes), they improve the competitive factor to $O(k/\log k)$. Furthermore, they present the general lower bound of $2 - 1/k$ for the competitive factor of every exploration algorithm using k robots.

Dobrev et al. show in [DFPS02, DFK+02, DFS04, DFPS06] how a team searches for a malicious node (a black hole) which is a harmful host that destroys visiting robots, living no trace of such a destruction. They minimize the number of robots needed to fulfill this task as well as the number of edge traversals. This represents an interesting contribution to the problem of collective cooperation of robots.

1.3 Our contribution

We study the problem of collective graph exploration, where a team of k robots, positioned in a selected node of the graph, has to jointly visit all nodes of a graph and eventually return to its initial position. We investigate two settings – either the graph might be known to the algorithm in advance (*the offline model*) or the algorithm has no information about it and has to learn it progressively (*the online model*). We present algorithms as well as general bounds for the efficiency measure we introduce. Our results are new to this research area and have erected high interest in the community.

In the offline model, we observe how the optimal total exploration time is influenced by the graph topology, and we prove that it essentially depends only on the radius and the number of nodes in the graph. Furthermore, we present some easy approximations as well as methods for computing the optimal solution for trees.

In the online model, we study the competitiveness of the exploration algorithm, i.e. the ratio between the cost of the online and the optimal offline algorithm. We improve the results of Fraigniaud, Gasieniec, Kowalski and Pelc presented in [FGKP06] for the model minimizing the total exploration time (*time model*). Our significant general lower bound for the competitive ratio of an arbitrary algorithm proves that the exploration can be inefficient, if the graph is not given to the algorithm in advance. Furthermore, we present the algorithm for a restricted class of trees and show its competitive ratio which does not grow with a size of a team.

Moreover, we introduce a new cost function related to the energy used by a robot instead of the total exploration time. For this model (we call it *energy model*), we show both the general lower bound and efficient online algorithms. In case of our algorithm for trees, the competitiveness matches the general lower bound for this model (up to a constant factor). Finally, we introduce a framework which enables to explore arbitrary graphs, using tree-exploration algorithms.

Although we have made much progress in this field, there are still many interesting open questions asking to be answered. We try to state them in the concluding chapter of this work.

1.4 Work overview

In Chapter 2, we introduce the main model and the major problem as well as the notions and definitions used in this work. In Section 2.1 we discuss various communication models. In Section 2.2, we specify a model in which a graph is given to the algorithm in advance, before the exploration starts. We also introduce there two cost functions measuring the quality of a solution obtained by such algorithms. In Section 2.3, we show a model where the algorithm has to progressively learn the graph, as it is not given to it in advance. Furthermore, in Section 2.3.1, we define the measure (competitive ratio) that enables us to compare the efficiency of algorithms.

In Chapter 3, we investigate the offline model and the problem of precomputing the collection of walks. In Section 3.1 we make some basic observations on the optimal cost and in Section 3.1.1 we observe that an optimal exploration of graphs has a similar cost as an optimal exploration of their relaxed shortest-path trees. As finding the optimal solution is NP-hard, we present in Section 3.2.1 an easy 2-approximation for trees and the problem addressed in [FGKP06]. Furthermore, we show in Section 3.3 how to construct optimal walks for trees: in Section 3.3.1 by clustering the leafs in time $O(k^{n+1} \cdot n)$ and in Section 3.3.2 by using a dynamic programming in time $O(n^k)$.

In Chapter 4, we introduce a new cost model, in which we are interested in the maximal energy used by a robot rather than the total exploration time. In Section 4.1 we present

a general lower bound of 3/2 for the competitive ratio i.e. there exists no randomized algorithm which obtains the competitive ratio smaller than 3/2 (against the adaptive adversary). In Section 4.2 we present two algorithms collectively exploring trees: in Section 4.2.1 an 8-competitive and in Section 4.2.2 an improved, $(4 - 2/k)$-competitive algorithm. In Section 4.3 we show a framework allowing to use tree exploration algorithms to explore wider classes of graphs and in Section 4.3.1 we present an example of usage of this framework. We conclude in Section 4.4 with a simple modification of one-robot algorithm optimally exploring all edges of the graph enabling it to work in our multi-robot scenario.

In Chapter 5, we investigate more deeply the model where the algorithm's cost is the total exploration time. In Section 5.1 we significantly improve result from [FGKP06] and show the first non-constant general lower bound of $\Omega(\log k / \log\log k)$ for the competitive ratio of a randomized algorithm (against adaptive adversary). Furthermore, in Section 5.2, we present an online algorithm for *sparse trees*, i.e. those which each subtree contains a small number of nodes in comparison with the height. We prove that its competitive ratio depends on the density and the height of the tree, and for instance, for trees being a subset of a mesh we obtain $O(\sqrt{D})$-competitive algorithm. The advantage of our algorithm is that it uses strictly local communication, unlike the algorithm presented in [FGKP06], which uses global communication or writing in the graph.

Finally, in Chapter 6, we conclude this work and present an outlook for the future research as well as some interesting open questions.

1.5 Bibliographical notes

Most of the results presented in this thesis were published in a preliminary form as a conference papers [DKS06, DKMS06] and [DLS07]. Moreover, this dissertation contains results which, at the time of writing, are not yet published. A dynamic programing technique, lower bound of 3/2 and the preliminary version of 8-competitive algorithm published in [DKS06] are partially used in my master thesis [Dyn06] and in the improved form described consecutively in Section 3.3.2, Section 4.1, and in Section 4.2.1. We also show how the lower bound works for arbitrary, randomized algorithm against the adaptive adversary.

Basing on [DLS07] we present, in Section 4.2.2, an online algorithm and its analysis showing the competitive ratio of at most $4 - 2/k$. This is an improvement over the 8-competitive algorithm presented in [DKS06].

The general lower bound of $\Omega(\log k / \log\log k)$ for the competitive ratio of an arbitrary algorithm (in the time-related model) presented in Section 5.1 is published in [DLS07].

In [DKMS06], we have studied the problem of exploration of, so called, sparse trees. Section 5.2 bases on this publication, and contains a definition of density, a improved description of the online algorithm and its analysis showing a bound on the competitive ratio.

The framework enabling exploration of an arbitrary graphs i.e. the online tree selection scheme from Section 3.1.1 and observations in 4.3 as well as an example of its application for city-block graphs from Section 4.3.1, are still unpublished results. The same holds for the online algorithm from Section 4.4 exploring arbitrary graphs under the energy model.

CHAPTER 2

Preliminary definitions

We start with the general description of the problem studied in this work. First, we present some basic definitions and notions, and later in Section 2.3 and 2.2, two settings which let us derive (in Section 2.3.1) the notion of the competitive ratio – the main efficiency measure in this thesis.

We assume that explorers are placed in an environment which can be modeled as connected, undirected graph $G = (V, E)$ with $n + 1$ uniquely labeled nodes[1]. We are aware that it might be quite a challenging problem for a real robot to detect (using only its sensors) what is a "node" or an "edge" in a terrain, especially because it has to be a common and consistent definition over time and over all robots in a distributed team. Fortunately, there usually exists a mechanism allowing to distinguish edges and nodes using e.g. a stereo vision, GPS or compasses (see various compass models in [KTI+07]).

There are $k > 1$ autonomous robots which together form *a team*. Within a team each robot can be distinguished by its unique ID being a value between 1 and k, and therefore, we avoid many problems related to the indistinguishable robots (see e.g. [Kat05]). Each robot has the ability to change its location by moving to neighboring nodes along adjacent edges. Denote by $p_i(t) \in V$ the position of the i-th robot at time t and by $\mathcal{N}[v]$ the neighborhood

$$\mathcal{N}[v] := \{w \in V : (v, w) \in E\} \cup \{v\}$$

of node $v \in V$ in graph G. The robot can move only to a neighboring node i.e. at step $t + 1$ the robot occupies node

$$p_i(t + 1) \in \mathcal{N}[p_i(t)] \ .$$

The above assumptions determine that the robot is physically related to its position i.e. it cannot "teleport" itself within the graph and it has to traverse all edges along the path between its current position and the chosen destination. We assume that a robot

[1] Ironically, the reason why it is $n+1$, and not just n, is that we obtain a simplified notion (in some proofs), but also because the minimum spanning tree of G contains n edges.

entering some node v, receives information on the neighborhood of v, i.e. it "sees" all nodes connected with v by an edge and also reads unique labels of those nodes. This enables it to address the destination of its movement.

In *a round*, a robot may decide to stay in its current position or traverse an arbitrary edge attached to it. To keep the model really simple, we assume that rounds are fully synchronous and atomic operations. Therefore, an edge traversal cannot be interrupted, as this would result in some intermediate robot's position on the edge[2]. In our work, if a robot decides to traverse some edge, it has to traverse it completely, and it takes one step to do that (uniform edges).

There is one distinguished node $s \in V$ in graph G which serves as *a base station*. When the exploration starts, all robots are positioned there (i.e. $p_i(0) = s$ for all i), and after all nodes are jointly visited, the team has to finally return to s. To measure distances in the graph we denote by $d(v, v')$ the length (number of edges) of the shortest path between nodes $v \in V$ and $v' \in V$. For simplicity, denote by $d(v)$ the distance to the base station i.e. $d(v) = d(s, v)$. Radius D of the graph is the maximal possible distance in G from the base to the furtherest node, i.e.

$$D := \max \{d(v) : \ v \in V\} \ .$$

During the exploration a robot may visit one node many times, and thus elements of sequence $(p_i(0), p_i(1), p_i(2), \ldots)$ do not have to be distinct.

Definition 2.1 (A walk)**.** *A* walk *of the i-th robot is a finite sequence*

$$Walk_i := (p_i(0), p_i(1), p_i(2), \ldots, p_i(t-1))$$

of length $|Walk_i| = t$ of all consecutive positions of the robot, done during the exploration. We require that $p_i(0) = s$.

If $p_i(j) \neq p_i(j+1)$, then $(p_i(j), p_i(j+1)) \in E$, as $p_i(j+1)$ lies within the neighborhood of $p_i(j)$. Clearly, the walks of all robots in a team have the same length

$$\forall_{i,j} \ |Walk_i| = |Walk_j| \ .$$

We also need the notion of $Cover_i$ for a set of nodes visited by the i-th robot, not taking into account the order of those nodes or the number of visits of the robot to a particular node (like in $Walk_i$).

Definition 2.2 (A coverage of a robot)**.** *A* coverage *of the i-th robot is a set of nodes*

$$Cover_i := \{p_i(0), p_i(1), p_i(2), \ldots, p_i(t-1)\}$$

that the robot visits during the exploration of graph $G = (V, E)$.

[2] In some publications the intermediate position on an (weighted) edge is allowed (see e.g. [AB97]).

Given a robot's walk, the corresponding coverage is uniquely defined, but a coverage might be a result of many different walks. Finally, we notice that $|Walk_i| \geq |Cover_i|$, and unlike by the walks which are of the same length, it might happen that $|Cover_i| \neq |Cover_j|$ for some i and j.

Definition 2.3 (Collective graph exploration)**.** *We say that graph $G = (V, E)$ with base station $s \in V$ is* explored *by the algorithm using k robots initially positioned in s ($p_i(0) = s$), when*

$$\bigcup_{1 \leq i \leq k} Cover_i = V$$

where $p_i(|Walk_i|) = s$, and where $Cover_i$ is a coverage of the i-th robot.

We note that visiting all nodes in the setting described above is a different problem than traversing all edges of the graph (like in e.g. [DP02]). The crucial factor here is that the robots are aware of all labels of nodes in their neighborhood, and thus edges connecting to the already visited nodes do not have to be traversed at all. Furthermore, in the context of the terrain exploration, visiting nodes seems to be more appropriate than traversing all edges. We are interested in visiting all "locations" at least once, and not in approaching each location from any possible direction.

2.1 Communication models

Robots are autonomous entities distributed over the graph, and in order to cooperate they have to be able to exchange information. Each robot has only a very local view of the topology of the graph, but as it moves through the graph, it progressively learns more about the graph's topology. It can store information gathered in this process and use it to adjust the appropriate current exploration strategy. The local knowledge of a robot can be extended by explicit communication with other team members. As they might have visited distinct regions, the information they carry might represent an important contribution to decision making process and adaptation of a local strategy to the current situation in the system (see [HM84] for elaboration on distributed local knowledge).

Each robot runs the same algorithm, and assuming that strategies are deterministic, the only reason for two robots to communicate is essentially to merge their local maps. In the case of randomized algorithms, where the random events influence local decisions, there might be a need to send additional information.

There are many communication patterns which might be considered in our context (see [DJMW96, CFK97] for a taxonomy of multi-agent systems), but we mention here only a few the most interesting ones:

- **No communication.** There is no communication granted, and therefore all common information possessed by robots is essentially only their algorithm which is the same for all robots.

- **Local communication.** A robot can exchange information with another robot positioned within some predefined-size neighborhood i.e. two robots positioned respectively in v and v' communicate only, if $d(v, v')$ is smaller than some fixed constant. This model is motivated by the wireless communication devices which have bounded communication radius. For the simplicity, we assume a zero-size neighborhood – only robots positioned in the same node can communicate.

- **Writing in graph.** Robot can put some data in a node and also modify the data which was stored there before (assume that there is a white board placed in each node). Other robots entering a node can read the data stored in a node and in that way extend their local information.

- **Global communication.** Each robot can communicate with an arbitrarily chosen robot at no cost. It results in global knowledge shared by a team. New findings of one robot are directly visible to a whole team.

Global communication represents a complete freedom of exchanging information, what essentially results in a common knowledge of the system state and opens the possibility to make a centralized computation. Given global communication, it is easy to simulate all remaining models, and thus this specific model is the strongest in our setting.

Writing in the graph allows to easily emulate local communication – robots placed in the same node can write information on the common white board instead of just sending messages. However, given local communication pattern it might be difficult to simulate writing in the graph. This is because by local communication the presence of a robot is required, unlike in writing in the graph, where information is just left behind without worrying about who and when is going to read it.

Many problems expose different complexity, when assuming different communication patterns. For instance, Fraigniaud et al. show in [FGKP06] that the exploration without communication leads to trivially high cost of the algorithm. A collective exploration is highly inefficient, if robots do not communicate. Here we show the results for local and global communication, but there is a challenging open question, if and how the selected pattern influences the efficiency of the collective exploration or even more general team cooperation problems.

2.2 Offline exploration

The main assumption of *the offline model* is that the environment is known to an exploring algorithm in advance. The algorithm (we call it *offline algorithm*) receives a graph as a fixed input, before the exploration even starts. This extra information can be used to disperse a team in the graph more accurately, as it is known which parts of the graph require more exploration power.

Clearly, an exploration in the offline model loses pretty much its flavor as the map is already there, so the main exploration goal is already fulfilled. Nevertheless, in Chapter 3 we show some applications to this problem. Moreover, basing on the offline cost, we are able to measure the algorithm's efficiency in a more sophisticated model, described later in Section 2.3.

We propose a fully synchronized approach consisting of two types of rounds. Before the exploration, at the time when all robots are positioned in the base and no robot has moved so far, the distributed algorithm (each robot) obtains graph $G = (V, E)$ as an input. The goal for the team is to jointly visit all nodes of G, and therefore appropriate walks have to be computed. All robots are positioned in the same node (base station s), and thus all information needed by the computation can be exchanged by messages sent within the team, so it can easily agree upon the exploration strategy. It might also happen that one robot (a leader) computes everything centrally, and then the results are sent to all team members. Eventually, the robots' walks jointly covering V are constructed, and the i-th robot stores (in a local memory) its predefined walk $Walk_i$. An interval of initial computation and communication is called a COMPUTE round.

When the walks are precomputed, robots leave the base station, disperse in the graph and jointly visit all nodes, according to the predefined schedule defined by $Walk_i$. In a MOVE step, each robot reads the appropriate element of $Walk_i$ and selects it as the destination for this round. Then all robots traverse selected (uniform) edges in a synchronous way, and at the end of the step all robots finally reach their destination. There is no need for robots to communicate before or during this step, as at this time there is no information which might influence the process of the exploration. All information is already used in the initial COMPUTE round and taken into consideration by the precomputation of walks.

Both rounds mentioned above are shown in Figure 2.1 presenting COMPUTE-MOVE-MOVE model, where there is only one initial round of communication and computation and many rounds where robots move without passing further messages or local computation.

In our model, we neglect both the time needed for this computation and message passing in COMPUTE round. Therefore, there might be an arbitrary much time invested by the algorithm to compute the solution which is arbitrarily close to (or equal) the

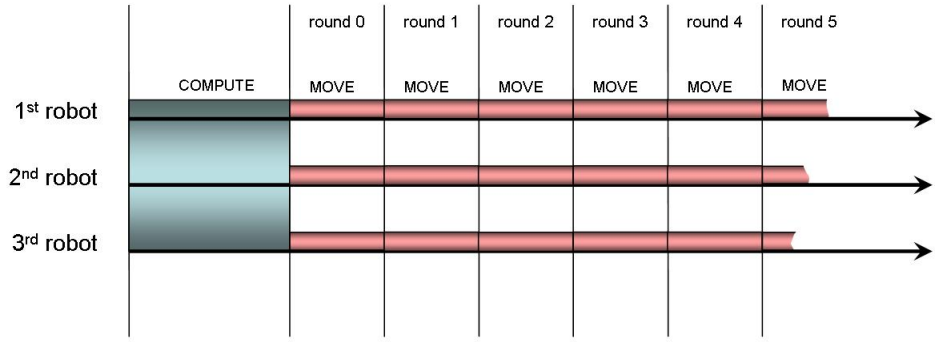

Figure 2.1: Centralized COMPUTE and distributed MOVE rounds.

optimal one. We are aware that e.g. NP-hard problems can be solved here, or exponential time algorithms applied. However, we are more interested in observing the team general cooperation abilities, reflected in movements of its members, than in the complexity of the computation needed to obtain the walks.

Assume that the team running distributed, offline algorithm *ALG* has jointly explored graph G. The **offline exploration cost** is defined as the total number of exploration rounds

$$\widetilde{C}_{ALG}(G, s) := |Walk_1| ,$$

where, we recall, all sequences $Walk_i$ are of the same length. For given graph G and base station s, term

$$\widetilde{C}_{OPT}(G, s)$$

denotes the optimal cost that is achieved by an offline exploration algorithm.

We would like to observe a cooperation of the team during the collective exploration of G and the increase in the efficiency of algorithms in comparison to a one-robot scenario. The main goal for the team is to fairly disperse in G, so that the work load is balanced among the robots. Since there are many robots, there might be many edges which are traversed in parallel. In a perfect situation, k cooperating robots working simultaneously traverse k distinct edges, which has a positive impact on the progress of exploration.

Chapter 3 is entirely devoted to the offline model. We make the observation on the bounds of this cost for a given graph, and also discuss the problem of computing the optimal (or approximate) solution to the offline exploration problem.

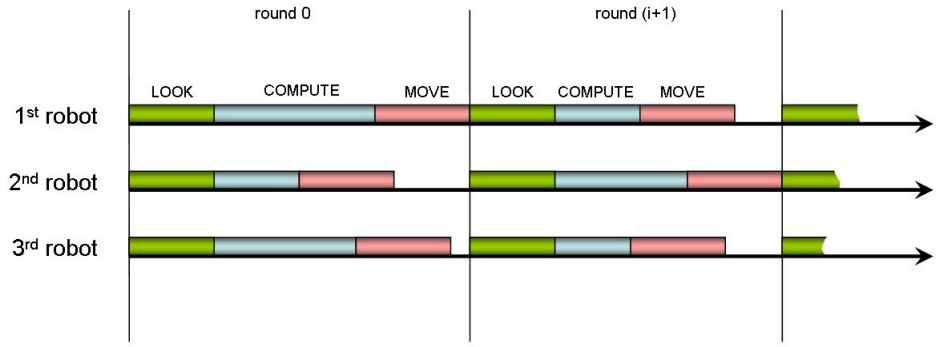

Figure 2.2: Synchronized LOOK-COMPUTE-MOVE rounds.

2.3 Online exploration

Unlike in the offline model, in the *online model* the graph is not known in advance, and therefore the robot's walks cannot be precomputed. Each robot running the *online algorithm* has to construct its walk progressively while learning the graph. As the exploration begins with a team positioned in base s, robots have to spread in order to jointly visit all nodes of the graph. There might be some parts of the graph which require different "exploration power", depending on the complexity of the subgraph. As each traversal of an edge brings a unit cost, a careful dispersal of robots let avoid an unnecessary and costly robots relocation. The selection of an appropriate communication model (see Section 2.1) plays an important role in the propagation of knowledge in such a distributed setting. Robots have to exchange discovered information on the graph's complexity in order to work efficiently and adapt exploration strategy.

In the online model, each robot follows the LOOK-COMPUTE-MOVE scheme (see e.g. [Pre01b, SS96]). Figure 2.2 shows an example of initial rounds of a distributed algorithm for three robots. At the beginning of round t, "sensors" of a robot are activated (LOOK), and the i-th robot positioned at $p_i(t)$ gets information on neighborhood $\mathcal{N}[p_i(t)]$ in graph G. There might be some nodes which are already known to the robot, but maybe also some which are completely new to it or even to the team. The whole information on the graph, revealed to the online algorithm at time t, can be described as

$$\sigma_t = (\mathcal{N}[p_1(t)], \mathcal{N}[p_2(t)], \ldots \mathcal{N}[p_k(t)]) \ ,$$

but, depending on the communication model (see Section 2.1), the information is differently distributed among the robots.

Denote by $K_i(t)$ the set of nodes that the i-th robot knows at time t. Clearly, the robot also knows the graph induced by $K_i(t)$, and therefore we say that $K_i(t)$ is a *knowledge* of

the i-th robot at time t. If there is no communication, then the knowledge of the i-th robot

$$K_i(t) = \{\mathcal{N}[p_i(t')] \mid t' \leq t\} .$$

For local communication, we have $K_{(i,1)} = \{s\}$ for all i, and

$$K_i(t) = K_i(t-1) \cup \bigcup_{i' \in I} (K_{i'}(t-1) \cup \mathcal{N}[p_{i'}(t)]) ,$$

such that $I = \{i' \mid p_{i'}(t) = p_i(t)\}$ is a set of all robots positioned in $p_i(t)$ at time t. The knowledge of two robots that are positioned in the same node at time t is merged so that each of them extends its local view.

After the information about the neighborhood is revealed and exchanged within the team (with respect to the communication model), each robot is allowed to run local computation (COMPUTE), basing on the knowledge $K_i(t)$ it posses and maybe on some random bits in case of a randomized algorithm. Basically, this results in an update of local state of the robot and computation of the destination node into which the robot will move. Finally, the robot moves (MOVE) toward the computed destination. Note that it might point out its current position as a destination node, and thus decide not to move in the current round. As mentioned before, the robot cannot jump within the graph as it can move only to a neighboring node.

At some step, it can happen that all nodes contained in the elements of the sequence

$$\sigma = \sigma_1, \sigma_2, \sigma_3, \ldots$$

are already visited by some robot. At this point of time, all neighborhoods exposed to the algorithm contain only visited nodes. If additionally all robots are positioned in the base, all robots can compare their knowledge and find out that (according to Definition 2.3) the exploration is accomplished. At this time, the complete sequence σ defines graph G which has been explored by algorithm ALG. The walks of robots are defined and now we can measure the efficiency of the offline exploration. The cost of the solution can be measured in the two following models (further investigated in Chapter 5 and Chapter 4):

- **energy model** – the cost is the maximal "energy" used by a robot (the maximal number of edges traversed by a robot until the end of the exploration)

$$\mathcal{E}_{ALG}(G, s) := \max_{1 \leq i \leq k} \left| \{0 < t \leq |Walk_i| : p_i(t-1) \neq p_i(t)\} \right| ,$$

- **time model** – the algorithm's cost $C_{ALG}(G, s)$ is the total time of the exploration (number of MOVE rounds)

$$C_{ALG}(G, s) := |Walk_1| .$$

Notice that $C_{ALG}(G,s) \geq \mathcal{E}_{ALG}(G,s)$ for each algorithm ALG and each graph G with selected node s. Therefore, the algorithm with a small cost in the time model has at most the same cost in the energy model.

Finally, we can also consider the problem of constructing the walks of a robot basing on the knowledge of graph G (see offline problem in Section 2.2). The optimal *offline algorithm OPT* constructs the walks with the optimal cost of $\widetilde{C}_{OPT}(G,s)$. Clearly, we have $\widetilde{C}_{OPT}(G,s) \leq C_{ALG}(G,s)$ and $\widetilde{C}_{OPT}(G,s) \leq \mathcal{E}_{ALG}(G,s)$ for every online algorithm ALG, graph G and node s. We will later see (in Section 4.1 and Section 5.1) that for some graphs and selected base station online and offline costs (for both energy and time model) differ significantly.

For optimal offline algorithm OPT all three cost measures define the same function, i.e.

$$\widetilde{C}_{OPT}(G,s) = C_{OPT}(G,s) = \mathcal{E}_{OPT}(G,s) \,.$$

The first equality holds even for an arbitrary offline algorithm, as the only difference in the definition of those functions concerns the algorithm's knowledge on the graph (online/offline setting). To show the second equality, observe that the most exhausted robot has done $\mathcal{E}_{OPT}(G,s)$ moves. It takes $\mathcal{E}_{OPT}(G,s)$ steps to do $\mathcal{E}_{OPT}(G,s)$ moves, and therefore the optimal total exploration time is at most $\mathcal{E}_{OPT}(G,s)$. As we have $C_{ALG}(G,s) \geq \mathcal{E}_{ALG}(G,s)$ for every algorithm, we obtain $C_{OPT}(G,s) = \mathcal{E}_{OPT}(G,s)$.

2.3.1 Online analysis and competitive ratio

We decided to use a notion of the classical *competitive analysis* (see e.g. [BEY98, ST85]) in the context of our online model. Typically, the online model assumes that there is some input sequence to which the online algorithm has to react. For instance, in classical request-answer games the sequence contains requests exposed to the algorithm, which have to be processed before the next request can be exposed. In the collective graph exploration problem, the configuration of edges discovered by an algorithm can be seen as an online input sequence. However, here we do not require that all discovered nodes have to be visited before the next "request" comes. We only say that all request have to be processed in order to accomplish the exploration.

In our setting, the graph to be explored is exposed gradually by sequence σ, during the execution of the online algorithm. At time t the algorithm gets element σ_t which describes how the neighborhoods of current robots' positions look like. We assume that the sequence is controlled by a malicious authority, called the *adversary*, which aims at creating the worst case setting, and thus increasing the cost of an online algorithm.

Adversary models

We consider adversaries which demonstrate different power and model different worst-case scenarios. We have adapted the adversary models presented in [BEY98] to a specific characteristics of our setting. Some of the classical models do not apply to the collective exploration scenario[3], and formal definitions of other models are slightly changed.

We distinguish two adversarial models, *oblivious* and *adaptive*[4] adversary, that differ by the time at which they define the online input sequence σ. In both models the online algorithm *ALG* is known to the adversary. The *oblivious adversary* explicitly defines graph G to be explored, just before the exploration starts. This defines all neighborhoods, which is the essential information contained in $\sigma_t = (\mathcal{N}[p_1(t)], \mathcal{N}[p_2(t)], \ldots \mathcal{N}[p_k(t)])$. Therefore, later at each time during the online exploration, the adversary can expose the neighborhood of each node containing a robot.

Note that in a classical oblivious adversary model, the input sequence (and not the structure which allows to construct the input sequence) is fixed beforehand. This is not possible in our setting in the case of a randomized algorithms. For any fixed σ it might happen that at some time t, σ_t exposes a neighborhood of a node which is not occupied by any robot.

The *adaptive offline adversary* defines the sequence gradually, while observing the behavior of a (randomized) online algorithm. It constructs element σ_t exactly at time t, when it knows the current distribution of robots. In the case of randomized algorithms, this is more powerful adversary than the oblivious adversary, as it knows also random decisions made by the algorithm up to time t. It can construct a setting that is much difficult for current the condition of a team. Graph G, which is finally constructed, depends on random algorithm *ALG*, and therefore algorithm cost as well as optimal offline cost $\widetilde{C}_{OPT}(G, s)$ are random variables.

By the deterministic algorithm, both models represent exactly the same power, because even the adaptive adversary could define its online sequence before the exploration. Our results include exclusively deterministic algorithms. However, we have also shown interesting lower bounds for the efficiency of the randomized algorithms against the adaptive adversary.

Competitive ratio

The *competitive ratio*[5] reflects the ability of an online algorithm to deal with an unpredictable input sequence.

[3] The *adaptive online adversary* model is not feasible in our setting.

[4] Adaptive adversary corresponds to the *adaptive offline adversary* in a classical theory.

[5] There are also many other, but similar terms used in the literature concerning the graph exploration (e.g. a penalty in [PP98] or an overhead in [DP02, FGKP06]).

Assume some adversarial model and the online algorithm ALG that have explored graph G from base s with cost $C(G,s)$. We have either: $C(G,s) = \mathcal{E}_{ALG}(G,s)$ for the energy-model, or $C(G,s) = C_{ALG}(G,s)$ for the time-model, depending on the cost model we consider. Moreover, we denote by $\widetilde{C}_{OPT}(G,s)$ the optimal offline cots on the same graph G with base station s. We would like to compare the online algorithm's cost and the optimal offline cost. The following definition comes from [BEY98], but is a bit more restricted.

Definition 2.4 (The competitive ratio)**.** *Randomized online algorithm ALG is* c-competitive*, if for every graph* G *and base* s *(constructed by the adversary)*

$$C(G,s) \leq c \cdot \widetilde{C}_{OPT}(G,s) \,,$$

where $C(G,s)$ *is the cost of ALG on graph* G*. The parameter* c *is* a competitive ratio *of ALG.*

In a classical definition, the expected value of costs is taken into the consideration. Here, by omitting the expectations, we use even a stronger notion of competitiveness.

The competitive ratio measures how well an algorithm deals with the fact that the graph topology is not known in advance. It might happen that the online algorithm has a large cost, but it might still be acceptable, if the optimal offline algorithm has a large cost too. It would just mean that the complexity of this particular graph is high, and it is difficult to explore it anyway. On the other hand, if the offline cost appears to be much smaller than the cost of an online algorithm, it means that the algorithm is incapable of dealing with the exploration of an unknown environment.

Algorithms which achieve small competitive ratio are for us of a higher quality, as they better deal with the exploration without knowing the graph. In the next chapters, we show, that in general, the problem generates some bounds for the competitive ratio. In Section 4.1, we present a lower bound for the competitive ratio in the energy model, and in Section 5.1, we show even stronger lower bound for the time model.

CHAPTER 3

Collective offline exploration

In the offline exploration model (introduced in Section 2.2), the graph is known to the algorithm in advance, before the exploration starts, and when all robots are positioned in s. Therefore, the robots' walks can be easily computed beforehand, so that the robots do not have to compute/modify them during the exploration (the COMPUTE-MOVE-MOVE scheme). Then each robot follows its precomputed walk synchronously with other team members, and the team jointly visits all nodes of the tree.

As all walks $Walk_i$ constructed by the algorithm are of the same length, we define the cost of such a solution as

$$\widetilde{C}_{ALG}(G,s) := |Walk_1| \,.$$

The optimal offline cost $\widetilde{C}_{OPT}(G,s)$ is a lower bound on this cost.

As we have mentioned in Section 2.2, we neglect the time spent on computation and message passing. Therefore, the algorithm may carry out even a time consuming computation of the optimal solution. However, we will later see that this is an NP-hard problem to compute the optimal solution, and thus we still put much effort to show how to make the computation efficiently (even at the expense of accuracy).

Definition 3.1 (Approximation)**.** *Algorithm ALG is an α-approximation to the collective offline exploration problem, if for all G and s*

$$\frac{\widetilde{C}_{ALG}(G,s)}{\widetilde{C}_{OPT}(G,s)} \leq \alpha \,.$$

The parameter α is an approximation ratio *of the algorithm ALG.*

We measure the quality of the solution delivered by an offline algorithm by observing a bound on its approximation ratio $\widetilde{C}_{ALG}(G,s)/\widetilde{C}_{OPT}(G,s)$. Clearly, the optimal solution (approximation ratio of 1) can be found in exponential time, and we show how to do it for trees using the dynamic programming technique. We also show that the solution

can be computed more efficiently, but its cost (and thus the approximation ratio) will be increased.

Certainly, the exploration of a known graph is not as challenging as the exploration of an unknown graph, where the "map" has to be constructed, but still there are many interesting applications for the offline case. Consider the situation where the robots' walks have to be frequently computed for different graphs. There are many contributions investigating problems like e.g. mail or parcel pickup and delivery, scheduling of automated guided vehicles (vehicle routing problem, multi-vehicle scheduling problem), or other basic logistic problems. In these settings, the problem gains a new interesting context. Graph G is known, but there is a need for a collection of optimal walks covering a subgraph of G. A fair decomposition of nodes into subgraphs of almost equal size has to be found. This has to be computed efficiently, and also the solution is required to be as close to the optimal one as possible.

Moreover, by the offline model we give a base for further online investigation in Chapter 4 and Chapter 5. The main outcome here is information on the optimal cost for a given graph expressed in terms of number of nodes and the radius of the graph (see Section 3.1). This will serve us as the value to which we compare the efficiency of online algorithms. In Section 3.2, we present a constant approximation which we will additionally extend in Chapter 4 to obtain an online algorithm working without a graph given in advance. We close this chapter by showing a way to find an optimal solution for trees (see Section 3.3).

3.1 The optimal cost

Cost $\widetilde{C}_{OPT}(G, s)$ of an optimal offline algorithm is a lower bound for the cost of online algorithms in the energy and the time model. We find a good lower bound on the optimal cost and use it later by estimations of the competitive ratio of online algorithms. Two easy lower bounds are presented and then combined into a bound which (as we will see in Section 3.2) is up to a constant factor tight.

Given graph $G = (V, E)$ with the distinguished base $s \in V$ where $|V| = n + 1$, denote by D the radius of G, i.e. distance from s to the furtherest node. Assume that all robots start from the base, and in each step each robot visits a new node – never visited by any robot before. The whole team discovers k such nodes in a step, and even then (as there are n nodes besides the base) the exploration takes at least $\lceil n/k \rceil + 1$ steps. Since the team cannot jointly discover more than k new nodes per step, it results in the first lower bound:

$$\widetilde{C}_{OPT}(G, s) \geq \lceil n/k \rceil + 1 \, .$$

To obtain the second bound, observe that there is a node in G which lies at a distance D from base s. One robot has to visit this node and return to s afterward. Even if it does it directly, without any detours, it takes at least $2D$ steps. The exploration will not be completed until this robot returns to s, and therefore

$$\widetilde{C}_{OPT}(G,s) \geq 2D \ .$$

We combine our two simple lower bounds to obtain a general lower bound for the cost of the optimal collective graph exploration

$$\widetilde{C}_{OPT}(G,s) \geq \max\{2D, \lceil n/k \rceil + 1\} \ .$$

In this work, we present algorithms exploring trees, and thus we also make here some observations for this class of graphs. A tree T of $n+1$ nodes, rooted at s, contains n edges, and D is the distance from s to the furtherest leaf. Since the exploration ends when all robots return to the root, and as each path in the tree is unique, if a robot traverses an edge once, it has to traverse it a second time, returning to the base. Altogether, there are $2n$ traversals which have to be made by k robots. It takes at least $\lceil 2n/k \rceil$, and thus for trees

$$\widetilde{C}_{OPT}(T,s) \geq \lceil 2n/k \rceil \ .$$

This kind of exploration is a great example of a perfect group cooperation. The tasks are fairly distributed among all members, which are able to execute them using a full power of parallel processing. The lower bound of $2D$ holds also for trees where one robot has to visit a distant node. Here no cooperation is possible, since other robots cannot help visiting this leaf sooner. Combining these two simple bounds, we obtain a slightly better general lower bound for trees

$$\widetilde{C}_{OPT}(T,s) \geq \max\{2D, \lceil 2n/k \rceil\} \ .$$

In Section 3.2, we present an algorithm exploring an arbitrary graph G and obtaining the cost of $O(D + n/k)$, and therefore $\widetilde{C}_{OPT}(G,s) = O(D + n/k)$, matching the lower bound (up to the constant factor). This proves that the cost of an optimal algorithm exploring graph G is related only to D and n, and does not significantly depend on the actual topology of G.

Theorem 3.2. *If $\widetilde{C}_{OPT}(G,s)$ is the optimal cost of the offline exploration of graph $G = (V,E)$ with base $s \in V$ by a group of k robots, then*

$$\widetilde{C}_{OPT}(G,s) = \theta\left(D + n/k\right) \ ,$$

where D is the maximal distance of a node from base s and $n+1$ is the number of nodes in G.

3.1.1 A relaxed SPT and the cost of its optimal exploration

Assume that we have some spanning tree of G. If robots, moving only along tree edges, visit of all nodes of the tree, they also completely explore graph G. We show that the offline exploration of a spanning tree with a certain property is of roughly the same complexity as the exploration of G.

Definition 3.3 (γ-relaxed SPT)**.** *For given graph G and selected node s, a γ-relaxed shortest path tree is a spanning tree of G rooted at s, which is at most $\gamma \cdot D$ in height, where D is the radius of G.*

The γ-relaxed SPT T_γ of G contains all nodes of G, and therefore the exploration of T_γ leads to the exploration of G. However, the optimal offline algorithm for G might traverse some edges which are not in T_γ. Therefore, even the optimal algorithm for T_γ might have an increased cost. Certainly, there is

$$\widetilde{C}_{OPT}(G, s) \leq \widetilde{C}_{OPT}(T_\gamma, s) \,,$$

but also another, more interesting bound can be shown here.

Theorem 3.4. *Let T_γ be a γ-relaxed SPT of G with base s. We have*

$$2\gamma \cdot \widetilde{C}_{OPT}(G, s) \geq \widetilde{C}_{OPT}(T_\gamma, s) \,,$$

where $\widetilde{C}_{OPT}(G, s)$ is the cost of the optimal offline solution for graph G and base s.

Proof. In the previous section, we have observed the following lower bound for the cost of the optimal offline algorithm:

$$\widetilde{C}_{OPT}(G, s) \geq \max\{2D, \lceil 2n/k \rceil\} \,.$$

The maximum of two numbers is not smaller than its arithmetic mean, and thus we obtain $\widetilde{C}_{OPT}(G, s) \geq n/k + D$. In Section 3.2.1, we present the algorithm which explores an arbitrary tree by cost $2n/k + 2D \cdot (1 - 1/k)$ (see Theorem 3.5). The optimal algorithm has no greater cost, and therefore

$$2n/k + 2(1 - 1/k) \cdot \gamma \cdot D \geq \widetilde{C}_{OPT}(T_\gamma, s) \,.$$

Using the two bounds presented above and $\gamma \geq 1$, we obtain

$$2\gamma \cdot \widetilde{C}_{OPT}(G, s) \geq 2\gamma \cdot (n/k + D) \geq \widetilde{C}_{OPT}(T_\gamma, s) \,,$$

which concludes the proof. ■

We have made some observations on the cost of the optimal solution, without knowing how this solution might look like. In Section 3.3, we show how to compute the optimal walks for k robots.

3.2 Efficient approximations

The problem of finding an optimal solution for the collective offline exploration problem is proved to be NP-hard. In 1978, Fredericson, Hecht and Kim (see [FHK78])) showed a reduction to k partition problem, and Fraigniaud et al. [FGKP06] proved that the problem remains NP-hard, even if restricted to trees. In [FHK78] Frederickson et. al. present $(5/2 - 1/k)$-approximation for k-traveling salesmen problem (k-TSP) on a weighted, undirected graph. They base on the good tour for $k = 1$ (1.5-approximation algorithm for 1-TSP introduced by Christofides, Edmonds and Johnson), which is later split into k subtours.

Averbakh and Berman consider k-TSP in trees and in [AB96] they give a heuristic for $k = 2$. They prove the worst case approximation factor of 2 (which is a highly inefficient result for this setting). In [AB97] they extend their results for trees and present an approximation for an arbitrary k, where both an initial location and k tours have to be computed. They achieve the worst case approximation factor of $(2 - 2/(k + 1))$, but the time needed for computation is $O(k^{k-1}n^{k-1})$. In [AB02] they study the special case of $k = 2$ and $k = 3$, where they find (in polynomial time) only the optimal starting location of the robots (without computing their optimal tours).

Nagamochi and Okada improve the time needed for the computation of k-TSP tours to $O((k - 1)! \cdot n)$, fulfilling the same as Averbakh and Berman approximation guarantee of $(2 - 2/(k + 1))$. Please notice that both of these results need time exponential in k.

Sinha et al. in [EGK+04] cover all nodes of a weighted graph with a set of k trees rooted in k predefined, distinct nodes. Their polynomial algorithm achieves the worst case approximation factor of 4. Arkin, Hassin and Levin in [AHL06] also deliver a few approximations for a weighted graph, in the context of vehicle routing (4-approximation for node-cover with open-paths, (3+e,3+e)-approximation for star-covers, and 4-approximation for unrooted trees).

3.2.1 An easy $O(1)$-approximation

In Section 3.1 we have observed the lower bound on the cost of an optimal offline algorithm. To show a tight bound (up to a constant factor), we present (basing on [FHK78] and [DKS06]) an easy, constant approximation algorithm for arbitrary graphs. Consider undirected, connected graph $G = (V, E)$ on $n + 1$ nodes with base station s. The shortest path tree $T = (V, E')$ of G rooted at s can be easily computed in polynomial time, using e.g. the Dijkstra's algorithm (see [CLRS90]). We show how to explore this tree by a team of k robots in $O(D + n/k)$ parallel rounds. Once T is explored, all $n + 1$ nodes are visited by at least one robot, which means that also graph G is fully explored.

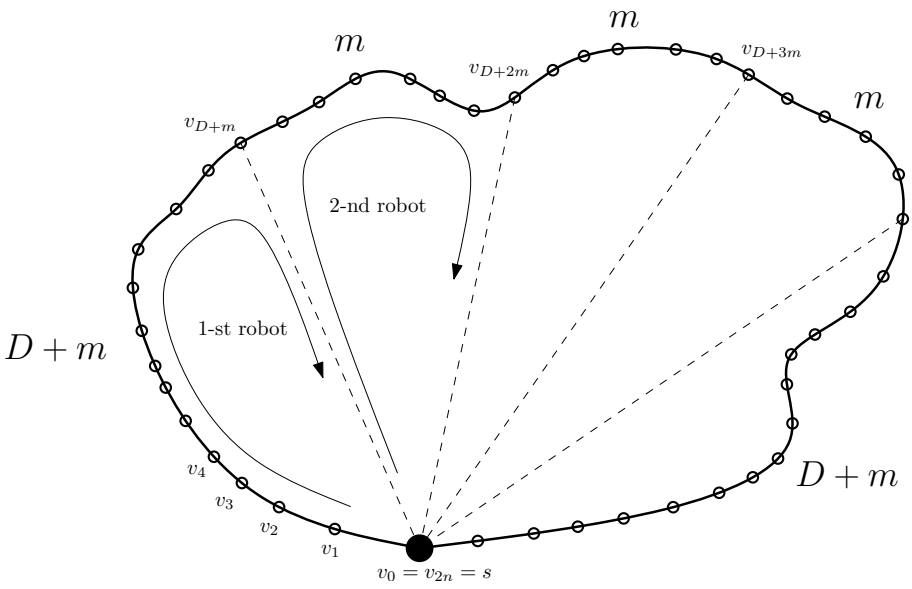

Figure 3.1: Dfs-sequence of a tree.

Tree T consists of n edges, and if D is the distance to the furtherest node in G, then the furtherest leaf in T is at distance D from base s (T is the shortest path tree of G). We construct k walks (each containing node s) that jointly cover all nodes of T and no walk is longer than $2D \cdot (1 - 1/k) + 2n/k$.

First, we compute a dfs-sequence (see the textbook [CLRS90]) of T represented by a sequence of nodes $v_i \in V$ for $0 \leq i \leq 2n$, such that $v_0 = s = v_{2n}$, $(v_i, v_{i+1}) \in E$ and $\bigcup v_i = V$. Figure 3.1 shows an example of such a sequence. There are some nodes which appear in this sequence many times, and therefore in Figure 3.1 there are presented separately.

Let $m := 2n/k - 2D/k$ and $w_0 = v_{D+m}, w_2, \ldots w_l = s$ be a simple path from v_{D+m} to s ($l \leq D$ as no node lies further than D). We define $Walk_1(t)$ of the first robot by giving explicitly its consecutive positions $p_1(t) := v_t$ for $0 \leq t \leq D + m$ and $p_1(t) := w_{t-D-m}$ for $D + m < t \leq D + m + l$. The length of the walk created in this way is at most

$$D + m + D \leq 2n/k + 2D \cdot (1 - 1/k) .$$

The i-th robot ($1 < i \leq k - 1$) goes directly to node $v_{D+i \cdot m}$, then continues to visit nodes $v_{D+i \cdot m+1}, v_{D+i \cdot m+2}, \ldots v_{D+i \cdot m+m}$, and finally returns to the base station using a simple path. We have always the same bound of $2n/k + 2D \cdot (1 - 1/k)$ on the length of such a walk. The last robot does not return to the base using a simple path, but just continues the dfs-traversal until the end of it.

There are $k - 2$ robots which extend the dfs-sequence at least by m steps, and thus in total, they traverse $m \cdot (k - 2)$ dfs-edges. The first and the last robot extend the sequence

by $D + m$, and thus the team traverses

$$m \cdot (k-2) + 2 \cdot (D+m) \geq 2n$$

edges of the dfs-sequence. This means, that all nodes of the whole tree T are jointly visited by robots and the latter finish their walks in base station s. In this way, graph G is also explored in time which can be easily bounded by $2n/k + 2D \cdot (1 - 1/k)$.

Theorem 3.5. *In the offline exploration model an arbitrary graph on $n+1$ nodes, and D in height can be explored by a team of k robots in time $2n/k + 2D \cdot (1 - 1/k)$.*

As we have observed in Section 3.1 for trees, the optimal offline cost $\widetilde{C}_{OPT}(G,s) \geq 2D$ and $\widetilde{C}_{OPT}(G,s) \geq 2n/k$. Therefore, the above algorithm is $(2-1/k)$-approximation for the collective exploration problem for trees. Considering arbitrary graphs, we have slightly weaker bounds on the cost of the optimal solution (see Section 3.1): $\widetilde{C}_{OPT}(G,s) \geq \lceil n/k \rceil + 1$ and $\widetilde{C}_{OPT}(G,s) \geq 2D$. This proves the approximation factor of $(3 - 1/k)$ for arbitrary graphs.

Lemma 3.6. *There exists a polynomial $O(1)$-approximation for the collective graph exploration problem.*

3.3 An optimal solution for trees

Given arbitrary tree T, we show how to compute the collection of k robots' walks, each starting and terminating in the root of T, jointly covering all nodes of T and minimizing the length of the longest walk. In Section 3.3.1 we make some observations, which enables us to compute an optimal solution in time $O(k^{n+1} \cdot n)$. However, if the number of nodes is large, comparing to the team size, this result might become unacceptable. Especially, when we think of the problem of constructing an optimal solution for, say, $k = 3$, this method does not seem to be an efficient one. In Section 3.3.2, we show how to obtain the optimal solution in time related to n^k rather than k^n.

3.3.1 Optimal solution using compact walks

First, observe that each closed walk of a robot, containing the root of T, can be represented by a subtree $T' \subseteq T$ obtained by taking all nodes and edges visited or traversed by this walk. We say that the walk is *compact* in this tree, if all leaves of subtree T' are also leaves in T. The collection of k compact walks is of *size* $\vec{a} = (a_1, a_2, \ldots, a_k)$, if the number of distinct edges traversed by the i-th walk is a_i. We say that *the collection is compact*, if all its walks are compact, and for $i \neq i'$ the sets of leaves visited by the walks i and i' are disjoint.

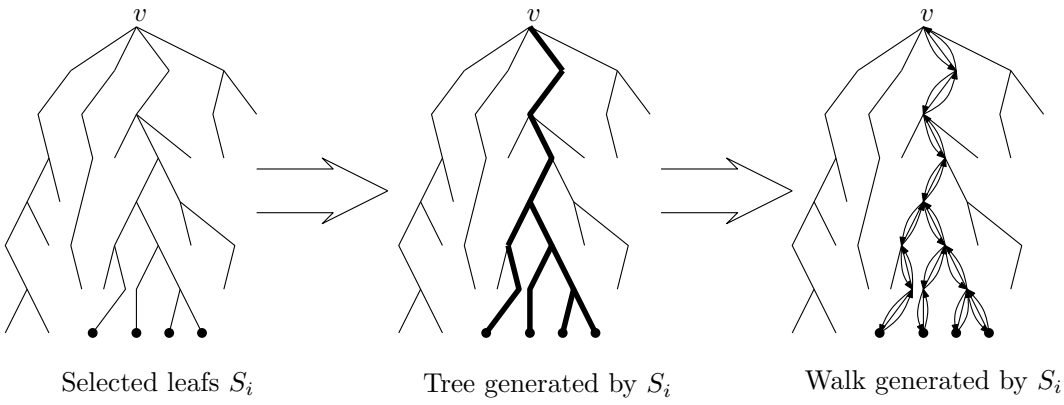

Figure 3.2: The method of obtaining feasible walks that visit selected leaves.

Given an arbitrary subset S of leaves of T, consider tree T'' obtained by taking all edges that are contained in the simple paths between the root of T and nodes in S. We say that T'' is *a tree generated by leaves* S (compare Figure 3.2). A sequence of edges obtained from a dfs-traversal of T'' (that starts and terminates in the root) defines a feasible walk W of a robot. We say that W is *a walk generated by set of leaves* S. Observe that a generated walk is also a compact walk.

There might exist many solutions with the optimal cost, and we show in the following lemma, that among those solutions there exists one with the properties we desire. In the remaining part of this section, we show how to find such an optimal solution.

Lemma 3.7. *There exists an optimal solution which is a compact collection of walks.*

Proof. First, observe that if two robots enter the same leaf, we can easily modify the walk of one of them in such a way that it does not enter this leaf. Certainly, it will not increase the overall cost of the solution.

Now, let T' be a subtree defined by a walk of some robot i. Assume that there is a leaf s in T' that is not a leaf in our tree T. Denote by v the father of s and by v' any son of s in T (there exists at least one son, as s is not a leaf in T).

The robot i enters s from v and immediately leaves s through v. This happens to be an unnecessary movement, as node s is visited also by another robot. Indeed, as walks belong to a feasible solution, there exists robot $i' \neq i$ that visits v'. As the graph is a tree, there is no other way to explore v' than by going through s. The walk of the i-th robot can be improved so that it does not enter s at all, by removing sequence (s, v) from it.

This does not increase the solution's cost, so the improved walks state also an optimal solution. In this way, we assure that there exists an optimal solution with compact walks. ■

From Lemma 3.7 we conclude that there exists an optimal solution in which each walk can be represented as a collection of leaves visited by a given robot. In such a way, our general problem can be reduced to the problem of clustering of the tree's leaves. Each cluster defines a subtree (a collection of simple paths between leaves and s) of a certain size. The goal is to cluster leaves in such a way that the size of the maximal tree is minimized.

Let S be the set of all leaves in T. Consider a partition of S into k disjoint sets (some of them might be empty). We obtain sequence $C = (S_1, S_2, \ldots, S_k)$ such that $\bigcup_i S_i = S$. The compact walks W_i, defined by S_i, jointly cover T, so the sequence C defines a feasible solution to the problem.

To find an optimal solution, we check all possible partitions of S into k disjoint sets. As $|S| \leq n$ (there are $n+1$ nodes in the rooted tree T), we have at most k^n such partitions – each defining some sequence C. To check the cost of the solution defined by C, we need to compute the length of each walk W_i. As each walk is of length not greater than $2n$, the dfs-traversal of all walks takes at most $2n \cdot k$ steps. Therefore, sequence C with the minimal cost can be found in time $O(k^{n+1} \cdot n)$.

Lemma 3.8. *The optimal solution can be computed in time $O(k^{n+1} \cdot n)$.*

3.3.2 Dynamic programming approach

A tree is a graph which incite to use a dynamic programming. The algorithm presented in this section uses information gathered about subtrees to obtain the solution for a larger trees containing those subtrees. For each subtree T_v of T, the algorithm maintains k-dimensional array A_v of size n, which stores up to n^k different walks of k robots over this subtree. Element $A_v[a_1, a_2, \ldots, a_k]$ is sequence $(S_1, S_2, \ldots, S_k)$ of sets $S_i \subseteq V$ $(1 \leq i \leq k)$, each being a subset of leaves in T_v, such that:

- leaves S_i for some i are visited by a robot,
- the subtree generated by S_i contains a_i edges.

Therefore, each non-empty set S_i defines a compact walk of length $2 \cdot a_i$ within T_v (the length of a dfs-sequence over a subtree consisting of a_i edges).

The recursive algorithm `ComputeTraversals`(v, T_v) computes array A_v for node v (which is not a leaf). The array for larger trees is constructed in a bottom-up fashion, starting from the smallest subtrees of a given tree. Consider node v with non-empty set of sons $s_1, s_2, \ldots, s_d$. For certain s_i, two cases have to be considered: either array A_{s_i} is already computed, or s_i is a leaf, and an appropriate initial array has to be computed. The algorithm constructs and extends these arrays, so that they describe walks in trees $T_{s_i} \bigcup \{(v, s_i)\}$, and then combines them to build array A_v describing walks in subtree T_v.

Algorithm 1 `ComputeTraversals`(v, T_v)

Require: v is not a leaf

$A_{\text{null}} \leftarrow$ empty array (all elements equal $(\emptyset, \emptyset, \ldots, \emptyset)$)
$A_v \leftarrow A_{\text{null}}$
for (each son s of v) **do**
  `/* - - - - - compute` A'_s `array - - - - - */`
  $A'_s \leftarrow A_{\text{null}}$
  if (s is a leaf) **then**
    $A'_s[1, 0, 0, \ldots, 0] \leftarrow (\{s\}, \emptyset, \emptyset, \ldots, \emptyset)$
    $A'_s[0, 1, 0, \ldots, 0] \leftarrow (\emptyset, \{s\}, \emptyset, \ldots, \emptyset)$
    ...
    $A'_s[0, 0, 0, \ldots, 1] \leftarrow (\emptyset, \emptyset, \emptyset, \ldots, \{s\})$
  else
    $A_s \leftarrow$ `ComputeTraversals`(s, T_s)
    for $\vec{b} \in [1, 2, \ldots, n]^k$ where $\vec{b} \neq (\emptyset, \emptyset, \ldots, \emptyset)$ **do**
      $\vec{c} \leftarrow (\vec{b}$ with all non-zero elements increased by one)
      $A'_s[\vec{a}] \leftarrow A_s[\vec{b}]$
    end for
  end if
  `/* - - - - - update` A_v `by merging with` A'_s `- - - - - */`
  if ($A_v == A_{\text{null}}$) **then**
    $A'_v \leftarrow A'_s$
  else
    $A'_v \leftarrow A_{\text{null}}$
    for (each $\vec{a}, \vec{b}$ s.t. $A_v[\vec{a}] \neq (\emptyset, \emptyset, \ldots, \emptyset)$ and $A'_s[\vec{b}] \neq (\emptyset, \emptyset, \ldots, \emptyset)$) **do**
      $\vec{c} \leftarrow \vec{a} + \vec{b}$
      $A'_v[\vec{c}] \leftarrow A_v[\vec{a}] \cup A'_s[\vec{b}]$ `/* element-wise sum */`
    end for
  end if
  $A_v \leftarrow A'_v$
end for
return A_v

In the first case, if for some s_i array A_{s_i} has been computed before, then for a certain vector $\vec{b}$ the element $A_{s_i}[\vec{b}]$ is the sequence describing the walks of k robots in T_{s_i}. There might be some robots which do not enter subtree T_{s_i} at all (zeros in $\vec{b}$), and thus they are omitted by constructing array T_v (where v is the father of s_i). On the other hand, all robots which explore T_{s_i} have to enter node s_i through node v, which introduces an additional edge (v, s_i) to the subtrees describing their movements. The walks are still described by the same collection of leaves $A_{s_i}[\vec{b}]$, but now they have to be stored at address $\vec{a}$ obtained from $\vec{b}$ by increasing all non-zero elements. In such a way, the new array A'_{s_i} is obtained.

In the second case, where s_i is a leaf, initial array A'_{s_i} is computed. As a part of the optimal solution, there is only a single robot which enters an edge (v, s_i), and thus there are k non-empty entires in an appropriate array:

$$A'_{s_i}[1, 0, 0, \ldots, 0] \leftarrow (\{s_i\}, \emptyset, \emptyset, \ldots, \emptyset)$$

$$A'_{s_i}[0, 1, 0, \ldots, 0] \leftarrow (\emptyset, \{s_i\}, \emptyset, \ldots, \emptyset)$$

$$\ldots$$

$$A'_{s_i}[0, 0, 0, \ldots, 1] \leftarrow (\emptyset, \emptyset, \emptyset, \ldots, \{s_i\})$$

For all other (non-unit) vectors $\vec{a}$, the element $A'_{s_i}[\vec{a}] = (\emptyset, \emptyset, \ldots, \emptyset)$.

Given arrays A'_{s_i} for all sons of v, we can merge them to obtain array A_v describing walks of robots in subtree T_v. The integration is done sequentially for each son s_i. Assume that $i-1$ sons are already integrated, and that a robot, by going from v through sons $s_1, s_2, \ldots, s_{i-1}$, reaches set S of leaves. If the robot reaches set S_i of leaves, going through the son s_i, then after integration it visits leaves $S \bigcup S_i$. The size of the integrated tree is a sum of sizes of subtrees being integrated (as these subtrees are edge-disjoint). After the last array A'_{s_i} is integrated, array A_v contains the walks of the robots on tree T_v.

Lemma 3.9. *The* `ComputeTraversals`(v, T_v) *algorithm terminates in time* $O(n^{2k+1})$.

Proof. Each edge (v, u) in T takes part only once in the integration of arrays, where each pair of vectors is computed in time $O(n^{2k})$, and only once in the process of extending walks (time $O(n^k)$) over subtree T_u that is attached to this edge. Therefore, the algorithm `ComputeTraversals` terminates in time $O(n^{2k+1})$, as there are n edges in the tree. ■

In the next lemma, we prove that the array returned by the algorithm has the desired property, which let us later conclude that the optimal solution is also stored in this array.

Lemma 3.10 (Feasibility of the array)**.** *Assume that A_v is the array returned by the algorithm* `ComputeTraversals`(v, T_v). *If there exists collection C of k compact walks of size $\vec{a}$ covering tree T_v, then*

$$(S_1, S_2, \ldots, S_k) := A_v[\vec{a}]$$

generates collection C' of k compact walks of size $\vec{a}$ that also covers T_v.

Proof. We present an inductive proof of the desired properties of the array returned by the `ComputeTraversals`(v, T_v) algorithm. Assume that tree T_v is 1 in height. That is, it consists of root v and some number d of sons. Consider a list of all non-empty elements of the initial arrays created by the algorithm for each son:

A'_{s_1}: $(\{s_1\}, \emptyset, \emptyset, \ldots, \emptyset), (\emptyset, \{s_1\}, \emptyset, \ldots, \emptyset), (\emptyset, \emptyset, \{s_1\}, \ldots, \emptyset), \ldots, (\emptyset, \emptyset, \emptyset, \ldots, \{s_1\})$,

A'_{s_2}: $(\{s_2\}, \emptyset, \emptyset, \ldots, \emptyset), (\emptyset, \{s_2\}, \emptyset, \ldots, \emptyset), (\emptyset, \emptyset, \{s_2\}, \ldots, \emptyset), \ldots, (\emptyset, \emptyset, \emptyset, \ldots, \{s_2\})$,

$\ldots$

A'_{s_d}: $(\{s_d\}, \emptyset, \emptyset, \ldots, \emptyset), (\emptyset, \{s_d\}, \emptyset, \ldots, \emptyset), (\emptyset, \emptyset, \{s_d\}, \ldots, \emptyset), \ldots, (\emptyset, \emptyset, \emptyset, \ldots, \{s_d\})$.

The algorithm computes element-wise sums of every combination of those sets by taking one set per row (in a pseudo-code of the algorithm, it is done sequentially by combining row by row). Therefore, the resulting array A_v is non-empty for each vector $\vec{a}$, such that $\sum_i a_i = d$. Moreover, $A_v[\vec{a}]$ contains the collection of sets of leaves that generates walks of size $\vec{a}$, and as all leaves are visited by a robot (each row is taken into the combination), tree T_v is jointly covered by those walks.

Assume that the algorithm returns a feasible array for every tree $h \geq 1$ in height. We show that for tree T_v which is $h + 1$ in height, the constructed array A_v is also feasible. Denote by d' the number of sons of v which are not leaves and by d the number of sons which are leaves in T_v.

Take s as a non-leaf-son of v. Collection C describes walks that also enter subtree $T_s \subseteq T_v$ and cover it completely. Therefore, for each s there exists collection C_s of compact walks covering $T_s \cup (s, v)$, and we denote its size by $\vec{b}^{(s)}$. As the trees T_s are all of height of at most h, the computed arrays A_s are all feasible, and by the inductive assumption, $A_s[\vec{b}^{(s)}]$ generates the collection of compact walks of size $\vec{b}^{(s)}$ jointly covering T_s.

For $(S_1, S_2, \ldots, S_k) := A_s[\vec{b}^{(s)}]$, the sequence $(S_1, S_2, \ldots, S_k)$ defines the compact walks also in tree $T_s \cup (s, v)$, but its size is now described by vector $\vec{c}^{(s)}$, obtained from vector $\vec{b}^{(s)}$ by increasing its all non-zero elements by one – the cost of an additional edge (s, v). The algorithm moves an element $(S_1, S_2, \ldots, S_k)$ from address $\vec{b}^{(s)}$ in array A_s to address $\vec{c}^{(s)}$ to obtain a new array A'_s defining compact walks in tree $T_s \cup (s, v)$. We have

$$\sum_s c_i^{(s)} = a_i \,,$$

as this is the sum of the edges visited by the i-th compact walk in tree T_v. For each son s being a leaf the initial array A'_s is computed as in the case of trees 1 in height.

Therefore, for each s being a son of v the algorithm computes array A'_s, such that $A'_s[\vec{c}^{(s)}]$ contains a definition of a collection of k compact walks of size $\vec{c}^{(s)}$ covering $T_s \cup (s, v)$, and furthermore $\sum_s c_i^{(s)} = a_i$. As in the integration phase, all combinations of elements of arrays corresponding to sons are considered, there are also elements corresponding to all vectors $\vec{c}^{(s)}$ summed element-wise. And thus, in resulting array A_v, address $\vec{a}$ (where $\sum_s c_i^{(s)} = a_i$) is occupied by the collection of k compact walks covering tree T_v. ■

From Lemma 3.10 we can conclude the following lemma.

Lemma 3.11. *For array A_v, computed by* `ComputeTraversals`(v, T_v)*, there exists address $\vec{a}$, such that $A_v[\vec{a}]$ contains sequence $(S_1, S_2, \ldots, S_k)$ where S_i generates a walk of l_i in size, and $\max_i\{l_i\}$ is the optimal cost of the collective tree exploration problem.*

The algorithm `ComputeOpt`(s, T) (see Algorithm 2) uses the `ComputeTraversals`(v, T_v) to obtain an appropriate array, and then extracts the optimal solution from the array.

Algorithm 2 `ComputeOpt`(s, T)

```
1: if (s is a leaf) then
2:     return ∅
3: end if
4: A ← ComputeTraversals(s, T)
5: find a⃗ minimizing max_i{a_i} for which A[a⃗] ≠ (∅, ∅, ..., ∅)
6: (S_1, ..., S_k) ← A(a_1, ..., a_k)
7: return collection of walks described by S_1, ..., S_k
```

Lemma 3.12. *The algorithm* `ComputeOpt`(s, T) *computes an optimal solution to the problem in time $O(n^{2k+1})$.*

Proof. The first step of `ComputeOpt` is a call to subalgorithm `ComputeTraversals`, which computes in time $O(n^{2k+1})$ array A containing all reasonable traversals of T. As A contains $O(n^k)$ collections and therefore the optimal traversal of T can be easily found in $O(n^k)$ time. This implies that `ComputeOpt` finds the optimal solution in $O(n^{2k+1})$ steps, which is polynomial in the size of the tree. ■

From Lemma 3.12 and Lemma 3.8 we can conclude that there exists an algorithm which computes the optimal solution in $O(\min\{k^{n+1} \cdot n, n^{2k+1}\})$. It is sufficient to check if $k^{n+1} \cdot n < n^{2k+1}$. If this is the case, we run the first algorithm. Otherwise, we use `ComputeOpt`(s, T).

Chapter 4

Energy-aware graph exploration

Autonomous robots act independently, and there is no central authority controlling them. In this situation, it is more likely that they are not connected by any rope or wire, which would deliver energy or e.g. fuel. Each robot carries its own source of energy, like a fuel tank or a battery, which unfortunately is always very limited in its capacities. The limitations might be dictated by the weight of the battery or the tank, which basically means additional costs of storing such enormous amount of energy.

We rather tend to construct the strategies, so that the maximal energy used by a robot is as small as possible. In our context, we assume that the energy is basically used only for a movement of a robot. In some cases, the energy spent on communication is an important fraction of an overall usage, however, here we neglect the energy used to send data packets (using wireless devices) or to carry out the local computations. Certainly, it would be an interesting extension of our model to minimize not only the energy for movements but also the communication cost.

The number of edge traversals done by the i-th robot can be interpreted as the total energy used by this robot until the end of the exploration. We recall the energy cost of online algorithm ALG

$$\mathcal{E}_{ALG}(G,s) := \max_{1\le i\le k} \left|\{0 < t \le |Walk_i| : \ p_i(t-1) \neq p_i(t)\}\right| ,$$

which is the maximal energy used by a robot in the team.

The optimal offline cost $\widetilde{C}_{OPT}(G,s)$ (the graph is known in advance) is not greater than the energy cost $\mathcal{E}_{ALG}(G,s)$ of the online algorithm. Clearly, knowing the topology robots can plan their strategy more carefully, and therefore gain some better energy-savings. We aim at minimizing ratio $\mathcal{E}_{ALG}(G,s)/\widetilde{C}_{OPT}(G,s)$, so that the competitive ratio is as small as possible. The smaller the competitive ratio, the better the algorithm copes with an unknown topology and a lack of a map.

In Section 4.1, we show a lower bound on the competitiveness of an arbitrary online algorithm. There exists a simple tree, such that if its topology is not known to the algorithm in advance, the algorithm will fail to explore it by the optimal offline cost $\widetilde{C}_{OPT}(G, s)$. Furthermore, in Section 4.3, we show two algorithms efficiently exploring an unknown tree.

4.1 The lower bound of 1.5 for the competitive ratio

Consider arbitrary, randomized, online algorithm *ALG* exploring graphs using k robots. In this section, we present a star which is constructed by *adaptive adversary*, depending on the behavior of a team during the exploration. The star is easy to explore offline with a small cost, but it cannot be optimally explored, if its topology is not known to *ALG* in advance. We show bounds on those costs and derive the general bound on the competitive ratio.

For an arbitrary large $p \in \mathbb{N}$ consider a star, shown in Figure 4.1, where the center is base s. The star is a union of $2k - 2$ paths of length p and one path of length $2p$. The i-th short path consists of p nodes $v_{1,i}$ trough $v_{p,i}$. Parameter q defines which path is of an extended length consisting of $2p$ nodes $v_{1,q}$ trough $v_{2p,q}$. Knowing how the algorithm *ALG* explores a star, the adversary declares q in such a way that it additionally increases the algorithm's cost.

Star $T_{p,q}$ consists of $2p \cdot k$ edges, and no graph with such number of edges can be explored by k robots with a cost smaller than $(2p \cdot k) \cdot 2/k = 4p$. However, the topology is known beforehand, and if each but the first robot explores two short paths, and the first one explores only the long path, it takes exactly $4p$ energy units per robot to explore $T_{p,q}$.

Lemma 4.1. *For an arbitrary $p \in \mathbb{N}$ and $1 \leq q \leq 2k - 1$ an optimal offline algorithm explores $T_{p,q}$ at cost of*

$$\widetilde{C}_{OPT}(G, s) = 4p \, .$$

Algorithm *ALG* does not know the topology, and thus does not know which path is extended (value q).

Assume for a while that all paths are short, and denote by t_0 the first step at which there exists a robot, say the i-th robot, positioned in $v_{p,q'}$, which has visited some $v_{p,q''}$ ($q'' \neq q''$) before. The adversary extends exactly the path, the i-th robot is positioned in (we lay $q = q'$). At this time, the i-th robot uses at least $3p$ energy units, as it has also visited $v_{p,q''}$. If this robot explores an extended path (visits $v_{p,q''}$), it will use at least $3p$ additional energy units to reach $v_{2p,q}$ and return to the base.

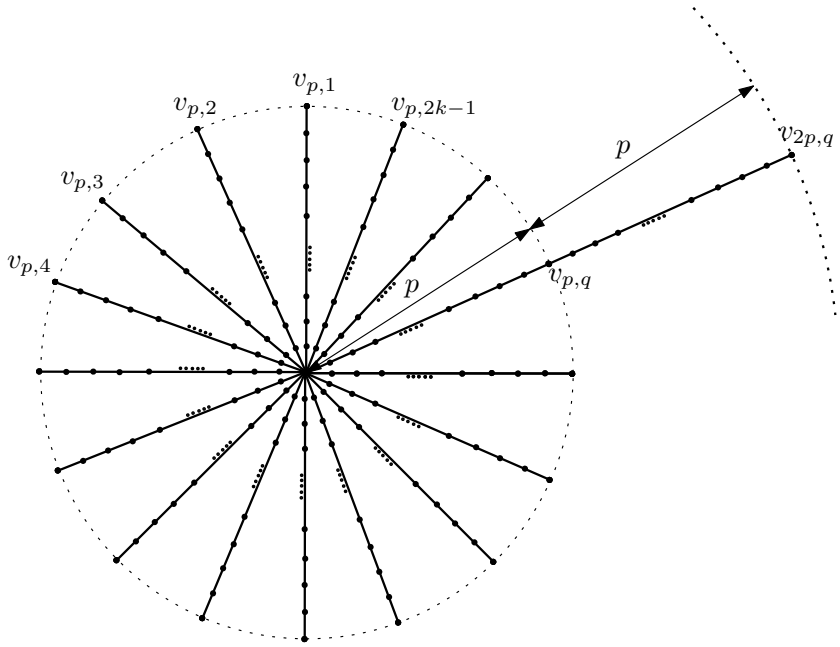

Figure 4.1: The star with an extended path.

We show that even if another robot, say robot i' (where $i' \neq i$), explores the extended path, it will also result in the energy-cost of at least $6p$. Assume that i' has not visited dead-end of any short path before (otherwise it would use $6p$ energy units). Starting in the base, it visits an extended path and returns to the base, which takes at least $4p$ energy units. Again, it will not visit any other dead-end, as it is not possible to do with the cost smaller than $6p$. If the algorithm tries to maintain the cost not greater than $6p$, it needs some robot (different than i-th robot) to explore **only** the long path and no other paths at all[1].

Let's have a closer look at the situation at time t_0. There are $m-1$ short paths explored by $m-1$ robots and one short path explored by the i-th robot (by definition of t_0). Each robot uses at least $2p$ energy units until t_0. Beside the extended path, there are still $2k-2-m$ short paths to be explored, and neither i nor i' can help by their exploration. Even if $m-1$ robots explore a short path each, there are $2k-2-m-(m-1) = 2k-1-2m$ short paths waiting to be explored, but only $k-2-(m-1) = k-1-m$ robots which can help with it. Assuming that all $k-1-m$ robots are 'fresh' (they have not traversed any edge until now), they can explore at most $2k-2-2m$ short paths. Then, there still exists one short path to be explored at the moment, where all robots have used at least $4p$ energy units. No robot can do it with the cost smaller than $6p$.

[1] It might enter other paths, but cannot explore it completely, so that no other robot will have to enter it afterward.

Lemma 4.2. *Every online algorithm ALG using k robots to explore $T_{p,q}$, where $p \in \mathbb{N}$ and parameter q ($1 \leq q \leq 2k-1$) is set by the adversary, has energy cost*

$$\mathcal{E}_{ALG}(G, s) \geq 6p \, .$$

Using observations made in Lemma 4.1 and Lemma 4.2, we obtain the following main theorem of this section:

Theorem 4.3. *Every randomized, online graph exploration algorithm ALG for k robots has the competitive ratio of at least $3/2$, against the adaptive adversary.*

4.2 Tree exploration algorithms

Consider an exploration of a tree rooted at s, consisting of n uniform labeled edges, where the furtherest node is D edges away from s. All k robots $R_1, R_2, \ldots, R_k$ are initially placed in s, and each can traverse one edge per step to jointly visit all nodes of a tree. Robots can *locally* communicate, when they are in the same node.

In the energy model, whenever a robot traverses an edge, it uses one energy unit. We are interested in costs of the exploration, defined as the maximal energy used by a robot. Once again, we compare the cost of the online algorithm to the cost of the optimal offline algorithm to obtain an competitive ratio. In the energy model, robots do not care about an overall time of the exploration. Therefore, in some situations, halting and waiting for new information may be more desirable for a robot than a further exploration. This is exactly the case in our algorithms. During a round, there is only one active robot, and all other robots are positioned in s and do not move.

Consider a dfs-traversal of a tree as a sequence of nodes $\{w_1, w_2, \ldots w_{2n}\}$. Nodes might appear in the sequence many times, but their length always equals $2n$. Observe that we can easily extend the partial dfs-sequence $\{w_1, w_2, \ldots w_m\}$ (where $m < 2n$), assuming that w_j for $1 \leq j \leq m$ is known.

In our algorithms, the active robot first goes to the last node of the known (partial) dfs-sequence, say node v_{j-1}, and then continues extending this sequence for a certain number of steps until node v_j. It eventually returns to s, where another active robot is selected for exploration. Additionally, the current dfs-subsequence is broadcasted within the team, and as all robots are positioned in the root at this moment, only a strictly *local communication* is used. In the next round, the active robot goes to v_j, and the whole round is executed again in the similar way.

We present two algorithms which follow this pattern. The simple algorithm in Section 4.2.1 is 8-competitive and bases on observations made about the offline approximation. Later, in Section 4.2.2, we improve this result to $4 - 2/k$-competitive algorithm.

4.2.1 Simple 8-competitive tree exploration algorithm

In Section 3.2, we have seen that if we split the dfs-sequence of tree T in sections of $M := 2 \cdot \max\{n/k, D\}$ edges in size, then it leads to 2-approximation of the offline exploration problem. Here we use this observation to obtain a fully online exploration algorithm minimizing the maximal energy used by a robot.

Notice that if M is known to the algorithm, it does not require any additional knowledge to work properly, as the dfs-sequence can be constructed sequentially by extending it by an additional edge. Denote the dfs-sequence of T by $v_i \in V$ for $0 \leq i \leq 2n$, such that $v_0 = s = v_{2n}$, $(v_i, v_{i+1}) \in E$ and $\bigcup v_i = V$. Giving any initial sequence $v_0, \ldots, v_j$, node v_{j+1} can easily be determined. Moreover, the simple path from any v_i to the root is known and can be easily used to travel forth and back between base station s and node v_i.

In the algorithm presented in Section 3.2 the first robot R_1 extends the (empty) dfs-sequence by M steps and then R_2 goes to the place where R_1 stopped and extends the sequence for another M steps. This procedure is continued until the whole dfs-sequence is traversed, and thus T explored. However, in our online setting, we cannot directly define those sections, since neither n nor D, needed to compute M, is known to the algorithm beforehand. We use a simple technique to approximate the real value M. Algorithm 3 explores an unknown tree T, binary searching for M.

Algorithm 3

$v_0 \leftarrow s$
$e_{\max} \leftarrow 2$
while (T is not yet explored) **do**
  $e_{\max} \leftarrow 2 \cdot e_{\max}$
  $v_i \leftarrow \emptyset$ for all $1 \leq i \leq k$
  for $i = 1$ to k **do**
    $R \leftarrow R_i$ *//the active robot*
    R travels to v_{r-1}
    R executes $e_{\max}/2$ steps of DFS but breaks if it leads deeper than $e_{\max}/4$ from s
    $v_r \leftarrow$ current node the robot R is in
    R returns to s
  end for
end while

An iteration of the *for-loop* is *a round*. In each round, there is selected robot R, which is the only one which is active (moves) in this round. As the loop runs from 1 through k, each robot is selected exactly once per so called *epoch* (an iteration of the *while-loop*). Therefore, there are exactly k rounds within an epoch – a round per one robot from the team.

When a robot gets selected, it receives the energy supply of e_{max}, which is used to explore as much dfs-edges as possible. Parameter e_{max} is defined once at the beginning of each epoch, in the following way. Initially, e_{max} equals 2, but its value is doubled at the beginning of the epoch, such that the energy available for a robot increases.

In a round, the active robot R uses its available energy in the following way. First it travels to node v_{r-1} where the previous active robot stopped its exploration. This node never lies deeper than $e_{\max}/4$ from s (see the 9-th line of the algorithm), and therefore it takes at most $e_{\max}/4$ energy to reach it. Then, robot R extends the known dfs-sequence for $e_{\max}/2$ steps, and that changes its position from node v_{r-1} to node v_r. At the end, it takes at most $e_{\max}/4$ steps to return to s, using a simple path in the tree.

In an epoch, each robot extends the dfs-sequence by $e_{\max}/2$ steps, using at most $e_{\max}$ energy units. In initial epochs, the total energy $k \cdot e_{\max}/2$ used to extend the dfs sequence, might be too small to cover the complete dfs-sequence which is $2n$ in length. However, we know that $e_{\max} = 2M$ energy units per robot is sufficient for a team to jointly traverse a complete dfs-sequence

$$k \cdot e_{\max}/2 = k \cdot 2 \max\{n/k, D\} \geq 2n \ .$$

Value $e_{\max}$ is doubled each epoch, and thus eventually, there will be value $2M \leq e_{\max} < 4M$ found. This will be the last epoch, after which the tree will be completely explored, and the algorithm will terminate. Therefore, the total energy used by a robot in all epochs is at most $8M$.

Lemma 4.4. *Algorithm 3 explores an unknown tree and uses at most*

$$8 \cdot \max\{2D, 2n/k\}$$

energy units per robot.

As we have observed in Section 3.1, the cost of the optimal offline algorithm is related to radius D and the number of nodes in G divided by the team's size k

$$\widetilde{C}_{OPT}(G, s) \geq \max\{2D, \lceil 2n/k \rceil\} \ .$$

Therefore, the Algorithm 3 explores an arbitrary tree and achieves the competitive ratio of 8. There is only a very restricted communication pattern required, as robots exchange messages only in the root of the tree.

Lemma 4.5. *The online Algorithm 3 completely explores an arbitrary, unknown tree T rooted at s, and achieves a competitive ratio* 8.

4.2.2 Improved $(4-2/k)$-competitive algorithm for trees

Here, we improve the Algorithm 3, presented above, but we follow the same scheme as before. There is only one robot active at a time, and all remaining robots are waiting in s. Each active robot also extends the known dfs-sequence, but now with the larger number of "extending" steps. As we know, the lower bound on the cost of the optimal solution grows with D. Therefore, when the active robot detects a node at a large distance, it has a proof that the optimal solution also has a large cost. Thus, the active robot can make larger than previously, in Algorithm 3, number of extending moves. For detailed description of the improved algorithm see Algorithm 4.

Algorithm 4

```
1: v_0 ← s
2: h ← 1
3: r ← 0        // round counter
4: while (T is not yet explored) do
5:    R ← robot with the least energy used        // the active robot
6:    R travels to v_{r-1}
7:    e ← 0
8:    repeat
9:       R follows a dfs-step
10:      e ← e + 1
11:      h ← height of the visited subtree
12:   until (e < 2h)
13:   h_r ← h
14:   v_r ← a current node robot R is in
15:   R returns to v_root        // this takes at most h_r steps
16:   r ← r + 1
17: end while
```

We call *a round* one iteration of the *while-loop*. At the beginning of the round, the least exhausted robot is selected as an active robot. First, the active robot travels to node v_{r-1} in which the robot active in the previous round has finished its exploration. Then, it continues extending the dfs-sequence until the height of the subtree known to the algorithm is greater than 1/2 of the "extending" steps done (see *repeat-loop* in Algorithm 4). This guarantees that the return path is not too large in length, comparing to the work done on extending the dfs-sequence. After this, the current position of the robot is stored in v_r, and h_r denotes the height of the visited subtree.

Therefore, the active robot uses at most h_{r-1} energy units to reach node v_{r-1}, then it continues traversing consecutive dfs-steps until it collects $2h_r$ edges, and finally returns

to s, which certainly takes at most h_r energy units. This gives us an upper bound of $h_{r-1} + 3h_r$ on the energy used by a robot during round r. In the first round ($r = 1$), the active robot is the first working robot ever, so it uses only $0 + 3h_r$ energy units (we lay $h_0 = 0$ and also $v_0 = s$). The last round q is perhaps shorter and takes $h_{q-1} + w$ steps, because the robot gets to the root already during the *repeat-loop* loop, and thus does not have to pay any extra costs for the return to the base.

Let e_i be the energy of the i-th robot R_i after completely exploring the tree, and $E_{\text{total}} = \sum_{i=1}^{k} e_i$ be the total energy used by all robots. We have

$$E_{\text{total}} \leq 3h_1 + \sum_{r=2}^{q-1}(h_{r-1} + 3h_r) + h_{q-1} + w = 4 \cdot \sum_{r=1}^{q-1} h_r + w$$

as the upper bound for this energy. On the other hand, we know that the robot active in round $r \leq q-1$ has done exactly $2h_r$ steps of the DFS tour (the last one – exactly w steps), which for the whole tree takes exactly $2n$ energy units. It must be that $(\sum_{r=1}^{q-1} 2h_r) + w = 2n$ and thus we have

$$E_{\text{total}} \leq 4n \leq 2k \cdot \widetilde{C}_{OPT}(G, s) ,$$

where $\widetilde{C}_{OPT}(G, s)$ is the energy cost of the optimal offline exploration.

Let $e_{\min} = min\{e_i : 1 \leq i \leq k\}$ and $e_{\max} = \max\{e_i : 1 \leq i \leq k\}$ be respectively the minimal and the maximal energy used by robots $R_{\min}$ and $R_{\max}$. Sequence h_r is non-decreasing, and so is the energy cost $h_{r-1}+3h_r$ of the robot active in round r. After each turn (probably but the last one) in which $R_{\max}$ is active, there is one turn in which $R_{\min}$ is active, so

$$e_{\max} - e_{\min} \leq h_{q-1} + 3h_q \leq 4D \leq 2 \cdot \widetilde{C}_{OPT}(G, s) .$$

To continue our proof, we need the following observation on an arbitrary sequence of natural numbers.

Lemma 4.6. *For an arbitrary sequence $a_i > 0$ of length k we have*

$$a_{\max} \leq \frac{1}{k} \cdot \sum_{i=1}^{k} a_i + \left(1 - \frac{1}{k}\right) \cdot (a_{\max} - a_{\min}) ,$$

where $a_{\max}$ and $a_{\min}$ denote the minimal and the maximal value respectively.

Proof. In the sequence, there is at least one element with value $a_{\max}$, and certainly all other $k-1$ elements are at least $a_{\min}$. Therefore, the total sum of all elements is bounded by

$$\sum_{i=1}^{k} a_i \geq a_{\max} + (k-1) \cdot a_{\min} .$$

We easily obtain

$$\frac{1}{k} \cdot \sum_{i=1}^{k} a_i \geq a_{\max} - \left(1 - \frac{1}{k}\right) \cdot a_{\max} + \left(1 - \frac{1}{k}\right) \cdot a_{\min} ,$$

which leads to the bound stated in the lemma. ■

Applying Lemma 4.6 to sequence e_i, we get the bound on the number of edges traversed by the most exhausted robot

$$e_{\max} \leq \frac{2k \cdot \widetilde{C}_{OPT}(G,s)}{k} + (1 - 1/k) \cdot 2 \cdot \widetilde{C}_{OPT}(G,s) ,$$

and therefore

$$\mathcal{E}_{ALG}(G,s) = e_{\max} \leq (4 - 2/k) \cdot \widetilde{C}_{OPT}(G,s) .$$

This leads to the upper bound of $4-2/k$ on the competitive ratio of our online exploration algorithm.

Lemma 4.7. *Algorithm 4 explores an arbitrary, unknown tree T from root s and achieves the competitive ratio of at most $4 - 2/k$.*

The analysis can be slightly improved (at the cost of readability), but it can also be proved that the competitive ratio of this algorithm asymptotically converges to 4. Note that this still does not close the "gap", as the general lower bound for the competitive ratio of an algorithm in this model is 3/2.

4.3 Exploration of spanning trees

Let T be the shortest path tree of graph $G = (V, E)$, such that T is rooted at s. Certainly, if some algorithm jointly explores T, it also explores graph G. Furthermore, we have observed in Section 3.2.1 that the offline cost of the exploration of G is $O\{D + n/k\}$, which is related only to the radius and the number of nodes in G. Therefore, as those two parameters are equal in T and in G, the exploration of T has the same (up to the constant factor) complexity as the exploration of G.

Observe that the shortest-paths property is much more than we actually need here. It is sufficient that T is γ-relaxed SPT T_γ rooted at s, whose height is bounded by $\gamma \cdot D$ for some γ (see Definition 3.3). By Theorem 3.4 the offline exploration cost of such a tree is close to the offline cost of the optimal algorithm exploring G

$$2\gamma \cdot \widetilde{C}_{OPT}(G,s) \geq \widetilde{C}_{OPT}(T_\gamma, s) \geq \widetilde{C}_{OPT}(G,s) .$$

The idea is that the online algorithm for trees could explore also a wider class of graphs, if each robot perceived the graph as a tree. It simply receives some selected edges instead of all edges in the neighborhood of its current position. Those selected edges build a tree which is explored by the algorithm. It is required, that the tree can be constructed in an online fashion. Recall that in the online setting graph G is not known to the algorithm in advance, and therefore there is no possibility to construct the relaxed shortest path tree in advance. The tree has to be constructed at certain places (dictated by the algorithm), edge by edge, when a moving robot asks about the neighborhood of the node it is already in. If the algorithm has the ability to chose only those nodes belonging to T_γ, the graph exploration can be reduced to the problem of the exploration of trees.

Let ALG be the online graph exploration algorithm which can explore trees, and whose all robots are positioned in node s of an unknown graph G. Let $E(t)$ be the set of edges that the algorithm ALG knows at time t. Set $E_{\text{new}}(t) = E(t) \setminus E(t-1)$ is a collection of edges that the algorithm discovers for the first time at step t.

Definition 4.8 (Online tree selection scheme)**.** The online tree selection scheme *selects at step t subset $E'(t) \subseteq E_{\text{new}}(t)$ of edges, such that the collection of selected edges $\bigcup_{t' \leq t} E'(t')$ form a tree that spans nodes contained in $E(t)$.*

The online tree selection scheme enables the algorithm to percept the graph as a tree. For algorithm ALG we are interested in such a scheme that constructs γ-relaxed SPT T_γ. Observe that having such a scheme for an efficient tree exploration algorithm, it is easy to obtain an efficient algorithm that explores arbitrary graphs.

Theorem 4.9. *For the online tree selection scheme constructing γ-relaxed SPT T_γ and α-competitive algorithm ALG exploring trees, there exists $(2\alpha \cdot \gamma)$-competitive algorithm exploring graphs for which the scheme is defined.*

Proof. The algorithm is α-competitive, and therefore its cost $\mathcal{E}_{ALG}(T_\gamma, s)$ is bounded by

$$\mathcal{E}_{ALG}(T_\gamma, s) \leq \alpha \cdot \widetilde{C}_{OPT}(T_\gamma, s) \, .$$

However, the Theorem 3.4 states that

$$\widetilde{C}_{OPT}(T_\gamma, s) \leq 2\gamma \cdot \widetilde{C}_{OPT}(G, s) \, ,$$

and therefore we obtain the bound of

$$2\alpha \cdot \gamma \cdot \widetilde{C}_{OPT}(G, s)$$

on the cost of algorithm ALG, which proves that it is $(2\alpha \cdot \gamma)$-competitive algorithm for graph G. ∎

In Section 4.3.1, we show an example of an online tree selection scheme and its usage to obtain an online algorithm for some interesting subclass of graphs.

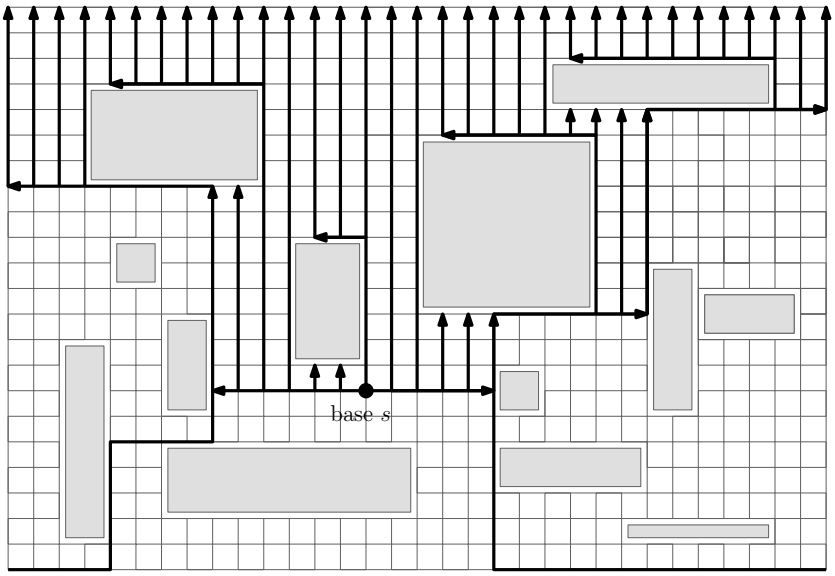

Figure 4.2: A tree constructed by the online tree selection scheme in a city-block graph.

4.3.1 Exploration of city-block graphs

We present the online tree selection scheme for interesting class of graphs (introduced by Betke, Rivest and Singh in [BRS95]), and the tree exploration algorithms described in Section 4.2. The main property of these algorithms is that there is always only one active robot which moves in the tree and, moreover, it moves along the dfs-sequence. This property enables us to deliver an easy tree selection scheme, as at each step there is only one node at which the outgoing edges have to be selected as a tree-edges (see Definition 4.8). Therefore, the selection scheme is reduced to the problem of constructing a dfs-sequence of a tree, spanning graph G, such that it is of a bounded height. We require that the finally constructed spanning tree is γ-relaxed SPT for some small γ.

A city-block graph is a rectangular grid occupied by a rectangular, axis-parallel obstacles (see Figure 4.2). Two obstacles cannot touch each other, i.e. there is always a path going between them. The borders of the grid are treated as obstacles. In Figure 4.2, the direction to the top is called a northern direction and other respectively southern, eastern and western. Given base station s, we define *monotone paths* (see a four-way decomposition in [BRS95]) that divide the graph into 4 regions (north, south, east and west). The east-north monotone path proceeds east, until the obstacle is hit. The direction is switched to north, and again the path is extended, until the obstacle is hit. The procedure is repeated, until the corner of the graph is reached[2]. All other paths (east-south, west-north, and

[2] Corners are well defined in the city-block graph.

west-south) are constructed analogically.

We show the scheme which bases on the *ray algorithm* introduced in [BRS95]. We present only the construction for the northern region, as it works analogically for all remaining regions. The ray always proceeds north (starting from the western part of the north-west monotone path), until it is blocked by an obstacle. When the ray approaches the eastern, far corner of the obstacle, it turns to the west and proceeds until the western, far corner of an obstacle. Then the procedure constructing northern rays is repeated, always with a one-to-the-east excursion, until the north-east monotone path is traversed.

Note that when the node is approached by a ray, it is always clear, which neighboring edged belong to the tree being constructed. This holds also on the border of regions defined by the monotone paths. Therefore, the scheme basing on this algorithm results in a spanning tree of the northern region of G.

Lemma 4.10. *The tree constructed by the above scheme for algorithm ALG is* $(3/2)$*-relaxed SPT of G with base s.*

Proof. Each path between base s and a leaf proceeds east only along the monotone paths. At all other nodes, it proceeds either north or west. As there are at most edges going east on the monotone path, $D/2$ levels that can be traversed going north, and $D/2$ levels going west, the total length of the path connecting s to any leaf is bounded by $3/2 \cdot D$. ■

If we use the $(4-2/k)$-competitive online algorithm for trees (see Section 4.2.2) together with the scheme described above, we obtain (by Theorem4.9) the algorithm with the competitive factor of $2(4 - 2/k) \cdot (3/2)$.

Lemma 4.11. *There exists* $(12 - 6/k)$*-competitive algorithm exploring city-block graphs.*

4.4 Exploration of general graphs

We show a technique to explore arbitrary graphs by a team of k robots. Suppose for a moment that a single robot has to traverse all edges of a graph. Clearly, the efficiency of such an exploration depends on the graph's complexity. It is not hard to imagine a graph where some edges have to be traversed more than once. If the graph is Eulerian, there exists the Eulerian path that visits every edge exactly once.

If the graph is known beforehand, the optimal walk of a robot can be computed (Eulerian path in the case of Eulerian graph), and then the exploration is optimal. The problem gets more difficult when the algorithm does not know the graph beforehand. Deng and Papadimitriou in [DP99], Albers and Henzinger in [AH00], and Fleischer and Trippen in [FT05] observe that for directed, strongly connected graphs the algorithm

efficiency grows with, so called, deficiency, i.e. the number of edges that have to be added in order to make to the graph Eulerian.

In [BRS95, ABRS99, AK98], and finally in [DKK06], the piecemeal exploration of graphs by one robot with an additional fuel-constrain is studied, and the results consecutively improved. The fuel-constrain says that the robot must return to base station s after at most $2D \cdot (1 + \beta)$ traversals for, say, refueling, where β is a non-negative constant. Value $2D \cdot (1 + \beta)$ can be interpreted as the capacity of a fuel tank. The robot has to explore efficiently before the next refueling, to compensate the high costs of frequent base returns.

We adapt the one-robot algorithm, presented by Duncan, Kobourov and Kumar in [DKK06], to work in our multi-robot energy model. The lemma below comes from [DKK06] and states the main properties of the algorithm presented there.

Lemma 4.12 (see [DKK06])**.** *Any unknown graph $G = (V, E)$ with base node s and unknown radius D can be explored by a fuel-constrained robot with cost*

$$\Theta(|E|/\beta) ,$$

for $0 < \beta < 1$. The robot will never take more fuel than $2D \cdot (1 + \beta)$ (without knowing D).

Consider the algorithm mentioned in Lemma 4.12. It uses one robot and returns to s each $2D \cdot (1 + \beta)$ steps (or even more frequently). Let's use now all k robots, instead of only one robot, in the following way. When a current robot returns to s, the execution of the algorithm is temporarily terminated, and the robot with the smallest number of traversed edges (a lazy robot) is selected to be an active robot. The active robot continues the terminated algorithm. Let ALG be the algorithm using k robots and working as described above.

We will use the same technique as in Section 4.2.2 to prove the bound on the competitive ratio of ALG. We combine the bound on the total energy E_{total} used by all robots and the difference in the energy usage of the most and the least exhausted robot. Denote $|E|$ by m, then by Lemma 4.12, the total energy used by all robots

$$E_{\text{total}} = O(m/\beta) .$$

On the other hand, the difference between the total number of edges $e_{\min}$ traversed by a lazy robot and the total number of edges $e_{\max}$ traversed by a hard-working robot is bounded by the length of one trip. i.e.:

$$e_{\max} - e_{\min} \leq 2(1 + \beta) \cdot D ,$$

as ALG always selects the least exhausted robot to continue the exploration after "refueling".

For every sequence of positive numbers, each belonging to the interval of a bounded size, and where the sum of all elements of the sequence is also bounded, the maximal element cannot be too large. We use Lemma 4.6:

$$e_{\max} \leq \frac{1}{k} \cdot \sum_{i=1}^{k} e_i + \left(1 - \frac{1}{k}\right) \cdot (e_{\max} - e_{\min}) \ ,$$

and by substituting the bounds obtained above we get

$$e_{\max} = \mathcal{E}_{ALG}(G, s) = O\left(1/\beta \cdot m/k + 2(1 + \beta) \cdot D\right) \ .$$

We have to find a relation to the number of nodes, as those (and not edges) are objects to be explored in the model of this work. For given graph G on n nodes and m edges, value α is the minimal number fulfilling $m \leq \alpha \cdot n$. By applying this to the above bound, we obtain the upper bound

$$\mathcal{E}_{ALG}(G, s) = O\left(\alpha/\beta \cdot n/k + 2(1 + \beta) \cdot D\right) \ ,$$

formulated using the parameter n instead of m.

We recall from Section 3.1 that $D \leq \widetilde{C}_{OPT}(G, s)/2$ and $n/k \leq \widetilde{C}_{OPT}(G, s)/2$, and therefore we finally get

$$\mathcal{E}_{ALG}(G, s) = O\left(\frac{\alpha}{\beta} + 1 + \beta\right) \cdot \widetilde{C}_{OPT}(G, s) \ .$$

As the algorithm requires that $\beta < 1$, the best bound we get is

$$\frac{\mathcal{E}_{ALG}(G, s)}{\widetilde{C}_{OPT}(G, s)} = O(\alpha) \ ,$$

which is a bound on the competitive ratio of the online algorithm *ALG* in the energy model.

Lemma 4.13. *For an arbitrary undirected graph $G = (E, V)$ with $m = |E|$ edges and $n = |V|$ nodes, where $m = \alpha \cdot n$, our algorithm using k robots explores G from base $s \in V$ and achieves competitive factor of α.*

CHAPTER 5

Time-efficient graph exploration

In Chapter 4, we have studied a problem of exploration, aiming at minimizing the maximal energy used by a robot. An energy-awareness is an important aspect, since the abilities of each autonomous robot are restricted, as its accessible energy (stored in batteries) is bounded. Robots move carefully, since each motion uses costly energy, and as we have seen in Section 4.3, it is quite reasonable to move only one robot at a time and leave all remaining robots waiting in the base. This may result in energy-savings, but unfortunately leads to an extended exploration time.

There are scenarios, where we do care about the total exploration time. As an example, consider a rescue mission, where wounded people are waiting for help. In such cases, one does not care about the energy-savings. However, we will observe that in our model, minimizing the total exploration time entails minimizing the maximal energy as well. Minimizing time is more challenging, and we will later see, how it is reflected in the competitiveness of algorithms. The **time-cost** of algorithm ALG is formally defined as

$$C_{ALG}(G, s) := |Walk_1| \ ,$$

which is the total exploration time.

Assume that the graph is known to the algorithm beforehand, and denote by $\widetilde{C}_{OPT}(G, s)$ the optimal offline exploration cost. We measure the performance of an arbitrary online algorithm ALG by comparing its cost to $\widetilde{C}_{OPT}(G, s)$. In this model, $C_{ALG}(G, s)/\widetilde{C}_{OPT}(G, s)$ is competitive ratio of ALG, and we aim here at minimizing it.

Observe that, if ALG uses only one robot by the exploration, there are some graphs for which the competitive ratio is k. For instance, consider a star consisting of k edges outgoing from its center. Optimal algorithm needs 2 steps, and each algorithm using 1 robot needs at least k steps. Both algorithms presented in Chapter 4, use only one robot at a time, and all remaining robots are positioned in the base. Although it was sufficient in the energy-model to obtain a good performance, the worst-case competitive ratio of such algorithms in the time-model is $\Omega(k)$.

In this chapter, we show a surprising lower bound for the competitive ratio in the time-model. For each online algorithm, using even global communication, there exists a tree, whose exploration results in $\Omega(\log k/\log\log k)$-competitiveness (Section 5.1). We recall here, that there exists $O(1)$-competitive algorithm for the energy-model (see Chapter 4). Additionally, we show how to efficiently explore trees which can be embedded in multi-dimensional grids, and therefore are not too dense (see Section 5.2). Moreover, our methods use only very restricted, local communication. We start by presenting an overview of the results in this research area and describing how our results contribute to it.

5.1 The competitive ratio is $\Omega(\log k/\log\log k)$

Consider an arbitrary, randomized online algorithm *ALG* exploring graphs using k robots, each equipped with **global** communication devices. In this section, we show how the adaptive adversary constructs a tree which *ALG* explores inefficiently, comparing to the optimal offline cost – more precisely, we show lower bound of $\Omega(\log k/\log\log k)$ for the competitive ratio. Since the algorithm is arbitrarily chosen, this leads to the conclusion that in the worst case, no online algorithm can be efficient, no mater what the communication pattern is.

On the other hand, the tree presented below can be considered as a 'non-typical' graph[1] since its degree is large. If we model a terrain by a graph, it is quite an unusual case that the node's maximal degree is large. The same holds for networks, where much effort is put into avoiding such a situation where one node maintains many links.

We start our lower-bound construction by introducing an important component (*a tentacle*). Then, we show how to combine components of different sizes to obtain a lower bound tree, whose shape resembles some living creature in appearance, and thus will be called *a jellyfish tree*.

5.1.1 A basic component – a tentacle

A tentacle s in size, shown in Figure 5.1 consisting of a single path t in height and $s+1$ trees of degree t and 1 in height. The trees are numbered 1 through $s+1$ and the i-th tree is rooted in a leaf of the $(i-1)$-th tree (this leaf is called *a main node*). The first tree is rooted at the end of the single path. A whole tentacle consists of $t\cdot(s+1)$ edges. Main nodes v_i (where $1\le i\le s+1$) are defined as nodes which are visited very late by algorithm *ALG*. More precisely, for $d>t$ there is no node at distance d which is visited

[1] It is actually quite usual observation about lower bound constructions.

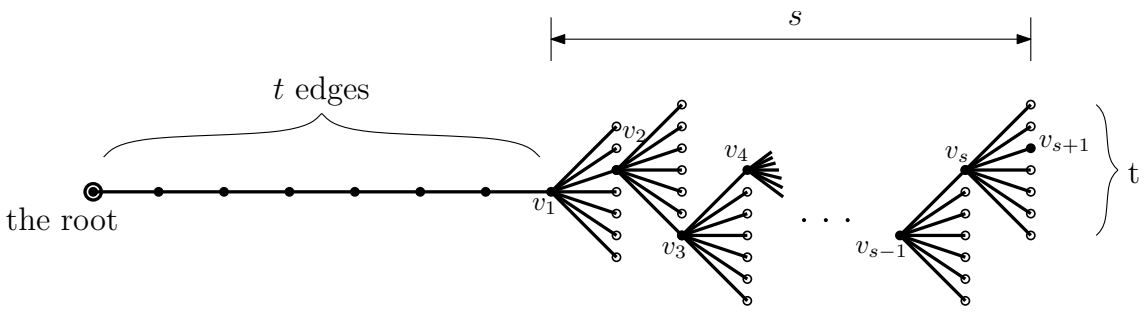

Figure 5.1: A tentacle s in size with the main nodes $v_1, \ldots, v_{s+1}$.

after main node v_{d-t+1} (also situated at distance d). Node v_{d-t+1} is the one, at distance d, which is visited as the last one.

The configuration of the main nodes makes the exploration harder. It takes many time steps to reach the far leaves of the tentacle, even if we consider many robots. We observe this property of the tentacle in the following lemma.

Lemma 5.1. *If v_l is the furthest visited main node, and the tentacle is explored for t time steps by k' robots, then the main node $v_{l+k'}$ is not visited.*

Proof. There are $k' \cdot t$ edges between nodes v_l and $v_{l+k'}$. At least $(k'-1)\cdot(t-1)$ of them do not lead to further edges, and thus are traversed twice. Since $t > k \geq k' \geq 4$, there is at least

$$2 \cdot (k'-1) \cdot (t-1) > k' \cdot t$$

traversals. This contradicts the fact, that a team of k' robots carries out at most $k' \cdot t$ traversals in t parallel time steps between time step $r \cdot t$ and $(r+1) \cdot t$. ■

Observe that no node at distance $d+1$ is visited before all nodes at distance d are visited. Therefore, the algorithm does not know the tentacle's size before completely exploring it. We will have many tentacles of different sizes, and this crucial property disables the algorithm to make a fair assignment of robots. The large tentacles should be explored by a large number of robots, but unfortunately, the algorithm does not know, which tentacles are large.

5.1.2 Jellyfish tree

To construct a lower bound tree, we use k tentacles in sizes of $s_1, s_2, \ldots, s_k$, defined as

$$s_i = \left\lceil \frac{k}{\log k} \cdot \frac{1}{i} \right\rceil .$$

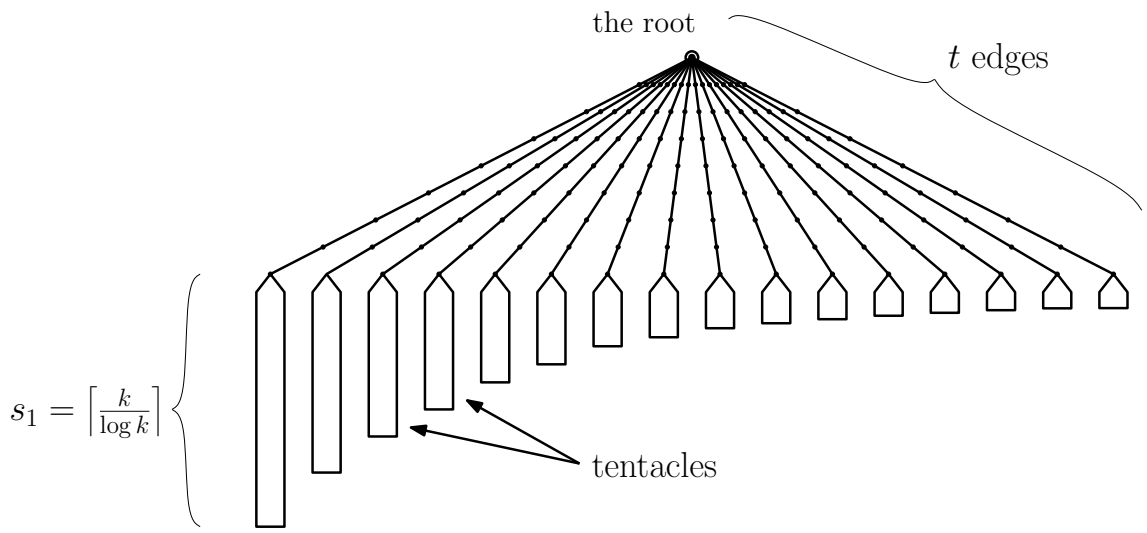

Figure 5.2: The jellyfish tree consisting of k tentacles.

All sizes are values between 1 and $\left\lceil \frac{k}{\log k} \right\rceil$, but note here that there might be more than one tentacle of a certain size. We glue together all roots of the tentacles, and obtain jellyfish tree J, shown in Figure 5.2.

As we have observed in Section 5.1.1, algorithm *ALG* does not know the size of the tentacle it explores before completely exploring it. This enables the adversary to rearrange the sizes of still unexplored tentacles, to make the exploration more difficult.

We split the algorithm's runtime into intervals $I_0, I_1, \ldots$, each t in length. At the beginning of each interval, the adversary declares the sizes of the tentacles, and then, the algorithm explores the tree for t steps. Some tentacles get completely explored, but as we will later see, many of them remain still unexplored.

Opening the initial interval I_0, the adversary declares the sizes in an arbitrary way. During the first t steps, robots may reach only the first main nodes in the tentacles, making an insignificant progress in the exploration. Now, consider interval $I_r = [rt + 1, \ldots, rt + t]$ and denote by $f_j^{(r)}$ the number of robots positioned in the j-th tentacle (but not in the root). The adversary orders the tentacle's sizes in the reversed order to the sequence $f_j^{(r)}$ (where $1 \leq j \leq k$). Therefore, the tentacles containing a large number of robots are small in size, whereas the tentacles with a small number of robots (or no robots) are large.

Each robot in the j-th tentacle can contribute only to the exploration of the main nodes within this tentacle; all other main nodes are just too far to be reached during the current interval. This implies that there are at most $f_j^{(r)}$ additional main nodes discovered during interval I_r (see Lemma 5.1). On the other hand, the adversarial assignment of tentacle's sizes makes the algorithm waste its exploration power – large groups of

robots are positioned in small-sized tentacles, where there is not much to explore within an interval. At the end of the interval there might be some tentacles which get fully explored, but the size of all residual (partially explored) tentacles is still unknown, which leaves the open space for the adversarial assignments done in the remaining intervals.

Lemma 5.2. *Let d be the distance from the root to the furtherest explored node at the end of I_r, where $0 \le r < \frac{1}{2} \cdot \frac{\log k}{\log\log k}$ and $\log k \ge 4$. For $y_r := t + \lceil (2\log k)^r \rceil$ we have $d \le y_r$.*

Proof. If at the end of I_0 there were a visited node at distance greater than $y_0 = t + 1$, it would mean that some robot traversed at least $t + 1$ edges in t time steps. Assume that at the end of I_{r-1} (at the beginning of I_r) all nodes up to the distance y_{r-1} are visited. The sequence $f_j^{(r)}$ describes a distribution of robots among tentacles, and the sizes of the tentacles are in reverse order to the order of $f_j^{(r)}$.

Assume that at the end of I_r there is a visited node v at distance $d > y_r$. This means that there were at least $y_r - y_{r-1}$ main nodes discovered in this interval. Node v lies within the j-th tentacle explored by many robots during I_r. Indeed, by Lemma 5.1 there were at least

$$y_r - y_{r-1} \ge (2\log k)^r - (2\log k)^{r-1} - 1 \ge (2\log k)^{r-1} \cdot (2\log k - 2)$$

robots exploring the j-th tentacle. All those robots at the beginning of I_r were situated in the j-th tentacle (and not in the root), and thus

$$f_j^{(r)} \ge y_r - y_{r-1} \ge (2\log k)^{r-1} \cdot (2\log k - 2)\,.$$

Now we prove that during the I_r, there were many tentacles containing at least $k' := f_j^{(r)}$ robots. Consider the following set

$$W_r = \{i :\ y_{r-1} < t + s_i \le y_r\}$$

describing all tentacles ending at the distance between y_{r-1} and y_r.

Observe that for $a = \lceil (k/\log k) \cdot 1/(2\log k)^r \rceil$ we have

$$t + s_a \le t + \left\lceil \frac{k}{\log k} \cdot \frac{1}{\frac{k}{\log k} \cdot \frac{1}{(2\log k)^r}} \right\rceil \le y_r\,,$$

and similarly, for $b = \lfloor (k/\log k) \cdot 1/(2\log k)^{r-1} \rfloor$ we have

$$t + s_b \ge t + \left\lceil \frac{k}{\log k} \cdot \frac{1}{\frac{k}{\log k} \cdot \frac{1}{(2\log k)^r}} \right\rceil \ge y_{r-1}\,.$$

Therefore, for all $a \le i \le b$ we have $i \in W_r$ (s_i is a decreasing function), and thus

$$|W_r| \ge b - a > \frac{k}{\log k \cdot (2\log k)^{r-1}} \cdot \left(1 - \frac{1}{2\log k} - \frac{(2\log k)^r}{k}\right).$$

Assuming $r < \frac{1}{2} \cdot \frac{\log k}{\log\log k}$, we obtain

$$|W_r| > \frac{k}{\log k \cdot (2\log k)^{r-1}} \cdot (1 - 1/\log k) \ .$$

All tentacles from set W_r are smaller than the j-th tentacle containing node v, and therefore, there are at least k' robots positioned in each such tentacle at the beginning of I_r (an adversarial order of the tentacle's sizes). Summing up the number of robots we obtain

$$|W_r| \cdot k' > k \cdot \left[\frac{(2\log k - 2) \cdot (1 - 1/\log k)}{\log k}\right] > k$$

assuming $\log k \geq 4$. This contradicts the fact that the algorithm uses k robots to explore the tree. ■

According to this lemma, in round $R = 1/4 \cdot \log k/\log\log k$ there is no visited node at distance greater than $y_R < t + \frac{k}{\log k}$. This means that at the end of I_R the largest tentacle is still not completely explored, and since each interval takes t time steps, we obtain the following lemma.

Lemma 5.3. *Assuming* $\log k \geq 4$*, the arbitrary algorithm ALG exploring graphs using k robots needs at least*

$$\left[\frac{1}{4} \cdot \frac{\log k}{\log\log k}\right] \cdot t$$

time steps to explore the jellyfish tree.

Assume now that the jellyfish tree is known to the algorithm before the exploration starts. This means all walks can be precomputed by an offline algorithm. Knowing the number of edges $n = O(t \cdot k)$ and height $h = O(t)$, we also know that an offline algorithm needs $O(t)$ time steps to explore the jellyfish tree. Nevertheless, here we explicitly show, how to explore a given jellyfish tree, using the fact that the sizes of tentacles are fixed and known.

Lemma 5.4. *An optimal algorithm needs* $O(t)$ *steps to explore the jellyfish graph.*

Proof. The exploration is carried out in two phases, where in each phase the algorithm takes some part of the largest remaining tentacles and explores them completely. In the first phase, there are only $\lceil k/\log k\rceil - 1$ largest tentacles explored, i.e. these in size of

$$s_1, s_2, \ldots s_{\lceil k/\log k\rceil - 1} \ ,$$

and the algorithm sends

$$r_i = \left\lceil \frac{k}{\log k} \cdot \frac{1}{2i} \right\rceil$$

robots to explore a tentacle which is s_i in size (where $1 \leq i \leq \lceil k/\log k \rceil - 1$). This is a feasible assignment (k robots suffices), since

$$\sum_{i=1}^{\lceil k/\log k \rceil - 1} r_i \leq \frac{k}{2\log k}\left(1 + \log\frac{k}{\log k}\right) + \frac{k}{\log k} \leq k \cdot \left(\frac{3}{2\log k} + \frac{1}{2}\right) \leq k\,,$$

having $\log k \geq 4$ and using $\sum_{i=1}^{m} 1/i \leq 1 + \log m$.

First, the robots go to the assigned tentacle, say the one s_i in size, until they reach the first main node v_1 (compare Figure 5.1). Then, the collective exploration starts. Each robot from the team takes exactly one unexplored edge and explores it in two steps, and then finally returns to the main node. Since there are t such edges, it takes exactly $2\lceil t/r_i \rceil + 1$ steps to completely explore one level of a tentacle and reach the next main node. Therefore, the complete exploration of this tentacle takes at most

$$t + s_i \cdot (2\lceil t/r_i \rceil + 1) - 1 + s_i + t \leq s_i \cdot \lceil t/r_i \rceil + 4t$$

steps, and since

$$2 \cdot s_i \cdot \lceil t/r_i \rceil \leq 2 \cdot \left\lceil \frac{k}{\log k} \cdot \frac{1}{i} \right\rceil \cdot \left\lceil \frac{t}{\frac{k}{\log k} \cdot \frac{1}{2i}} \right\rceil \leq 7t\,,$$

the first phase takes at most $11t$ time steps.

In the second phase, all remaining small-sized tentacles are explored. The algorithm assigns exactly one robot per tentacle larger than $s_{\lceil k/\log k \rceil - 1}$ in size, i.e these are $k - \lceil k/\log k \rceil + 1 \leq k$ tentacles of sizes $s_{\lceil k/\log k \rceil}, \ldots, s_k$. Observe that

$$s_{\lceil k/\log k \rceil} \leq \left\lceil \frac{k}{\log k} \cdot \frac{1}{\lceil k/\log k \rceil} \right\rceil \leq 1\,,$$

and therefore all remaining tentacles, explored in the second phase, are 1 in size. It takes at most $4t$ to explore each of them and finishes the exploration of the jellyfish tree in $O(t)$ time steps. ■

As an easy observation, combining Lemma 5.3 and Lemma 5.4, we obtain the bound on the competitive ratio.

Theorem 5.5. *Every randomized online collective graph exploration algorithm ALG has competitive ratio of at least*

$$\Omega\left(\frac{\log k}{\log\log k}\right)$$

against the adaptive adversary.

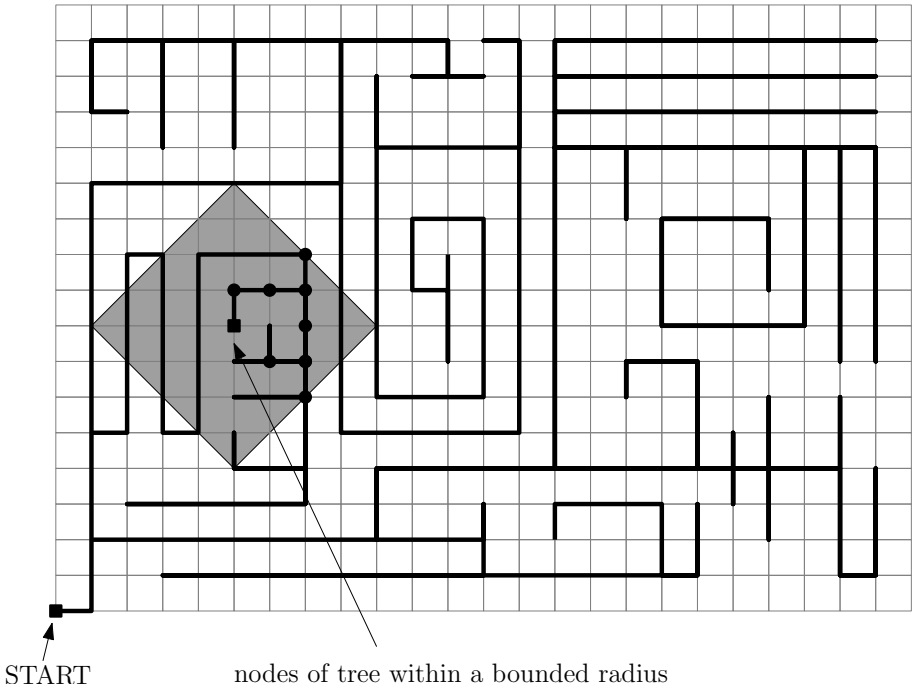

Figure 5.3: A tree embedded in 2-dimensional grid.

5.2 Local exploration of sparse trees

Consider a graph describing a geographical terrain, where nodes correspond to interesting locations, and edges show accessibility between two locations. It is quite likely that such a graph has a bounded degree, and what is more, there cannot be many nodes within a certain distance. This means that the graph is sparse and this property holds also for all its subgraphs.

In Figure 5.3, there is a grid which models all possible locations, and as there are some obstacles, it makes some locations inaccessible. The tree-like graph is embedded in the grid, and thus it is also sparse. The number of nodes within some bounded distance is also bounded.

In this section, we study the collective exploration of a restricted group of trees, and we present an algorithm for k robots using strictly local communication; in fact they communicate only when they meet in a node.

5.2.1 Density of a tree

Consider a rooted tree $T = (V, E)$ with $n(T)$ nodes and $h(T)$ in height. For $v \in V$ we denote by T_v a subtree of T rooted at v, and additionally, by $T_v(h)$ a tree obtained from

T_v by removing all edges (and corresponding nodes) at distance greater than h from v. Therefore, we have $h(T_v(h)) \leq h$.

Definition 5.6 (p-dense tree)**.** *We say, that $T = (V, E)$ is* p-dense*, if $p \in \mathbb{N}$ is the minimal positive number, such that for all h, $v \in V$ and $T' = T_v(h)$*

$$n(T') \leq [2h(T') + 1]^p \ .$$

Each tree embedded in a 2-dimensional grid (like in Figure 5.3) is either 1-dense or 2-dense. Indeed, here no tree can be 3-dense, since it would mean $n(T') > [2h(T') + 1]^2$ for some $T' = T_v(h)$, which is not possible in a grid where there are only $2h^2 + 2h + 1$ different nodes within any distance h from v. Since the 3-dimensional grid has less than $4 \cdot h^3$ nodes within a distance of h, then all embedded trees are either 1,2 or 3 dense.

5.2.2 Algorithm for sparse trees

Assume that we have p-dense tree T rooted at s. We show the `SparseExplore` algorithm using k robots, which efficiently explores this tree and achieves competitive ratio of

$$O\left(D^{1-1/p} \cdot \min\left\{p, \log p \cdot D^{1/2p}\right\}\right) \ ,$$

without knowing T or any parameter (like p or D) related to the tree.

Algorithm 5 `SparseExplore`(p_{init})

```
1: if (id = 1) then
2:     act as referee
3: end if
4: if (1 < id ≤ k/2) then
5:     BinarySE(p_init)
6: end if
7: if (id > k/2) then
8:     UnarySE(p_init)
9: end if
```

It consists of two algorithms (see Algorithm 5): `BinarySE` and `UnarySE`, whose description and analysis is the main part of this section. Each robot in a team plays one of 3 roles: it executes the first or the second algorithm, or is *a referee*. More precisely, there is only one referee, and the group of remaining $k-1$ robots are split roughly in half. The referee is positioned in s and stays there until the end of the exploration. Its only task is to inform a sub-team when the exploration is completed by another sub-team. In this way, the exploration stops roughly as soon as the faster algorithm finishes.

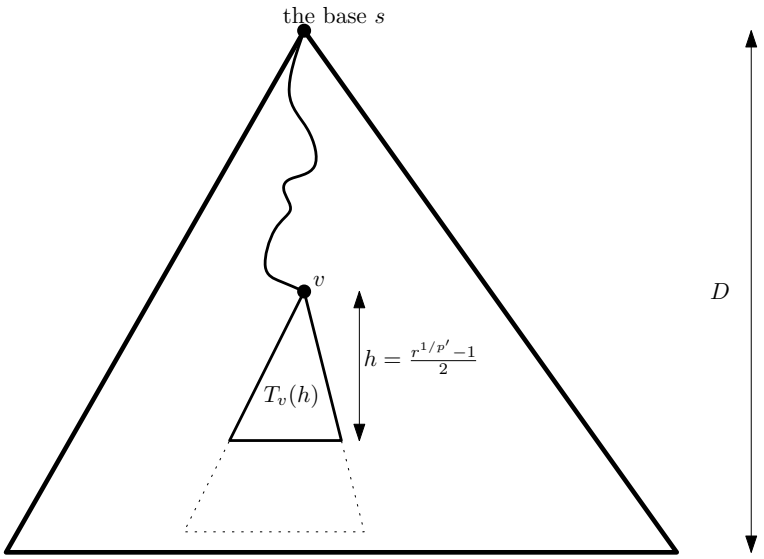

Figure 5.4: Subtree $T_v(h)$ of tree T.

Now we describe the algorithms `BinarySE` and `UnarySE`, where the latter is a small modification of the former.

Algorithm `BinarySE` and its running time

Assume that T is a p-dense tree and consider subtree $T_v(h)$ which is rooted at v and consists of all nodes at distance h from v (see Figure 5.4). By taking small h, we guarantee that the number of nodes in each subtree is bounded. If we take $h := (r^{1/p'} - 1)/2$ for sufficiently large p', then each such a subtree contains at most r nodes, for instance, this happens already for $p' = p$. Indeed, since T is a p-dense tree, according to the definition, each subtree of T, which is $(r^{1/p} - 1)/2$ in height, contains at most

$$\left[2 \cdot \left((r^{1/p} - 1)/2\right) + 1\right]^p \leq r$$

nodes.

In order to test whether height h guarantees that subtree $T_v(h)$ has at most r edges, the algorithm `ExploreSubtree`(v, h) tries to explore this tree using $2r$ steps. At first, it goes from s to v, and then, using a dfs-rule, moves for $2r$ steps along the dfs-sequence of $T_v(h)$. If the subtree has at most r edges, then $2r$ steps suffice to explore this subtree. Otherwise the dfs-traversal is broken, and the robot executing `ExploreSubtree`(v, h) returns to base s. In this case, to get to v it takes at most $h \leq r$. Therefore, in any case the total exploration time of `ExploreSubtree`(v, h) is bounded by $5r$.

If `ExploreSubtree`(v,h) fails to explore $T_v(h)$ for certain h, then it means that the subtree contains more than r edges, and in this case, the height has to be decreased. The `BinarySE` algorithm decreases $h = (r^{1/p'} - 1)/2$ by doubling p', and thus after at most $\log p$ such increases, p' approaches density p. As we have observed above, parameter h is then small enough to guarantee the existence of at most r edges in the subtree $T_v(h)$ which, in this case, is completely explored by `ExploreSubtree`(v,h). Doubling p' may rapidly converge to p, but also may end with an inaccuracy, since the real value p might be over-jumped, such that $2p > p' > p$. The consequences of this observation will negatively influence the efficiency of the algorithm presented below.

Algorithm 6 `BinarySE`(p_{init})

1: $id \leftarrow$ robot's id
2: $r \leftarrow 1$
3: $p' \leftarrow p_{init}$
4: **repeat**
5: $\Phi \leftarrow \Phi(r)$ number of nodes at distance r from s
6: $v_1, \ldots v_\Phi \leftarrow$ all nodes at distance r from s
7: *trees_to_explore* $\leftarrow \{v_r : r = id + k \cdot j \mod \phi,\ \mathrm{j} \in \mathbb{N}\}$
8: $h \leftarrow (r^{1/p'} - 1)/2$
9: `ExploreSubtree`(v,h) for each $v \in$ *trees_to_explore*
10: **if** (for some v tree $T_v(h)$ is unexplored) **then**
11: $p' \leftarrow 2 \cdot p'$
12: **else**
13: $r' \leftarrow r + (r^{1/p'} - 1)/2$
14: $r \leftarrow \operatorname{argmin}\{\Phi(j) : \quad j \in [(r + r')/2,\ r']\}$
15: **end if**
16: **until** (T is completely explored)

The algorithm `BinarySE` (see Algorithm 6) works in the similar way, as described above, but uses $k/2$ robots to explore in parallel many subtrees attached to different nodes in the p-dense tree T. Assume that at the beginning of the i-th *epoch* all nodes up to some distance $r_i > 0$ are visited (and known to robots positioned in s). At the end of this epoch, all nodes up to the distance $r_{i+1} > r_i$ will be explored.

Assume that $v_1, v_2, \ldots v_\Phi$ are all nodes at distance r from base s at the beginning of the epoch ($\Phi = \Phi(r)$ denotes number of nodes at distance r from s). Consider all subtrees $T_{v_j}(h)$ (for $1 \le j \le \Phi$) rooted at those nodes (see Figure 5.5). Parameter $h = (r^{1/p'} - 1)/2$ is assigned like described above, by doubling parameter p', which is a variable common for all robots, initially set as 1. During the epoch $k/2$ robots travel to different nodes v_j, where $1 \le j \le \Phi$, and try to explore subtrees $T_{v_j}(h)$ attached there,

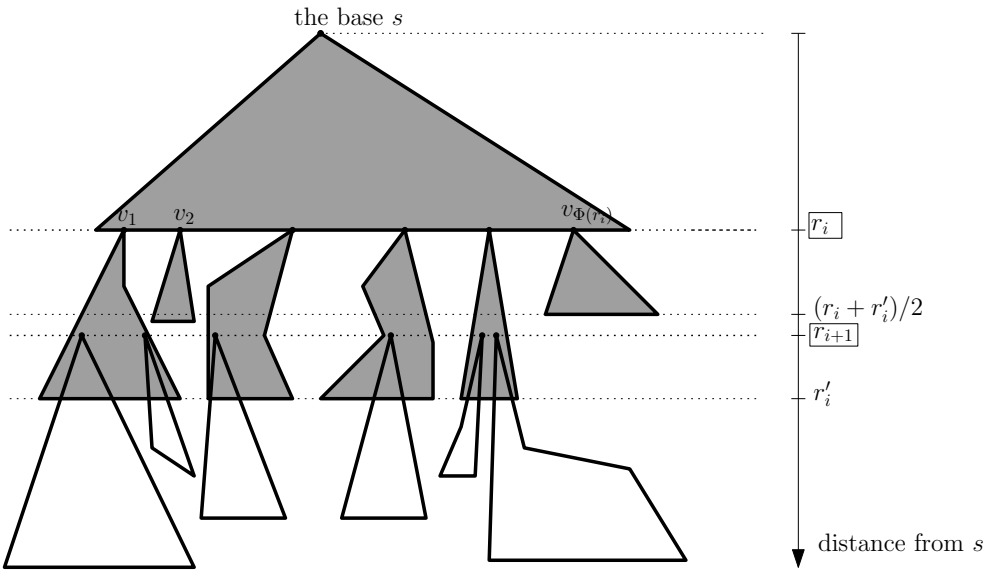

Figure 5.5: The i-th epoch of the algorithm.

using `ExploreSubtree`(v, h) algorithm. If for any robot the algorithm fails, it means that there exists a subtree, say $T_w(h)$, that contains more than r edges. Robots which after the execution of `ExploreSubtree`(v, h) are positioned in s double parameter p' to decrease h, and thus the number of edges in the subtrees. The i-th epoch ends when h is small enough to guarantee that all subtrees are sufficiently small, and therefore all those subtrees are completely explored by `ExploreSubtree`(v, h) algorithm. In this case, at the end of the epoch all nodes up to the distance $r_i + h$ are explored.

Surprisingly, r_{i+1} is not set as $r_i + h$. The value is determined to be the size of a "narrow" place in the tree

$$r_{i+1} \leftarrow \operatorname{argmin}\left\{\Phi(j) : \quad j \in \left[(r_i + r'_i)/2,\ r'_i\right]\right\} ,$$

where $r'_i \leftarrow r_i + (r_i^{1/p'} - 1)/2$. As $r_{i+1} \leq r_i + h$, at the end of the epoch all nodes up to the distance r_{i+1} are explored and known to the team positioned in s.

Denote by p_i the value of the local variable p' at the beginning of an epoch. The parameter p' was doubled $1 + \log(p_{i+1}/p_i)$ times during the i-th epoch, and each such a change requires a test on the size of the subtrees. Each test using `ExploreSubtree` takes $5r_i$ steps, and as there are $k/2$ robots, there are up to $k/2$ tests done in parallel by the team. The following lemma states the total time of an epoch.

Lemma 5.7. *The i-th epoch of* `BinarySE`(1) *algorithm takes*

$$O\left(\frac{\phi(r_i)}{k} \cdot r_i \cdot [1 + \log(p_{i+1}/p_i)]\right)$$

time steps, and at the end, all nodes up to the distance r_{i+1} are visited. Sequences r_i and p_i are defined by algorithm `BinarySE`(1).

By Lemma 5.7 and using $\frac{p_{i+1}}{p_i} \leq 2p$, we sum up over all epochs to obtain

$$\sum_{i\geq 0} \frac{1}{k}\phi(r_i)\cdot r_i \cdot [1+\log(p_{i+1}/p_i)] \leq \frac{1}{k}\sum_{i\geq 0}\phi(r_i)\cdot r_i \cdot 2\log p$$

as an upper bound on the exploration time.

Lemma 5.8. *The* `BinarySE`(1) *explores tree T in time*

$$O\left(\frac{\log p}{k}\sum_{i\geq 0}\phi(r_i)r_i\right),$$

where sequences r_i and p_i are defined by algorithm `BinarySE`(1).

Algorithm `UnarySE`

The algorithm `UnarySE`(1) is a simple modification of `BinarySE`(p_{init}), obtained by replacing (in Algorithm 6) the 11-th line $p' \leftarrow 2\cdot p'$ by $p' \leftarrow 1 + p'$. Therefore, `UnarySE`() adjusts p' more carefully, and we will later see that this might positively influence the competitive ratio of this algorithm. Note here that both sequences p_i and r_i differ from those defined by algorithm `BinarySE`(1).

Lemma 5.9. *The i-th epoch of* `UnarySE`(1) *algorithm takes*

$$O\left(\frac{\phi(r_i)}{k}\cdot r_i \cdot [1+p_{i+1}-p_i]\right)$$

time steps, and at the end, all nodes up to the distance r_{i+1} are visited. Sequences r_i and p_i are defined by algorithm `UnarySE`(1).

We can analogously state the similar lemma for `UnarySE`(1) algorithm. We have $p_{i+1} - p_i \leq p$, and thus

$$\sum_{i\geq 0}\frac{1}{k}\phi(r_i)\cdot r_i\cdot[1+p_{i+1}-p_i] \leq \frac{1}{k}\sum_{i\geq 0}\phi(r_i)\cdot r_i\cdot 2p\,.$$

Lemma 5.10. *The* `UnarySE`(1) *explores tree T in time*

$$O\left(\frac{p}{k}\sum_{i\geq 0}\phi(r_i)r_i\right),$$

where sequences r_i and p_i are defined by algorithm `UnarySE`(1).

Online analysis of `SparseExplore`

First, we show a lower bound for the optimal time, needed by the offline exploration of the same tree T. It is obtained by showing that T contains many nodes (expressed in terms of r_i, $\Phi(r_i)$ and p_i defined by the execution of `UnarySE`(1) or `BinarySE`(1)), and therefore using even all robots in parallel, it takes much time to visit all nodes.

Lemma 5.11. *The optimal offline algorithm needs*

$$\Omega\left(\frac{1}{k}\sum_{i\geq 0}\phi(r_i){r_i}^{1/p_i}\right)$$

time steps to explore T, where sequences r_i and p_i are defined either by `BinarySE`(1) *or by* `UnarySE`(1)*.*

Proof. Define the set $I_i \subset \mathbb{N}$ as follows

$$I_i = \left[(r_i + r'_i)/2,\ r'_i\right] ,$$

and let $\mathcal{I} = \bigcup_i I_i$ be the sum of these sets of levels. Levels I_i do not overlap, since

$$(r_{i+1} + r'_{i+1})/2 > r'_i .$$

Let us count how many nodes n_i there are at distances described by I_i. We know that $r_{i+1} \in I_i$ and that $\forall_{j\in I_i}(\phi(r_{i+1}) \leq \phi(j))$. Then, we have $n_i \geq 1/2 \cdot \phi(r_{i+1}) \cdot {r_i}^{1/p_{i+1}}$, and given $r_i \geq r_{i+1}/2$ we get

$$n_i = \Omega\left(\phi(r_{i+1}) \cdot {r_{i+1}}^{1/p_{i+1}}\right) ,$$

so there are at least $\Omega\left(\sum_i \phi(r_i){r_i}^{1/p_i}\right)$ nodes in the tree.

A group of k robots needs at least $\sum_i n_i/k$ time steps to explore a tree with $\sum_i n_i$ nodes, which concludes our proof. ∎

Before we prove the competitive factor of our algorithms, we need the following observation. For any $0 < \alpha < 1$ sequence $\{x_i\}_i$ and non-decreasing sequence $\{y_i\}_i$ we have

$$\frac{\sum_{i=1}^m x_i y_i}{\sum_{j=1}^m x_i {y_i}^\alpha} \leq {y_m}^{1-\alpha} . \tag{5.1}$$

Indeed, for all $0 \leq i \leq m$ we have $y_i^{1-\alpha} \leq y_m^{1-\alpha}$, and thus

$$\frac{1}{y_m^{1-\alpha}} \cdot \frac{\sum_{i=1}^m x_i y_i}{\sum_{i=1}^m x_i \cdot y_i^\alpha} \leq \frac{\sum_{i=1}^m x_i y_i}{\sum_{i=1}^m x_i \cdot y_i^\alpha y_m^{1-\alpha}} \leq \frac{\sum_{i=1}^m x_i y_i}{\sum_{i=1}^m x_i y_i^\alpha y_i^{1-\alpha}} \leq 1 .$$

Theorem 5.12. *The* `BinarySE`(1) *algorithm achieves the competitive ratio of*

$$O\left(\log p \cdot D^{1/2p} \cdot D^{1-1/p}\right)$$

and `UnarySE`(1) *the ratio of*

$$O\left(p \cdot D^{1-1/p}\right)$$

for a p-dense tree.

Proof. By Lemma 5.8 and Lemma 5.11 and using $1/p_i \geq 1/(2p)$, we find that the competitive ratio of `BinarySE`(1) algorithm is bounded by

$$O\left(\log p \cdot \frac{\sum_i \phi(r_i) r_i}{\sum_i \phi(r_i) r_i^{1/p_i}}\right) \leq O\left(\log p \cdot \frac{\sum_i \phi(r_i) r_i}{\sum_i \phi(r_i) r_i^{1/(2p)}}\right) .$$

By substituting $\alpha = 1/(2p)$, $y_i = r_i$ and $x_i = \phi(r_i)$ in (5.1) we obtain

$$\frac{\sum_i \phi(r_i) r_i}{\sum_i \phi(r_i) r_i^{1/(2p)}} \leq D^{1-1/(2p)} ,$$

and thus the competitive ratio of `BinarySE`(1) is bounded by

$$O\left(\log p \cdot D^{1-1/2p}\right) .$$

Similarly, for `UnarySE`(1) (using Lemma 5.10 and Lemma 5.11) we obtain

$$O\left(p \cdot \frac{\sum_i \phi(r_i) r_i}{\sum_i \phi(r_i) r_i^{1/p_i}}\right) \leq O\left(p \cdot \frac{\sum_i \phi(r_i) r_i}{\sum_i \phi(r_i) r_i^{1/p}}\right) ,$$

having a slightly better bound $1/p_i \geq 1/p$. Substituting $\alpha = 1/p$, $y_i = r_i$ and $x_i = \phi(r_i)$ in (5.1), and bounding in the similar way we finally obtain

$$O\left(p \cdot D^{1-1/p}\right) .$$

■

We have defined and analyzed two algorithms `UnarySE`(1) and `BinarySE`(1), whose performance depends on the ratio between p and D. The first one is better for trees with a small density (comparing to D), and the second is better for trees small in height.

To take advantage of these algorithms, the `SparseExplore` assigns one robot to act as *a referee* which does not move from s and waits, until one of the subteam (which executes either `UnarySE`(1) or `BinarySE`(1)) finishes the exploration. The `SparseExplore` terminates, after one of the groups reports (in the base) a completion of the exploration, and all robots return to s. Denote by $\widetilde{C}_{OPT}(G, s)$ the optimal offline exploration cost. The team executing `UnarySE`(1) needs at most

$$O\left(p \cdot D^{1-1/p}\right) \cdot \widetilde{C}_{OPT}(G, s)$$

time, and the team executing `BinarySE`(1) at most

$$O\left(\log p \cdot D^{1/2p} \cdot D^{1-1/p}\right) \cdot \widetilde{C}_{OPT}(G,s)$$

time steps. This leads to the following theorem:

Theorem 5.13. *The* `SparseExplore` *achieves the competitive ratio of*

$$O\left(D^{1-1/p} \cdot \min\left\{p, \log p \cdot D^{1/2p}\right\}\right)$$

for an arbitrary tree T, where D is the height of T and p the density.

If p is constant, then competitive factor in Theorem 5.13 can be reduced to

$$O\left(D^{1-1/p}\right).$$

The same happens, if a constant approximation of p is known. There is no need for a large number of adjustments of parameter h in `ExploreSubtree`(v_j, h), as it is initially set to an appropriate value guarantying that T_{v_j} contains at most r nodes.

Chapter 6

Conclusions and Outlook

We have discussed the problem of exploration of a graph by a team of robots. One of the properties that distinguish setting studied here from the other graph exploration problems is the physical property of the robot's movement. To change the position in the graph, the robot consecutively traverses all edges along some path between its current and target position. This disables many algorithms like e.g. a classical one-robot breadth-first search algorithm.

Clearly, it is much more challenging to explore unknown environments. We would probably dethrone all great explorers, if it turned out that they have used a complete map of the regions they discovered. The true challenge the unknown offers, rouses people to action. Led by this passion, we study the problem where a team of explorers is positioned in a node of a graph they do not know.

There are many interesting problems which arise here. The main issue we rise in this work is how the lack of a map influences the efficiency of collective exploration. We define cost functions measuring the exploration efficiency, and compare the algorithm's cost to the cost of an optimal exploration, assuming that the map is known – competitive ratio. The competitive ratio shows how well a team of robots copes with the lack of a map. If it was equal 1, this would mean that the team does not need a map to explore optimally. In fact, we show that for each algorithm there exists a graph which is difficult to explore, if not known beforehand i.e. we show a general lower bound on the competitive ratio. We also prove that this happens even if each team member has a global view of the situation of a whole team (global communication).

The communication pattern is another interesting aspect to be considered in the system of entities distributed over the graph. There are results proving that the cooperation by the exploration of certain terrains is not possible at all, if there is no communication granted. In this situation, the whole team explores with the same efficiency as one robot. As our lower bounds on the competitive ratio hold for the global communication

pattern, interesting open question is whether these bound can be further improved, assuming for instance a local communication pattern, where robots communicate only if they are in the same node.

On the other hand, all algorithms presented in this work use a strictly local communication pattern. In fact, robots communicate only within the base station, i.e. an initial position of all robots. As the general lower bounds for the competitive ratio are still smaller that the competitiveness of our algorithms, it might happen, that allowing the global communication pattern, there can be new interesting algorithms designed with improved competitiveness.

A "gap" between the general lower bound and the competitiveness of presented algorithms points out an interesting area of investigations. For the two proposed cost models (the total exploration time and maximal energy used by a robot) we have shown that the gap is significantly larger for the time-related cost. Fortunately, in the energy model the bounds match up to a constant factor, and therefore there is not much space for improvements.

We have observed that the exploration of graphs is almost as "easy" as the exploration of some of their spanning trees (a relaxed SPT). Basing on this observation, we have presented a framework which ables tree-algorithms to explore a wider class of graphs. As an example we have presented the application of this technique by the exploration of a city-block graphs. Applications for different classes of graphs seems to be a promising direction for further research.

The terrain exploration is only one of the applications of our techniques. We believe our results contribute also to other problems in robotics that require a fair distribution of resources (robots) in an unknown terrain (clustering problems). The physical movement property is an important restriction of our model, and we believe that it might be also reflected in different than robotics scenarios. For instance, the exploration of networks or other distributed environments, where a message carries a token, which does not allow it to rapidly change its position within the environment. Moreover, in our setting, it is not admissible that the robots create their own copies, as it is possible e.g. in the case of software agents exploring the network. However, even in the distributed computing there might be some additional restrictions, coming from the security or economical aspects, that forbid to create more than a bounded number of agents. There are also some problems where the number of resources in the system is bounded, and the goal is to use those resources in an appropriate way (e.g. dinning philosophers problem).

The collective exploration belongs to the wider class of problems concerning the cooperation of a group by fulfilling some task in general. The main goals that the team has to fulfill may vary and it might include e.g. the processing of tasks found in the terrain. However, we always desire an increase in the efficiency as a result of cooperation of all members of the group.

Bibliography

[AB96] I. Averbakh and O. Berman. A heuristic with worst-case analysis for minimax routing of two travelling salesmen on a tree. *Discrete Applied Mathematics*, 68(1-2):17–32, 1996.

[AB97] I. Averbakh and O. Berman. (p - 1)/(p + 1)-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective. *Discrete Applied Mathematics*, 75(3):201–216, 1997.

[AB02] I. Averbakh and O. Berman. Minmax p-traveling salesmen location problems on a tree. *Annals of Operations Research*, 110(1 - 4):55–68, 2002.

[ABCP93] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear cost sequential and distributed constructions of sparse neighborhood covers. In *Proc. 34th IEEE Symposium Foundations of Computer Science (FOCS)*, pages 638–647, 1993.

[ABRS99] B. Awerbuch, M. Betke, R.L. Rivest, and M. Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152(2):155–172, 1999.

[AH00] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

[AHL06] E. M. Arkin, R. Hassin, and A. Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59(1):1–18, 2006.

[AK98] B. Awerbuch and S. G. Kobourov. Polylogarithmic-overhead piecemeal graph exploration. In *Proc. of the 11th Annual Conference on Computational Learning Theory (COLT)*, pages 280–286, 1998.

[BBF+96] P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosén, and M. Saks. Randomized robot navigation algorithms. In *Proc. of the seventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 75–84, 1996.

[BEY98] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. 1998.

[BFR+98] M. A. Bender, A. Fernández, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: exploring and mapping directed graphs. In *Proc. of the 30th Annual ACM Symp. on Theory of Computing (STOC)*, pages 269–278, 1998.

[BH74] M. Bellmore and S. Hong. Transformation of multisalesman problem to the standard traveling salesman problem. *Journal of the ACM (JACM)*, 21(3):500–504, 1974.

[BRS91] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. In *Proc. of the twenty-third annual ACM symposium on Theory of computing (STOC)*, pages 494–504, 1991.

[BRS95] M. Betke, R. L. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2-3):231 – 254, 1995.

[BS94] M. A. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proc. of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 75–85, 1994.

[CFK97] Y. U. Cao, A. S. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7 – 27, 1997.

[CLRS90] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 1990.

[DFK+02] S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, and N. Santoro. Black hole search by mobile agents in hypercubes and related networks. In *Proc. of the 6th International Conference on Principles of Distributed Systems (OPODIS)*, pages 169–180, 2002.

[DFKP02] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. In *Proc. of the 13th annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 588–597, 2002.

[DFNS05] S. Das, P. Flocchini, A. Nayak, and N. Santoro. Distributed exploration of an unknown graph. In *Proc. of 12th Structural Information and Communication Complexity (SIROCCO)*, pages 99–114, 2005.

[DFPS02] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: optimal mobile agent protocols. In *Proc. of the 21st Annual Symposium on Principles of Distributed Computing (PODC)*, pages 153–162, 2002.

[DFPS06] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocols. *Distributed Computing*, 19(1):1–18, 2006.

[DFS04] S. Dobrev, P. Flocchini, and N. Santoro. Improved bounds for optimal black hole search with a network map. In *Proc. of 11th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 3104, pages 111–122, 2004.

[DJMW96] G. Dudek, M. R. M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375 – 397, 1996.

[DKK06] C. A. Duncan, S. G. Kobourov, and V. S. A. Kumar. Optimal constrained graph exploration. *ACM Transactions on Algorithms*, 2(3):380–402, 2006.

[DKLM06] M. Dynia, J. Kutyłowski, P. Lorek, and F. Meyer auf der Heide. Maintaining communication between an explorer and a base station. In *Biologically Inspired Cooperative Computing (BICC)*, pages 137–146, 2006.

[DKMS06] M. Dynia, J. Kutyłowski, F. Meyer auf der Heide, and C. Schindelhauer. Smart robot teams exploring sparse trees. In *Proc. of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 327–338, 2006.

[DKMS07] M. Dynia, J. Kutyłowski, F. Meyer auf der Heide, and J. Schrieb. Local strategies for maintaining a chain of relay stations between an explorer and a base station. In *Proc. of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 260–269, 2007.

[DKP98] X. Deng, T. Kameda, and C. H. Papadimitriou. How to learn an unknown environment i: The rectilinear case. *Journal of the ACM*, 45(2):215–245, 1998.

[DKS06] M. Dynia, M. Korzeniowski, and C. Schindelhauer. Power-aware collective tree exploration. In *Proc. of the 19th International Conference on Architecture of Computing Systems (ARCS)*, pages 341–351, 2006.

[DLS07] M. Dynia, J. Lopuszanski, and C. Schindelhauer. Why robots need maps. In *Proc. of the 14th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 37– 46, 2007.

[DP99] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

[DP02] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. In *Proc. of the 10th Annual European Symposium on Algorithms (ESA)*, pages 374–386, 2002.

[Dyn06] M. Dynia. Grupowa eksploracja drzew. Master thesis at Faculty of Mathematics and Computer Science, University of Wroclaw, 2006.

[EGK+04] G. Even, N. Garg, J. Könemann, R. Ravi, and A. Sinha. Min-max tree covers of graphs. *Operations Research Letters*, 32(4):309–315, 2004.

[FGKP06] P. Fraigniaud, L. Gasieniec, D. R. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48(3):166 – 177, 2006.

[FHK78] G. Frederickson, M. Hecht, and C. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

[FIRT05] P. Fraigniaud, D. Ilcinkas, S. Rajsbaum, and S. Tixeuil. Space lower bounds for graph exploration via reduced automata. In *Proc. of 12th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 140–154, 2005.

[FPSW01] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous oblivious robots with limited visibility. In *Proc. of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 247 – 258, 2001.

[FT05] R. Fleischer and G. Trippen. Exploring an unknown graph efficiently. In *Proc. of the 13th Annual European Symposium on Algorithms (ESA)*, pages 11–22, 2005.

[GBH97] N. Guttmann-Beck and R. Hassin. Approximation algorithms for min-max tree partition. *Journal of Algorithms*, 24(2):266–286, 1997.

[HAB+02] T. Hsiang, E. Arkin, M. Bender, S. Fekete, and J. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *In 5th International Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 77– 94, 2002.

[HM84] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. In *Proc. of the Symposium on Principles of Distributed Computing (PODC)*, pages 50–61, 1984.

[Kat05] B. Katreniak. Biangular circle formation by asynchronous mobile robots. In *Structural Information and Communication Complexity*, pages 185–199, 2005.

[KTI+07] Y. Katayama, Y. Tomida, H. Imazu, N. Inuzuka, and K. Wada. Dynamic compass models and gathering algorithms for autonomous mobile robots. In *Proc of the 14th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 274–288, 2007.

[NO04] H. Nagamochi and K. Okada. A faster 2-approximation algorithm for the minmax p-traveling salesmen problem on a tree. *Discrete Applied Mathematics*, 140(1-3):103–114, 2004.

[PP98] P. Panaite and A. Pelc. Exploring unknown undirected graphs. In *Proc. of the 9th annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 316–322, 1998.

[Pre01a] G. Prencipe. Corda: Distributed coordination of a set of autonomous mobile robots. In *In Proc. of European Research Seminar on Advances in Distributed Systems (ERSADS)*, pages 185–190, 2001.

[Pre01b] G. Prencipe. Instantaneous actions vs. full asynchronicity: Controlling and coordinating a set of autonomous mobile robots. In *Proc. of the 7th Italian Conference on Theoretical Computer Science (ICTCS)*, pages 154 – 171, 2001.

[PY91] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[RKSI93] N. Rao, S. Kareti, W. Shi, and S. Iyenagar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, 1993.

[She92] T. C. Shermer. Recent results in art galleries. *Proc. of the IEEE*, 80:1384–1399, 1992.

[SS96] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots, 1996.

[ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[SY96] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots—formation and agreement problems. In *Proc. of the 3rd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 1996.

[SY99] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.