UNIVERSITÀ DEGLI STUDI DI ROMA "LA SAPIENZA"

DOTTORATO DI RICERCA IN INGEGNERIA INFORMATICA

XVIII CICLO – 2005

Mobile Agents Security: Algorithmic Approaches for Hostile Hosts Detection

Fabiano Sarracco

UNIVERSITÀ DEGLI STUDI DI ROMA "LA SAPIENZA"

DOTTORATO DI RICERCA IN INGEGNERIA INFORMATICA

XVIII CICLO - 2005

Fabiano Sarracco

Mobile Agents Security: Algorithmic Approaches for Hostile Hosts Detection

Thesis Committee

Prof. Umberto Nanni (Advisor)
Prof. Giorgio Ausiello
Prof. Francesco Donini
Prof. Michele Flammini

Reviewers

Prof. Shmuel Zaks
Prof. Giorgio Gambosi

AUTHOR'S ADDRESS:

Fabiano Sarracco
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
E-MAIL: sarracco@dis.uniroma1.it
WWW: http://www.dis.uniroma1.it/∼sarracco

Acknowledgments

First of all, I would like to thank my advisor Umberto Nanni for his precious support and for the huge number of things he taught me during these years.
I wish to thank also the other members of my Thesis Committee: Giorgio Ausiello, Francesco Donini and Michele Flammini. I am sincerely grateful for their suggestions; I only regret we could not spend more time discussing together.
A very big thank goes to the external reviewers, Shmuel Zaks and Giorgio Gambosi, whose comments and suggestions have been very useful to improve the contents and the presentation of this thesis.
My biggest debt of gratitude goes to Ralf Klasing, Tomasz Radzik and Euripides Markou. Without them, without the hours we spent on discussing together, probably this thesis would have been a bunch of blank sheets.
I also wish to thank Ralf and Nathalie for making me feel at home even when I was thousands of kilometers away from my family (and not only for the moka coffee!).
Obviously, I have to thank also the people who invited me and financially supported my studies abroad: Jean-Claude Bermond (for the unforgettable months I spent in Sophia Antipolis) and Cyril Gavoille (for inviting me to Bordeaux).
While drawing these concentric circles of gratitude, I cannot avoid to mention all my friends at the Department in Rome. A special thank goes to the people with which I shared most of the working time and most of the doubts as a Ph.D. student, in the chaotic room 253: Luigi Laura, Andrea Vitaletti and Debora Donato.

A very special thought goes to my mother Carla and to my unforgettable father Virgilio. I hope this work will reward you for all the sacrifices you made to allow me to carry on my studies. Thank you also because if you had not buy that personal computer when I was only five years old, probably today I would not be a computer scientist.
Finally, I wish to thank Annalisa, Gabriele and Edoardo, my endless sources of joy. This work is dedicated to them.

Rome, Italy — Fabiano Sarracco
December 9th, 2005

Contents

Abstract

Mobile agents are software programs that are able to migrate through the hosts of a network and perform operations on them. Both theoretical works and implementations showed how a mobile agent framework can bring considerable advantages in many network services and applications like, for example, transaction processing and distributed information retrieval. Others have pointed out, however, that mobile agents introduce severe concerns for security. Recently there has been an increasing interest in the use of mobile agents for network computations, from an algorithmic perspective. In this work we study an algorithmic problem related to mobile agent security, known as *black hole search problem*. A black hole is a host destroying all the mobile agents visiting it, without leaving any trace. During a black hole search, a set of agents explores the network in order to detect all the black holes in it. Our target is to design optimal (fastest) black hole search protocols. We study two slightly different models; we rigorously formalize them and relate them to similar problems presented in the literature. We provide the best currently known upper and lower bounds for the complexities of the two problems. We also consider networks with some particular topologies, providing improved upper bounds for them.

Chapter 1

Introduction

In a distributed system, several computers or processors (also called *nodes* or *hosts*), interconnected by some communication hardware, accomplish a common task by exchanging informations between them. Even though the analysis and the implementation of distributed systems have manifested both theoretical and practical difficulties, which in some cases showed to be very hard to overcome (when not proved to be unsolvable at all), many of the computer applications we use in our daily routines involve some form of distributed computing. For instance, the *client/server paradigm* we use when we exchange e-mails, fetch Web-pages or download music from an Internet music store; not to mention the dizzy development of the *peer-to-peer paradigm* boosted by file-sharing and Voice-Over-IP applications.
One of the most challenging issues in distributed computing is related to the problem of *failures*. We cannot avoid to quote the famous Lamport's observation which every computer user had a chance to verify:

> *A distributed system is one in which the failure of a computer you did not even know existed can render your own computer unusable.*

Notice that, when using the term "failure", we actually refer to a more general notion which covers also concepts not strictly related to malfunctioning like disconnections or malicious behavior. In this direction, various sources have observed that the classical distributed paradigm is sometimes too "static" to accommodate the dynamics of open and large systems, like for instance the Internet, where a number of nodes continually join the network and disconnect from it. Moreover, the distributed paradigm of nodes exchanging messages between them seems to be unsuited for some applications like, for instance, transaction processing.
Let us take the classic example of a customer who wants to buy a flight ticket. He has some requirements (e.g., destination, dates of travel, seats number and classes), and wants to select, among a set of distinct offers from various companies, the best offer fulfilling his requirements. In a classical scenario, the customer should accede to the

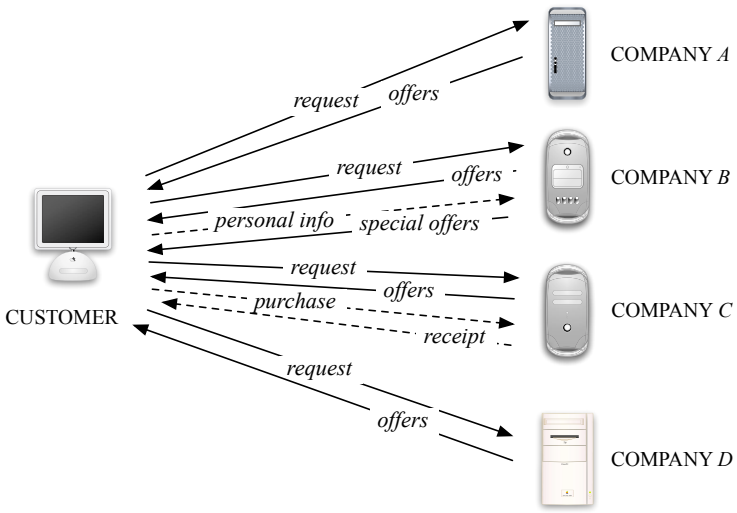

Figure 1.1: A schematic representation of the messages exchange among parties in a typical flight booking transaction. Messages represented by dashed lines contain confidential informations and thus need a secure transmission.

Web and contact the airlines websites. For each contacted company, he should provide it his requirements, and collect all its offerings. Moreover, the customer wants to provide the necessary personal informations to companies from which he can benefit by special low fares (e.g., if he is a frequent flyer). After collecting all the offerings, the customer selects the best one, according to some utility metric, contacts again the providing company and buys the ticket, by sending the necessary payment data (e.g., his credit card numbers), in a secure way. Finally, the company sends to the customer a receipt, containing detailed flight informations and possibly a confirmation code. The full process is schematically represented in Figure 1.1. This schema should give a rough idea of the number of messages that the parties have to exchange in order to accomplish the task. Moreover, the process could be made even more complex: the flight might be purchased in a non-atomic way (e.g., the best way to go from Rome to New York is to fly from Rome to London with *ACME-jet*, and from there to New York with *Rockerduck Airways*) or they might be offered through an auction process. The drawing becomes inextricable in the case that a group of customers, each with different requirements, wants to travel together on the same flight.
It is clear that things could be much easier if some entities (e.g., the customer), can be "transferred" to the places where their presence (in terms of knowledge and decisional power) is needed. Obviously, real entities cannot move through hosts, or at least it is very expensive for them to move. This is the main motivation of *mobile*

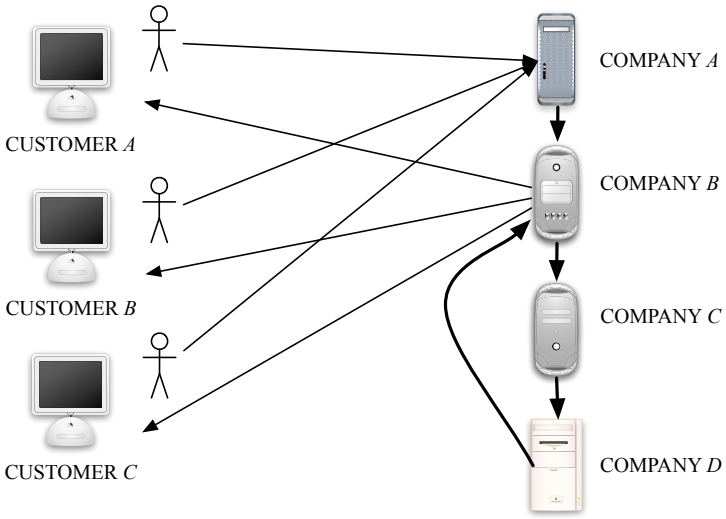

Figure 1.2: A schematic representation of the group flight booking process, performed through a mobile agents system.

agents systems. In a mobile agents system each entity involved in the process can create a “software alter-ego”, the *mobile agent*, which can move through the hosts in the system and act on behalf of its owner.
In Figure 1.2 we can see an example of how the group flight booking task can be performed by means of a mobile agents system. Each customer creates an agent containing all the necessary informations: the flight requirements, the list of companies to contact, personal informations for special fares, a criterion to score each offer, and data allowing the agent to pay for the ticket. Then the agents start their task; they can, for instance, visit the companies on their own and finally meet and agree on the best offering. Or they can meet initially and then perform the enquiries together. Finally, the agents get back to their respective owners, bringing them the receipts of the tickets acquired.

The introduction of mobile agents systems has opened many interesting opportunities. The most widespread use of mobile agents in network environments, from the World Wide Web to the Data Grid, is clearly to search, i.e., to locate some required ”item” (e.g., information, resource, ...) in the environment. Agents are also used for collaborative applications and data mining, where the data space is huge. In such applications, moving an agent to the server reduces network connectivity, eliminates latency and overcomes limitations imposed by firewalls. Other application domains include network management [39] and e-commerce. Just to cite one, in [32], a hypermedia electronic newspaper system is presented, based on mobile

agent technology. The mobile agents platform is run both by information providers (vendors) and by information consumers (readers). The vendors encapsulate articles and all their related content (like, for instance, the metadata) into *article agents*. The readers use agents to search and collect among the different information providers the informations related to their interests. Through public-key encryption, the system enforces copyright control and payment on a "pay-per-use" basis.

The introduction and usage of mobile agents systems has posed a number of new theoretical and practical problems to solve. Much effort, for instance, has been spent by AI and Logics researchers in order to design formal languages to describe the operations the agents have to execute (*agent programming languages*) and languages to allow agents to represent their knowledge and communicate between them (*knowledge representation languages*). However, the common belief (see [24, 33, 36, 40]) is that one of the main obstacles to the widespread use of mobile agents systems is security. Security of hosts from agents: apparently nothing can distinguish a "well-behaving" agent from a harmful virus/worm hiding behind the appearance of an innocuous agent. Security of agents from hosts (or other agents): in the flight booking example given before, an agent brings with him reserved informations about the owner, financial data, and the offers collected so far. A malicious company can for instance spy the offers from competitors in order to win the contest, or even force the agent to accept its offer, even if it is not the best one.
Security problem is twofold: not only proper security mechanisms have to be assured, but we need users to become confident enough in these mechanisms to send reserved informations through untrusted hosts and, at the same time, allow foreign programs to migrate and execute on their machines. A valid measure against some of the above mentioned security threats is cryptography (see [33, 36]). Public key encryption can be used for instance to preserve confidentiality of informations stored in the agents; it can also be used to provide authentication for hosts and agents. Still cryptography cannot solve all the problems. One of the most serious menaces in mobile agents systems is represented by malicious hosts. When an agent is loaded into a host, the host gains full control over the agent and, at least in some cases, nothing can prevent him from altering agent's behavior, or even terminate the agent. The fact that an agent may be eliminated by a (possibly competitor) host can be of enormous importance in some applications: for example, in a time-limited open-bidding auction, where a single unique item is sold to the best offerer (e.g., *E-bay* auctions). In [26] it is stated:

> *The protection of a mobile agents from malicious hosts is – at least from the viewpoint of the owner of the agent – as important as the protection of the host from malicious agents. As we will see [...] no technical approaches to solve this problem without special secure hardware exist so far. The solubility of this problem which is called the problem of malicious hosts is even estimated to be very low.*

The main contribution of this thesis is to provide techniques aimed to eliminate this problem. Our approach is algorithmic, in the sense that, given a formal definition of a problem, we are interested in finding an algorithm to solve it which uses the minimal set of resources (i.e., time, agents, etc.). Algorithmic approaches have been proposed, within mobile agents systems for other network exploration problems targeted, for instance, to build network maps ([1, 5, 6, 20, 13, 34]), or to search something (an information, other agents, or an intruder) in the network ([2, 3, 25, 4]). We focus our attention on harmful hosts which destroy all the mobile agents visiting them upon their arrival, leaving no observable trace of such a destruction. Because of their property of allowing inward migration and inhibit outward migration, and because of the fact that no trace of them is visible from the outside, these hosts have been called *black holes*. It should be clear that any approach directed to get rid of black holes cannot avoid to first identify them in the network. It should be clear, at the same time, that there is no way of recognizing a black hole without "sacrificing" any agent. Indeed, any black hole in the network requires that a distinct agent visits it and vanishes. The black hole search problem has been presented and analyzed by Dobrev *et al.* in [14, 15, 16, 17, 18, 19]. They consider an asynchronous model where no timing assumptions on the system can be done. In this scenario, the black hole search is very difficult and even unfeasible in some cases. The authors restrict their attention to searches performed by two agents, in networks containing exactly one black hole. Among other results, they show that $\Theta(n \log n)$ traversals are needed to solve the problem in arbitrary networks.
In our work we consider systems in which it is possible to fix an upper bound on the time needed by an agent to migrate through hosts. We also put some further realistic assumptions: we assume, for instance, that the two agents have a complete map of the network to explore and that they can communicate only when they are in the same node. This scenario allows us to give a neat formulation of the task. In this case, in fact, it is possible to provide to each agent a suited sequence of nodes it has to visit. The existence of the black hole can be indirectly deduced by the agents since periodical meetings between the two are scheduled in the sequences; the agent vanishing into the black hole will at some time miss one of these meetings. The target is to provide for a given network, the fastest pair of sequences for such a network, i.e., sequences allowing to infer the location of the black hole (or its absence) in the network in the shortest time. In some cases we could be acquainted with a set of safe nodes in the network, i.e., nodes which are surely not black holes; for example, they could be "trusted" hosts. We consider two different models, characterized by the fact that such information is available or not (respectively denoted as *general* and *restricted*). The restricted problem has been introduced in [12], where the authors provide optimal algorithms for the problem in the case that the network has a line topology and a particular tree topology. They also provide a $5/3$-approximation algorithm for tree networks and a 4-approximation algorithm for general graphs. The authors conjectured **NP**-hardness for the problem. The same authors introduced the

general problem in [11], showing its **NP**-hardness an providing a 9.3-approximation algorithm. For both problems, we show their **NP**-hardness, also in the case that the network can be represented as a planar graph, and their **APX**-hardness (in this case we show two explicit approximation lower bounds). For the case where a subset of the nodes of the network are known to be safe, we provide a 6-approximation algorithm. A better approximation algorithm (with ratio $27/8$) is shown for the problem where such knowledge is dropped. Within the restricted model, we also consider networks with particular topologies (trees, cycles, highly connected graphs), presenting improved black hole search algorithms for them.
We believe that the original results presented here provide a satisfactory answer to the menace brought by black holes; however, the topic is far from being closed and we believe that there is still room for further research. For instance, it is possible to consider more general (and thus complex) models by removing or changing some of the assumptions we considered here. Just to cite an example, a model in which the topology of the network is unknown, better fits in contexts, like the Internet, where such knowledge is impossible or very expensive to obtain. We discuss this further in the last concluding chapter.

The contents of the thesis is organized as follows.

- In Chapter 2 we describe mobile agents systems, providing some basic terminology, and a general execution model for a mobile agents system. We present an overview of the algorithmic problems presented in the literature somehow related to mobile agents. We devote the last section to *mobile agent security* issues: the target is to classify the various threats a mobile agents system has to manage and to spot which countermeasures have been proposed for them.

- In Chapter 3 we introduce the problem we study in this thesis, that is, the *black hole search* problem. We show how this is an ineluctable approach to face one of the most severe issues in mobile agents security: the presence of malicious hosts. We overview related "state of the art" works by showing differences and similarities with our work. Then we detail the model we considered and define the terminology we use in the rest of the thesis. Some preliminary lemmas and observations are presented in the last section of the chapter.

- In Chapter 4 we analyze the complexity of the general black hole search problem. We show that the problem is **APX**-hard by providing an explicit approximation lower bound; we then provide a 6-approximation algorithm.

- In Chapter 5 we study the restricted version of the problem, in which only the starting node is initially known to be safe. We show that in this case the problem is **NP**-hard for planar networks and **APX**-hard for arbitrary networks. We then present a $\frac{10}{7}$-approximation algorithm for arbitrary tree networks and a

$\frac{27}{8}$-approximation algorithm for arbitrary networks. Moreover, we show improved algorithms for networks with particular topologies, among which an optimal algorithm for ring networks.

- Finally, Chapter 6 is devoted to recall the results presented in this thesis and to lead the way for further works on black hole search and, in general, on algorithmics for mobile agents systems.

Chapter 2

Mobile agents system

2.1 What is a mobile agents system?

In a distributed system, several computers or processors (commonly called NODES or HOSTS), interconnected by some communication hardware, accomplish a common task by exchanging informations between them. A mobile agents system follows an alternative approach: instead of moving packet of data between stationary processes, the task is accomplished by moving agents through the nodes, and by allowing agents to interact.
MOBILE AGENTS are entities that are able to migrate from one node to another and to perform operations once they are loaded and executed into the nodes, thus fulfilling the task they were programmed for. They generally consist of code, data (e.g. instance variables) and control informations that allow them to carry on the execution on the nodes they visit.
MOBILE AGENTS PLATFORMS constitute the environment where the agents exist and operate. Their primary task is to run the agents that are present in the system, and to provide well-known services and operations to the agents. The platform thus realizes a virtual machine for agents; it may be a particular operating system, but preferably agents should be operating system independent. Most mobile agents systems today base the agent platform on the Java Virtual Machine. A fundamental requirement for a mobile agents platform is to allow agents' migration between host; it has to provide a mechanism to marshal and send agents to other hosts, and, on the other hand, to recover migrated agents and resume their execution.
Another important facility offered by mobile agents platforms is communication. As we will see, even if we were equipped with a (virtually) unlimited number of mobile agents, some problems would remain unfeasible unless we provide a way for the agents to interact between them. Depending on the model considered, communication between mobile agents is allowed:

- when they are located in the same agent platform;

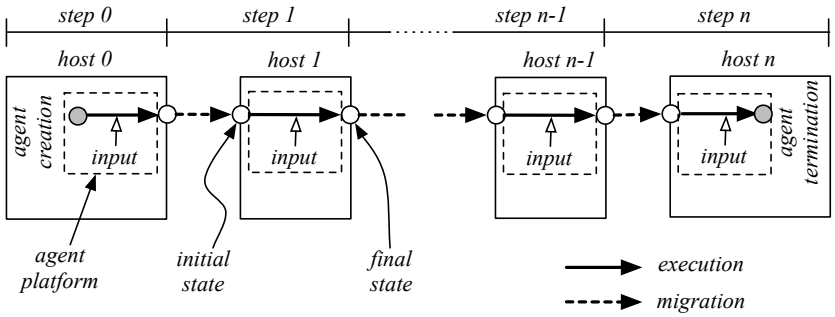

Figure 2.1: Agent execution model

- when they are within a certain range from each other (usually, when they are in two adjacent hosts);
- by leaving messages in agents platforms: in a typical scenario, each platform provides a bounded amount of storage, called WHITEBOARD; agents communicate by reading from and writing on the whiteboards.

A MOBILE AGENTS SYSTEM is a set of homogeneous mobile agents platforms executing on interconnected hosts.

We recall from [27] a typical execution model for mobile agents (see Figure 2.1). A mobile agents platform (typically hosted by the owner of the agent) creates the agent and provides it the data and the initial execution state; then it starts the execution session. During such session, the platform processes the agent using the code and some input, and produces a resulting agent state. The input includes all the data injected from the outside of the agent, i.e. both communication with other agents and data received directly by or via the current host. Examples of inputs from host are results from system calls like random numbers or the current system time. When the agent migrates to another host, the execution session is finished on this platform. The resulting state produced by one host is used as the initial state on the next host. We denote as STEP the migration on a host, together with the whole execution session on such host. By extension, step 0 covers the first execution session, where the agent is created.

What we can assume on the length of a step in a mobile agents system, determines one of the deepest distinction among the various models considered. Similarly to a distributed system, we can consider three different timing models: asynchronous, synchronous and partially synchronous systems. In the context of distributed systems, this definition has been applied to various aspect of the computation. Synchronicity, for instance, can be related to the message passing system (see [38]): a send event and the corresponding receive event are coordinated to form a single tran-

sition of the system, i.e., a process cannot send a message unless the destination of the message is ready to accept the message. Others (see [31]) identify a synchronous model as the one in which the execution progresses in synchronous rounds, driven by a common clock. In the mobile agents systems scenario, it is usual to relate synchronicity to the length of the steps of the agents' execution. In particular, a mobile agents system model is:

- **synchronous**, when each step (migration + execution session) requires a fixed amount of time;
- **semi-synchronous** or **partially synchronous**, if the time needed by a step is not fixed, but it can be upper bounded;
- **asynchronous**, if each step takes a finite but unpredictable amount of time.

Like in distributed systems, synchronous mobile agents systems represent a rather unrealistic abstraction of actual systems, and in many cases, it is impossible or at least inefficient to implement measures able to "synchronize" a system. However the study of synchronous models is justified by some important observations. Impossibility results obtained on the synchronous model extend directly to less well-behaved models. Moreover, as in the cases considered in this thesis, the analysis of some optimization problems often leads to completely different approaches and results, according to the timing assumption adopted.

2.2 Algorithmic problems within mobile agents systems

In recent years, a number of algorithmic questions has been raised on the use of mobile agents to compute in networked environments. In this section we provide a (necessarily incomplete) overview on such works.
The fundamental question is to determine which minimal resources are needed to solve a given problem. Generally, when analyzing the efficiency of presented algorithms, some different, and somehow complementary, cost measures can be taken into account. According to the classification given in [18], the main measures of complexity are:

- **size**: this is a measure of how many physical resources are needed to accomplish the task; usually, this corresponds to the (maximum) number of agents needed by the protocol, more detailed analysis consider also the storage space available within the agents and their computational power;
- **cost**: which is generally computed as the total (worst case) number of moves performed by the agents; this measure can have a remarkable meaning when there exists a monetary cost associated with each migration;

- **time**: in the context of network algorithms, this is a rather tricky concept; while in synchronous systems each step requires one time unit, and thus a common measure of time can be used (also called IDEAL TIME), in partially synchronous systems, the usual approach is to assume that each step requires *at most* one (normalized) time unit (this is the notion of BOUNDED DELAY); if the network is completely asynchronous, the most targeted time measure is the CAUSAL TIME, i.e., the length of the longest chain of causally related step, over all the possible executions.

In these works, a network is generally represented as a connected graph $G = (V, E)$, where nodes denote hosts and edges denote communication links. In this context, the terms graph and network, host and node, and link and edge are used interchangeably. We will adopt this terminology, although we tend to use the terms graph, edge and node to refer to the abstract representation of a network.

A very common and widespread use of mobile agents is as "cartographers", that is, to explore in order to obtain maps of existing networks whose topology is unknown, or maps of dynamically changing networks like, for instance, the Internet, the Web or ad-hoc mobile networks. Indeed, the problem of efficient exploration of a graph has been introduced and studied in the algorithmic community even before the development of mobile agents systems area. In these early works, the concept of graph is used to model much broader contexts (e.g., rooms with obstacles or the streets of a city) and the concept of finite state automaton (usually called *robot*) replaces the concept of agent; obviously, no trace of multiple agents protocols and communication mechanisms can be found in these works. According to the measures defined before, all these algorithms have optimal size. Note also that, in the case of single agent protocols, the concepts of cost and time overlap, at least for synchronous and semi-synchronous networks.
If the graph is undirected (i.e., the links of the network can be traversed in both directions), and its nodes are uniquely labeled, the problem of exploring and mapping a graph can be easily solved in time linear in the number of edges by depth first search; improved bounds have been presented in [34]. The fact that the underlying graph is directed, does not make the problem harder, since a greedy search algorithm can solve it in time $O(|V| \cdot |E|)$. Also in this case, more sophisticated techniques ([1, 13]) lower this bound.
Hardest problems arise in the case that the nodes of the network are not uniquely globally labeled (a *local* unique labeling of the links outgoing of each node is still needed anyway). It has been proved in [20] that the problem is unsolvable (the protocol might not terminate) unless the agent is provided with a way to mark and distinguish between nodes. For this purpose, the authors of [20] introduce the concept of *pebble*, a device with which an agent can mark a node (by dropping the pebble in it) and later identify the marked node when visiting it again; they show that an agent

provided with a single pebble can map an undirected graph in time $O(|V| \cdot |E|)$, repeatedly by marking nodes and backtracking. For what concerns directed graphs, in [6] it is shown that a robot cannot map graphs in polynomial time using a constant number of pebbles if it does not know a bound on $|V|$. The authors of [5] show that an agent provided with a single pebble and an upper bound $\hat{n}$ on $|V|$, can map any strongly connected graph in time $O(\hat{n}^2|V|^6d^2)$, where d is the maximal out degree of the graph. They also show that in the case that the bound $\hat{n}$ is not known, $\Theta(\log\log|V|)$ pebbles are necessary and sufficient.
We have to mention the paper [6], also because the authors introduce a different approach than the use of pebbles: they consider exploration performed by two agents. They show that two cooperating robots can explore and map an unknown directed graphs with n indistinguishable nodes in expected time polynomial in n, without any prior knowledge on the size of the graph. They also demonstrate that two robots are strictly more powerful than one robot with a constant number of pebbles. In this paper it is assumed that the two agents are initially placed in the same node (called *homebase*).

Mobile agents are also used to search for informations or other agents in the network. In [2] the rendezvous search problem in graphs is studied. In this problem, two agents are randomly placed in nodes of a graph; their target is to plan a walk in the network which minimizes the expected number of steps required to meet. Each agent knows both the graph and its position, but not that of the other agent. The authors consider two cases, the *symmetric* rendezvous problem, where the agents are constrained to use identical strategies, and *asymmetric* rendezvous, when the agents' strategies may differ. This is the graph restricted version of the more general rendezvous search problems, where two unit speed robots have to meet in a (continuous) metric space. They recall and extend the notion of *rendezvous value*, that is, the least expected time required for the two agents to meet; then, by mean of a probabilistic analysis, they show optimal randomized strategies for particular classes of graphs, like Hamiltonian graphs, for the asymmetric case, the complete graph K_3 and cycle graphs C_n for the more difficult symmetric case.

In [3] another optimization problem is presented. There, the target is to "wake up" a set of mobile agents, starting with only one "awake" agent. An agent can awake another one only when the two are in the same host. As soon as an agent is awake, it can assist in the awakening task. The time it takes for its traversal is associated to each link. The goal is to compute an awakening schedule which minimizes the makespan, i.e., such that the last agent is awakened in the shortest time. Because of its reminiscence of the children's game, it has been named *the freeze-tag problem*. The authors hint at a broad set of applications for such problem, like dissemination of informations, routing, scheduling and network design (an optimal scheduling is a minimum depth spanning binary tree of the graph). Among the other results, they show that the problem is **NP**-hard, even for the case of star graphs with one agent in each host. Moreover, while there exists a PTAS for star graphs, the problem is

APX-hard even for graphs of maximum degree $\Delta = 4$. The authors also consider the on-line version of the problem, where the awake agents ignore both the topology of the network and the location of the remaining agents. In this case they present a $O(\log \Delta)$-competitive algorithm.

In [25], the authors study the problem of searching for an item s contained in a node of a network, by mean of a mobile agent. The agent can move along the nodes of the network and ask to each node which is the successor in the shortest path to s. The problem becomes non-trivial since it may happen that a (bounded and unknown) set of nodes (called *liars*), give bad advice. They show deterministic algorithms to solve the problem in time at most polynomial in the number of liars.

In [4],the authors consider a problem which is somehow related to mobile agents security. The problem is to capture a (possibly hostile) intruder in a network by using a team of mobile agents, all starting from an initial homebase. Both the intruder and the agents move along the network links, but the intruder can be arbitrarily fast, and aware of the position of the agents. The intruder is captured as soon as it is surrounded by the agents and has no escape link. The problem is to design an efficient agents' strategy for capturing the intruder. For both practical and historic reasons, the main efficiency parameter is size, i.e., the number of agents in the team. Indeed, this is a special case of the well-known *graph-searching* problem, first discussed in [8]. There, we are given a network whose links are all "contaminated", the target is to decontaminate (clear) all links. A link (u, v) is decontaminated when an agent traverses it from u to v and remains clear only if another agent remains in u or all the other links incident to u are clear.
The authors show that the problem of minimizing the size of the chasers team is **NP**-complete in arbitrary networks. They hence focus their study on tree networks, considering also the weighted case, where the number of agents required to keep a link clear is not unitary. For both cases they give linear time algorithms that compute a search strategy requiring the minimum number of agents.

State of the art works on algorithmic approaches for hostile hosts detection, and thus related to black hole search, are presented in Section 3.3.

2.3 Mobile agents security

In most new technologies, designed with functionality as the main target, security measures are often a blanket added afterward; think, for example, to Internet protocols. In some of these cases this intervention brings some issues related to performances, compatibilities and security exploits.
This was not the case with mobile agents, since it is evident that any non trivial mobile agent implementation must address security as one of the main concerns.
By looking at the security requirements and the suggested solutions, it appears evident that security in mobile agents is an area for further research.

In [36] and in [40] the authors present an analysis of the security issues related to mobile agents, and in both of them some indications on how to solve at least some of these issues is provided. While in [36] there are some indications of technologies that turn out to be useful against these threats, like for instance sandboxes and cryptography, in [40] the focus is on describing the various kind of attacks by the mean of a calculus for mobile computations (the *Seal* calculus).

When speaking of security in mobile agents systems, a thing to remark is that the security concerns are focused to two symmetrical targets.

- Threats directed toward **host**, i.e., toward the resources of the machine on which mobile computation is executed;
- Threats directed toward **agents**, originated from an opponent listening on the network, from other mobile computations, and even from host themselves.

Orthogonally, we can provide a rough classification of the kinds of attack that can have place in mobile agents systems. These can be classified into the following categories.

- **Unauthorized disclosure**. Both an host and other agents may have strong incentives to access, and possibly spread, confidential data and informations stored within an agent. Analogously, an agent must be prohibited from access to any data on the host that it is not authorized for.
- **Unauthorized modifications**. An host can damage the agent by changing its code or data, and make it assume a corrupted behavior; this can be performed, for instance, to launch attacks on other hosts or simply to make the agent act in a way that is advantageous for the host (e.g., in distributed auctions where several host offer an object or a service the agent wants to buy). An agent can attack the host by reducing the availability of some shared system resource. Usual targets for these attacks are CPU, memory and network bandwidth.

The main security measure to ensure integrity and confidentiality is proper access control. Other measure may enhance this, like checksums and encrypted storage.

In this paper we focus on threats directed toward agents, [40] gives the following further classification.

- **Exogenous threats**. These are attacks that occur outside of the agent platform, either while the agent is transferred over the network or stored on disk.
- **Endogenous threats**. These threats are specific to an agent platform, and can be further divided in:
  - **Horizontal hostility**, i.e., attacks between agents running on the same host;

– **Vertical hostility**, i.e., attacks performed by the execution environment of the runtime system of the agent platform, commonly referred to as hostile host attacks.

Hostile hosts have been presented quite pessimistically in the mobile agent literature [9]. Even if security requirements are basically symmetrical, with both agent and host looking to protect their respective resources, data and services, control is asymmetrical. The host which executes an agent must see the agent's internal workings and has full control of the agent. Therefore, little can be done to protect a program from a machine that decodes and executes every single instruction of the agent's program. Given enough time, an attacker will be able to analyze the inner workings of any agent and subvert its intended meaning.

Chapter 3

The black hole search problem

3.1 Introduction

We discussed in Section 2.3 of the various attacks that a malicious host can bring to an agent. We observed that, among all these threats, the ones in which the agent execution is menaced are the most harmful and difficult to prevent. In [36] it is stated:

> *It is virtually impossible to prevent a host from capturing an agent and prevent it from further execution. Measures may be taken to track the progress of agents to at least detect such situations, and possibly identify the responsible host.*

The threat posed by black holes falls within this category. A BLACK HOLE is a node in a network which contains a stationary process destroying all mobile agents visiting the node, without leaving any trace outside the host. Mobile agents cannot prevent being annihilated once they visit a black hole. The only way of protection against such threat is to identify all the black holes in the network and thus avoid further visiting them. However, no hint about the presence of a black hole can be deduced by visiting its neighborhood, and it is also assumed that an agent visiting a black hole has no way of communicating with other agents before being terminated. Therefore, it should be clear that it is possible to locate a black hole only by "sacrificing" one agent and by using another agent to indirectly infer the existence of a black hole. An agent which is to visit an unknown node can, for instance, schedule a meeting with another agent after such visit, or write on a whiteboard in a neighbor node the label of the unknown node that he is visiting. If the visited node is a black hole, then the destroyed agent will neither turn up at the node where the meeting was scheduled nor write back to the whiteboard that the node has been successfully visited. In both cases the surviving agents can deduce that the visited node is a black hole. In Section 3.2 we delineate the model we study in this dissertation. In Section 3.3 the current state-of-the-art works on hostile hosts detection are presented. In Section 3.4 we provide some

basic definitions and formalize the black hole search problem. Finally, in Section 3.5 we show some preliminary lemmas we will recall in the following chapters.

3.2 The model

We represent a network as a connected undirected graph $G = (V, E)$, where nodes denote hosts and edges denote communication links. With no loss of generality, we assume G has no multiple edges or self-loops.
For the problem we consider, we assume that there are some nodes in G which destroy any agent visiting them without leaving any trace. We denote such nodes as BLACK HOLES, and the corresponding set as B. The remaining nodes are called SAFE NODES. The value of B, i.e. which nodes of the network are black holes, is (at least partially) unknown. During a Black Hole Search (or simply BHS), a set of agents starts from a special node s called the STARTING NODE, and explores graph G by traversing its edges and visiting its nodes. Obviously, the starting node s is known to be a safe node (i.e., it is in $V \setminus B$) since the beginning of the BHS; generally, a subset of nodes $S \subseteq V \setminus B$, initially known to be safe, is given as input. The target of the agents is to report to s the exact value of B, that is, which of the nodes in G are black holes.

For the problems considered in this thesis, we make the following further assumptions in the model.

1. $|B| \leq 1$, i.e., the network may contain either one black hole ($B = \{b\}$) or no black holes at all ($B = \emptyset$).
2. Only two agents perform the task; as explained in Section 2.2, this means that we are interested in algorithms with *optimal size*.
3. Agents have a complete map of G, and the nodes of the network are uniquely labeled; therefore, the agents always know their exact location in the network.
4. Agents have distinct labels (we will call them *Agent*-1 and *Agent*-2), and thus it is possible to assign them two different strategies.
5. Agents can communicate only when they are in the same node (and not, e.g., by leaving messages at nodes).
6. The system is synchronous or partially synchronous (according to the classification given in Section 2.1). In both cases we can assume that each step requires one time unit (possibly after normalizing the upper bound on the step length). In the following the terms “step” and “time unit” will be considered as equivalent.

The cost of a black hole search should be distinguished from the time complexity of an algorithm producing the scheme for the search. Informally, the former is the time of walking, while the latter is the time of preparing (planning) the walk. Follow-

ing [11] and [12], we study the optimization problem of computing (preparing), for a given network G and the starting node s, a minimum-cost black hole search scheme.

3.3 Other works on black hole search

The original formulation of the black hole problem was given for the first time in [15], and then the problem was further studied in [14, 16, 18, 19]. In these works, the model considered is slightly different than ours. First, the system is assumed to be totally asynchronous (i.e., no upper bound on the length of each step). The agents can communicate by writing and reading on whiteboards in nodes, and must be provided with the same protocol; the surviving agents are not required to return to the starting point. Within these assumptions, the authors prove that, in order to be able to solve the problem, at least two agents are needed, the network must be at least 2-connected. The main measures taken into account in these works are the *size*, i.e., the number of agents needed, and the *cost*, i.e. the total number of migrations required to complete the search.
In [18] the black hole search in anonymous ring networks is considered. It is shown that, if the size of the ring is not known, then no deterministic protocol exists which always correctly terminates. The authors then consider the exploration performed by two agents when they start from the same node (*co-located*) and when they start from randomly different nodes (*dispersed*). It is shown that in both cases $\Theta(n \log n)$ traversals are necessary and sufficient (by providing suited algorithms). The authors also consider efficient algorithms with respect to the ideal time, establishing a general trade-off between time and number of agents, and proving that at least $2n - 4$ ideal time units are always needed. Finally, the authors show that, if the ring is undirected, then in the dispersed case, at least three agents are necessary and sufficient.
In [16], the problem in arbitrary networks is studied. The authors establish tight bounds on size and cost, depending on the a priori knowledge on the map of the network. If the network is totally unknown, then the presented optimal algorithm has size $\Delta+1$ (Δ is the degree of the graph) and cost $\Theta(n^2)$. If the agents are not provided with the map of the graph, but have sense of direction[1], only two agents suffice and the cost is $\Theta(n^2)$. Finally, if complete topological knowledge in the graph is provided to agents, then two agents can find a black hole in $\Theta(n \log n)$ moves. In [19] and in [14] better upper bounds are provided for networks with particular topologies. In particular, if the network has diameter d, then $O(n + d \log d)$ moves are sufficient; if the network is a hypercube, a cube-connected cycle, a star, a wrapped butterfly, a chordal ring, a multi-dimensional mesh or a torus, then $O(n)$ moves are sufficient.

In [11, 12] the problem is studied under the model we consider in this thesis. The network is partially synchronous, i.e., there is an upper bound on the time needed

[1] A network is said to have a sense of direction if a consistent edge-labeling with directions in the graph is provided.

by an agent for traversing any edge. The partially synchronous network makes a dramatic change to the problem. The black hole can be located by two agents in any graph. Moreover the agents can decide if there is a black hole or not. To measure the efficiency of a black hole search, it is assumed that it takes exactly one time unit (one synchronized step) for each agent to traverse one edge (and to make all necessary computations associated with this move). Then the cost of a given black hole search (scheme) in a given network G and from a given starting node s is defined as the total time the search takes under the worst-case location of the black hole (or when there is no black hole in the network).

In [12] the Black Hole Search problem is studied in tree topologies, and the main results given are an exact polynomial-time algorithm for some sub-class of trees and a $5/3$-approximation algorithm for arbitrary trees. The existence of an exact polynomial-time algorithm for arbitrary trees is left open. The authors of [11] study a variant of the problem in which the input instance is a triple (G, S, s), where G and s are, as above, a network and the starting node, and $S \supseteq \{s\}$ is a given subset of nodes known to be safe (no black hole can be located in any node in S). The main results presented in [11] are that for arbitrary graphs this variant of the Black Hole Search problem is **NP**-hard but can be approximated within a ratio bound 9.3.

3.4 Basic definitions

The set of assumptions given in Section 3.2 allows us to give a precise formalization of the problem. We can observe that Assumption 3 on the knowledge of the network topology, reduces the task into a planning of a network exploration strategy that must be provided to the agents during their creation phase. Indeed, the facts that each step has a bounded length (Assumption 6) and that there is at most one black hole in the network (Assumption 1), confine the uncertainty only on the location of the black hole, and on which of the agents will find it. For these reasons we define the concept of exploration scheme. Given a triple (G, S, s), where $G = (V, E)$ is a connected undirected graph, $S \subsetneq V$ is a subset of nodes and $s \in S$, an EXPLORATION SCHEME $\mathcal{E}_{G,S,s} = (\mathbb{X}, \mathbb{Y})$ for (G, S, s), is a pair of equal-length sequences of nodes in G: $\mathbb{X} = \langle x_0, x_1, \ldots, x_T \rangle$ and $\mathbb{Y} = \langle y_0, y_1, \ldots, y_T \rangle$.

When the BHS based on a given exploration scheme $\mathcal{E}_{G,S,s}$ is performed in G, *Agent*-1 follows the path defined by $\mathbb{X}$ while *Agent*-2 follows the path defined by $\mathbb{Y}$. In other words, at the end of the i-th step of the BHS (at time i), *Agent*-1 is in node x_i, while *Agent*-2 is in node y_i. As soon as an agent deduces the value of B, it "aborts" the exploration and returns to the starting node s by traversing nodes in $V \setminus B$.
Note that in our model we do not account for the time of computing the shortest path that the surviving agents have to follow to return to s at the end of the exploration. We assume that either this time is negligible or the whole set of required shortest paths is precomputed and stored in the agents' memory.

It should be clear to the reader that not any pair of sequences can be effectively used as a basis for a BHS. We thus need to define which are the properties of a feasible exploration scheme. If $\mathbb{X} = \langle x_0, x_1, \ldots, x_T \rangle$ and $\mathbb{Y} = \langle y_0, y_1, \ldots, y_T \rangle$ are two equal-length sequences of nodes in G, then $\mathcal{E}_{G,S,s} = (\mathbb{X}, \mathbb{Y})$ is a FEASIBLE EXPLORATION SCHEME if the constraints 1–4 stated below are satisfied.

Constraint 1: $x_0 = y_0 = s$, $x_T = y_T$.
This corresponds to the fact that both agents start from the given starting node s. The requirement that the sequences $\mathbb{X}$ and $\mathbb{Y}$ end at the same node provides a convenient simplification of the reasoning without loss of generality.

Constraint 2: for each $i = 0, \ldots, T-1$, either $x_{i+1} = x_i$, or $(x_i, x_{i+1}) \in E$; and similarly either $y_{i+1} = y_i$ or $(y_i, y_{i+1}) \in E$.
This constraint models the fact that during each step, an agent can either WAIT in the node v where it was at the end of the previous step, or migrate by traversing an edge of the network to move to a node adjacent to v.
Note that the fact that an agent can wait in a node, requires us to slightly bend the strict definition of step as "migration+execution" we gave in Section 2.1. In our synchronized model, we can redefine a step as a fixed slot of time during which it is guaranteed that an agent *can* reach a neighbor host by traversing an edge and perform all the needed computations in such host.

Constraint 3: $S \cup \bigcup_{i=0}^{T} \{x_i\} \cup \bigcup_{i=0}^{T} \{y_i\} = V$.
This is to assure that each node in $V \setminus S$ (the set of nodes whose safety is initially unknown) is visited by at least one agent during the exploration.

In order to state Constraint 4, some preliminary definitions are required.
Given an exploration scheme $\mathcal{E}_{G,S,s} = (\mathbb{X}, \mathbb{Y})$, for each $i = 0, 1, \ldots, T$, we call the EXPLORED TERRITORY at step i the set S_i defined in the following way:

$$S_i = \begin{cases} S \cup \bigcup_{j=0}^{i} \{x_j\} \cup \bigcup_{j=0}^{i} \{y_j\}, & \text{if } x_i = y_i; \\ S_{i-1}, & \text{otherwise.} \end{cases}$$

Thus $S_0 = S$ by Constraint 1, $S_T = V$ by Constraint 1 and Constraint 3, and $S_{j-1} \subseteq S_j$ for each step $1 \leq j \leq T$. A node v is EXPLORED at a step i if $v \in S_i$, or UNEXPLORED otherwise. These definitions reflect the assumption that the agents communicate with each other, exchanging their full knowledge, only when they meet at a node. An unexplored node v may have been already visited by one of the agents, but it will become explored only when the agents meet (and communicate) next time. If both agents are alive at the end of step i, then the explored nodes at this step are all nodes which are known by *both* agents to be safe.

A MEETING STEP (or simply MEETING) is the step 0 and every step $1 \leq j \leq T$ such that $S_j \neq S_{j-1}$. Observe that, in each meeting step j, the agents must be

in the same node ($x_j = y_j$); we call this node a MEETING POINT. Note that the opposite is not necessarily true, i.e., there can exist non-meeting steps during which the agents are in the same node. The meeting steps are the steps when the agents meet and add at least one new node to the explored territory. A sequence of steps $\langle j+1, j+2, \ldots, k\rangle$ where j and k are two consecutive meetings is called a PHASE of length $k-j$; the length of a phase is the number of steps between the two meetings, that is $k-j$. Observe that an agent can discover the location of the black hole, and thus can abort the exploration and return to s, only at the meeting steps, i.e., at the end of a phase.

We recall from [12] the following lemma, showing two basic properties that every feasible exploration must satisfy.

Lemma 3.4.1 *During a phase of a feasible exploration scheme, an agent can visit at most one unexplored node and the same unexplored node cannot be visited by both agents.*

Proof. Suppose that an agent visits two or more unexplored nodes during the same phase. If one of the nodes is a black hole and hence the agent vanishes, then there is no way for the other agent to locate the black hole without vanishing.
By the fact that an agent can abort the exploration only at the end of a phase, if the two agents are scheduled to visit the black hole during the same phase, then both will vanish in it, thus not accomplishing the task. □

We enforce the property expressed in Lemma 3.4.1, by mean of a fourth feasibility constraint for an exploration scheme.

Constraint 4: for each phase with a sequence of steps $\langle j+1, \ldots, k\rangle$,

(a) $|\{x_{j+1}, \ldots, x_k\} \setminus S_j| \leq 1$ and $|\{y_{j+1}, \ldots, y_k\} \setminus S_j| \leq 1$; and
(b) $\{x_{j+1}, \ldots, x_k\} \setminus S_j \neq \{y_{j+1}, \ldots, y_k\} \setminus S_j$.

The (GENERAL) MINIMUM COST BHS PROBLEM, or simply the BHS problem, can be thus formalized in the following way.

Minimum Cost BHS Problem (BHS problem)

Instance : a triple (G, S, s), where $G = (V, E)$ is a connected undirected graph, $S \subsetneq V$ is a subset of nodes and $s \in S$.

Solution : a feasible EXPLORATION SCHEME $\mathcal{E}_{G,S,s} = (\mathbb{X}, \mathbb{Y})$ for (G, S, s). The feasibility of $\mathcal{E}_{G,S,s}$ is determined by Constraints 1–4 given before. The length of the exploration scheme $\mathcal{E}_{G,S,s}$ is defined to be T.

Measure : *cost*($\mathcal{E}_{G,S,s}$), i.e., the cost of the BHS based on $\mathcal{E}_{G,S,s}$.

Goal : minimization.

In order to fully delineate the problem, we have to define our notion of cost of a BHS. We first provide a preliminary definition. For an exploration scheme $\mathcal{E}_{G,S,s} = (\mathbb{X}, \mathbb{Y})$ and a fixed value of B, where either $B = \emptyset$ or $B = \{b\}$ for $b \in (V \setminus S)$, the EXECUTION TIME for $(\mathcal{E}_{G,S,s}, B)$ is defined as the number of time units taken by the agents following $\mathcal{E}_{G,S,s}$ to discover the value of B and to report it back to s as fast as possible. If $B = \emptyset$, then the execution time is equal to the length T of the exploration scheme, plus the shortest path distance from $x_T (= y_T)$ to s. In this case, in fact, the agents must perform the full exploration (spending one time unit per step) and then get back to the starting node to report that there is no black hole in the network. If $B = \{b\}$, then let j be the first step in $\mathcal{E}_{G,S,s}$ such that $b \in S_j$. Observe that j must be a meeting step and $1 \leq j \leq T$ since $S_0 = S$ and $S_T = V$. The execution time in this case is equal to j plus the shortest length of a path from $x_j (= y_j)$ to s not including b. In this case one agent, say *Agent*-1, vanishes into the black hole during the phase ending at step j, so it does not show up to meet *Agent*-2 at node $x_j = y_j$. Since, by Constraint 4, *Agent*-1 has visited only one unexplored node during the phase, the surviving *Agent*-2 learns the exact location of the black hole and thus it goes back to s, obviously omitting the black hole.

The COST of the BHS based on an exploration scheme $\mathcal{E}_{G,S,s} = (\mathbb{X}, \mathbb{Y})$ (or simply the cost of the exploration scheme) is denoted by *cost*$(\mathcal{E}_{G,S,s})$ and defined as the worst (maximum) execution time for $\mathcal{E}_{G,S,s}$, over all the possible values of B. In other words, in computing the cost of a BHS, we allow a malicious adversary, which exactly knows $\mathcal{E}_{G,S,s}$, to place the black hole (or not to place it at all) in such a way that the BHS requires as many time units as possible.
In the following, feasibility of exploration schemes will be implicit, and by exploration scheme we will always mean a feasible exploration scheme.

3.5 Basic properties

In this section we present some simple observations about exploration schemes, which will be frequently used in the following chapters.

Lemma 3.5.1 *If step $k \geq 1$ is a meeting step for an exploration scheme $\mathcal{E}_{G,S,s}$, then $x_k = y_k \in S_{k-1}$.*

Proof. Let j be the last meeting step before step k, and hence $S_j = S_{j+1} = \ldots = S_{k-1}$. By definition $x_k = y_k \in S_k$. If $x_k = y_k$ is not in S_{k-1}, then it is in both $\{x_{j+1}, \ldots, x_k\} \setminus S_j$ and $\{y_{j+1}, \ldots, y_k\} \setminus S_j$. In this case, at least one of the conditions of Constraint 4 is violated, since either the two sets are the same or at least one of the two contains more than one node. □

Lemma 3.5.2 *Each phase of an exploration scheme $\mathcal{E}_{G,S,s}$ has length at least 2.*

Proof. Let us suppose, by contradiction, that there exists in $\mathcal{E}_{G,S,s}$ a phase of length 1, and hence two adjacent meeting steps j and $j+1$. The step $j+1$ is a meeting if and only if $S_{j+1} \supsetneq S_j$, but, by Lemma 3.5.1, $x_{j+1} = y_{j+1} \in S_j$, and hence $S_{j+1} = S_j$. Therefore there cannot exist in $\mathcal{E}_{G,S,s}$ a phase of length 1. □

As a trivial consequence of Constraint 4, at the end of each phase, the explored territory may increase by at most 2 nodes. In view of Lemma 3.5.2, phases of length 2 which expand the explored territory by 2 nodes are of particular interest to us since they progress the exploration of the network at the fastest possible speed. Any phase $\langle j+1, j+2\rangle$ of this kind has to have the following structure. Let m be the meeting point at step j. During step $j+1$, *Agent*-1 visits an unexplored node v_1 adjacent to m, while *Agent*-2 visits an unexplored node v_2 adjacent to m as well, and $v_1 \neq v_2$. In step $j+2$, the agents meet in a node which has been already explored and is adjacent to both v_1 and v_2. This node can be either m, and in this case we denote the phase as **b-split**(m, v_1, v_2), or a different node $m' \neq m$, and in this case we denote the phase as **a-split**(m, v_1, v_2, m').

In what follows we introduce an operation aimed to modify an exploration scheme without altering its properties (i.e., feasibility, length, sequence of explored territories and the cost of the BHS based on it). We then define a notion of equivalence between exploration schemes which is based on such operation.

Definition 3.5.1 *Let $\mathcal{E}_{G,S,s} = (\mathbb{X}, \mathbb{Y})$ be an exploration scheme for (G, S, s), and let $\phi = (\mathbb{X}_\phi, \mathbb{Y}_\phi)$ be a phase in $\mathcal{E}_{G,S,s}$. Let $\mathcal{E}'_{G,S,s}$ be the exploration scheme obtained from $\mathcal{E}_{G,S,s}$ by swapping the paths of the two agents in phase ϕ, i.e., $\phi' = (\mathbb{Y}_\phi, \mathbb{X}_\phi)$. We call this operation a* PHASE-SWAP. *Two exploration schemes are* EQUIVALENT *if and only if one is obtained from the other by applying a finite sequence of phase-swaps.*

Lemma 3.5.3 *Let $\mathcal{E}_{G,S,s} = (\mathbb{X}, \mathbb{Y})$ be an exploration scheme for (G, S, s). Let $\mathcal{E}'_{G,S,s}$ be the exploration scheme obtained from $\mathcal{E}_{G,S,s}$ by applying a phase swap on $\mathcal{E}_{G,S,s}$. Then, the exploration scheme $\mathcal{E}'_{G,S,s}$ is feasible, has exactly the same meeting points, the same sequence of explored territories and the same length as $\mathcal{E}_{G,S,s}$. Moreover,* $\text{cost}(\mathcal{E}'_{G,S,s}) = \text{cost}(\mathcal{E}_{G,S,s})$.

Proof. Obviously nothing changes for the unaltered phases. Moreover, for what concerns the swapped phase, neither the meeting point, nor the length are subject to modification. Also the nodes added to the explored territory are the same, and hence the execution time for the cases in which B is one of such nodes, remains unvaried (even if the surviving agent changes). □

Corollary 3.5.4 *Two equivalent exploration schemes have exactly the same meeting points, the same sequence of explored territories and the same length. Moreover the cost of the BHS based on them is the same.*

Observe that, by the definition of cost of a BHS given in Section 3.4, the computation of the cost of a given exploration scheme requires to evaluate the execution time for all the possible values of B. The following lemma helps to simplify, at least in some cases, this task.

Lemma 3.5.5 *Let (G, S, s) be an input instance for the* BHS problem*, and let U be the set of initially unexplored nodes ($U = V \setminus S$). The case $B = \emptyset$ yields the maximum execution time for any exploration scheme in (G, S, s) over all the possible values of $B \in U$, if and only if, by removing any node $u \in U$ from G, each node in $V \setminus \{u\}$ either becomes disconnected from s, or maintains its shortest path distance from s.*

Proof. *(if).* Let us consider any exploration scheme $\mathcal{E}_{G,S,s}$ and the case $B = \{b\} \neq \emptyset$ (for any b). By hypothesis, we can remove b from G and have a partition of the nodes in two subsets: nodes becoming disconnected from s, and nodes maintaining the distance from s. Let m be the meeting point at the end of the phase ϕ of $\mathcal{E}_{G,S,s}$ during which b is visited for the first time. Node b cannot disconnect m from s (by Lemma 3.5.1 node m must be explored before ϕ and hence there must exist a path from s to m not containing b). Hence m maintains its distance from s even if we remove b from G. Therefore, the path from m to s the agents follow in the case $B = \emptyset$ cannot be shorter than the shortest path from m to s the surviving agent can follow in the case $B = \{b\}$.

(only if). Let us suppose there exists a node $u \in U$ such that its deletion from G increases the distance from a node $v \in V$ to s, say, by δ. Let $\mathcal{E}_{G,S,s}$ be an exploration scheme such that during the last phase node u is explored by one agent and the meeting point is node v. (It is easy to check that if node u disconnects v from s, then such scheme does not exist.) In the case $B = \emptyset$, the execution time is equal to the length of $\mathcal{E}_{G,S,s}$ plus the distance in V from v to s (say M), while in the case $B = \{u\}$ the execution time is equal to the length of $\mathcal{E}_{G,S,s}$ plus the distance in $V \setminus \{u\}$ from v to s, that is $M + \delta$. □

Corollary 3.5.6 *Let (G, S, s) be an input instance for the* BHS problem*. If G is a tree rooted at s, then the case $B = \emptyset$ yields the maximum execution time for any exploration scheme in (G, S, s).*

Proof. This assertion straightforwardly follows from the property that in any tree there is always a unique path from any node to the root.

Chapter 4

The General BHS Problem

This chapter is devoted to present the main approximability results related to the BHS problem. Here we consider the same model as in [11], where the authors prove the **NP**-hardness of the problem and present a 9.3-approximation algorithm, i.e., a polynomial time algorithm which, for any input instance (G, S, s), produces an exploration scheme whose cost is at most 9.3 times the best cost of an exploration scheme for such input. In Section 4.1 we show that the BHS problem is not approximable in polynomial time within a $1 + \epsilon$ factor, for any $\epsilon < \frac{1}{388}$, unless **P** $=$ **NP**. In Section 4.2 we give a 6-approximation algorithm for the problem. A preliminary version of these results can be found in [28].

4.1 Approximation lower bound

In this section we provide an explicit lower bound on the approximability of the BHS problem by showing an approximation preserving reduction from a particular subcase of the Traveling Salesman Problem, presented in [21], and defined in the following way.

(1,M)-Traveling Salesman Problem (TSP(1,M))

Instance : a pair (G, d), where $G = (V, E)$ is a complete graph (with $n = |V|$) and $d : V^2 \rightarrow \{1, \ldots, M\}$ is a distance function associating to each pair of nodes (v, u) a positive integer distance $d(v, u)$ between 1 and M (where M is a constant). Function d is symmetric (i.e., $d(u, v) = d(v, u)$) and satisfies the triangle inequality (i.e., $d(i, j) + d(j, k) \geq d(i, k)$, $\forall i, j, k \in V$).

Solution : a tour τ of G, i.e., a permutation $\tau = \langle v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)} \rangle$ of the nodes in V.

Measure : the *length* (or *cost*) of the tour, i.e.,

$$cost(\tau) = \sum_{i=1}^{n-1} d(v_{\pi(i)}, v_{\pi(i+1)}) + d(v_{\pi(n)}, v_{\pi(1)}).$$

Goal : minimization.

In the same paper, the authors also present the following lower bound on the approximability of such problem.

Lemma 4.1.1 *It is **NP**-hard to approximate* TSP(1,8) *within* $1 + \varepsilon$ *for any* $\varepsilon < \frac{1}{388}$.

Our approach to prove the **APX**-hardness of the BHS problemis the following. We first provide a reduction from instances (G, d) of TSP(1,M) to instances (G', S, s) of the BHS problem. Then we show how to construct a particular solution of the BHS problem (a τ*-based* exploration scheme $\mathcal{E}^{\tau}_{G',S,s}$), based on a corresponding solution of the first problem (a tour τ in G). We show (Lemma 4.1.4) that $cost(\mathcal{E}^{\tau}_{G',S,s}) = 2cost(\tau)$. Then, by introducing the concept of *regular* exploration schemes, we show that given any exploration scheme $\mathcal{E}_{G',S,s}$, we can find a tour τin G such that $cost(\mathcal{E}^{\tau}_{G',S,s}) \leq cost(\mathcal{E}_{G',S,s})$ (Lemma 4.1.8 and Lemma 4.1.9). Finally (Lemma 4.1.10), we show that if, for any instance of BHS problem constructed by reduction from an instance of TSP(1,M), we can approximate the optimal solution within a $(1 + \epsilon)$ factor, then we can approximate the optimal solution of the corresponding instance of TSP(1,M) within the same factor.

Polynomial time reduction from (G, d) **to** (G', S, s). Let (G, d) be an instance of TSP(1,M). We define the graph $G' = (V', E')$, the set S, and the starting node s, in the following way. Let v_1 be an arbitrary node in V. We add v_1 to V' and to S, and we define $s = v_1$ as the starting point of the BHS. For each node v_i ($2 \leq i \leq n$) in V, we add to V' a pair of nodes v'_i, v''_i. We denote node v_1 as the ISLAND I_1, and each pair of nodes v'_j, v''_j (with $j = 2, \ldots, n$) as the ISLAND I_j. Then we connect islands: for each edge (v_i, v_j) in E of length $d(v_i, v_j)$, we add to V' (and to E') a path of $2 \cdot d(v_i, v_j) - 1$ nodes, whose endpoints are adjacent respectively to v'_i, v''_i (or v_1 if $i = 1$) and to v'_j, v''_j (or v_1 if $j = 1$). We denote such path connecting island I_i with island I_j as BRIDGE $i \leftrightarrow j$. We add all the nodes of the bridge to S. We call as $b_{i,j}$ and as $b_{j,i}$ the endpoints of bridge $i \leftrightarrow j$ adjacent respectively to island I_i and island I_j (note that if $d(v_i, v_j) = 1$, then $b_{i,j} \equiv b_{j,i}$). Observe that each bridge is composed of at least one (safe) node, and that $|V' \setminus S| = 2(n - 1)$. An example of this reduction is presented in Figure 4.1.

Lemma 4.1.2 *The distance in G' between any node of island I_i and any node of island I_j (where $i \neq j$ and $i, j = 1, \ldots, n$) is equal to* $2 \cdot d(v_i, v_j)$.

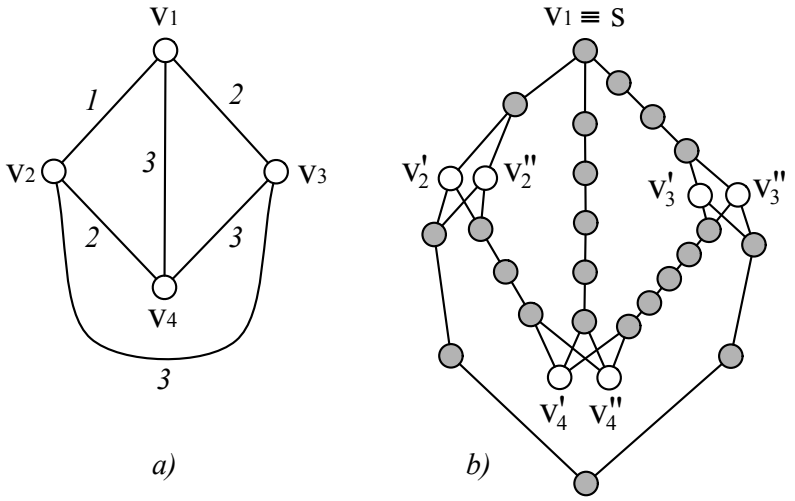

Figure 4.1: In b) an instance of the BHS problem obtained by reduction from an instance of the TSP(1,M) problem (in a)). The nodes in S are filled with gray color.

Proof. By construction, bridge $i \leftrightarrow j$ is composed of $2 \cdot d(v_i, v_j) - 1$ nodes. Hence the length of the path from I_i to I_j which uses such bridge is $2 \cdot d(v_i, v_j)$. Suppose, by contradiction, that there exists a path in G' from I_i to I_j of length less than $2 \cdot d(v_i, v_j)$. This path starts from I_i, visits some other islands (say $\langle I_{k_1}, \ldots, I_{k_k} \rangle$) and then ends in I_j. The length of such path is $2\,[d(v_i, v_{k_1}) + d(v_{k_1}, v_{k_2}) + \cdots + d(v_{k_k}, v_j)]$. This would mean that $d(v_i, v_{k_1}) + d(v_{k_1}, v_{k_2}) + \cdots + d(v_{k_k}, v_j) < d(v_i, v_j)$. By recursive application of triangle inequality on the distances in G, this is not possible. Contradiction. □

A relevant property we can observe is that assumptions of Lemma 3.5.5 hold for graph G'.

Lemma 4.1.3 *Let (G', S, s) be an instance of* BHS problem *produced with the above mentioned reduction. The case $B = \emptyset$ yields the maximum execution time for any exploration scheme in G'.*

Proof. Recall that the only unexplored nodes are the nodes in islands I_j (for $j = 2, \ldots, n$). Let v'_i be any node in U. By removing v'_i from G', no node becomes disconnected from s. Moreover, the node v''_i (the other unexplored node in the same island), is at the same distance as v'_i from s, and has exactly the same set of neighbors as v'_i. Therefore, each node in G' which has v'_i in his shortest path to s, can replace v'_i with v''_i in the path and remain at the same distance from s. By Lemma 3.5.5 the assertion is proved. □

The reduction presented before defines a relationship between instances of TSP(1,M) and corresponding instances of BHS problem. Now we want to investigate a similar relation between solutions of TSP(1,M) and corresponding solutions of the BHS problem. Given an instance (G, d) of TSP(1,M) and the corresponding instance (G', S, s) of BHS problem, and given a tour τ in G, we define an exploration scheme on G' which explores the islands in G', in the order defined by τ. In the following definition we introduce the new keyword **walk**. By **walk**(b) we mean that both agents (which are supposed to be already in the same node w), move to b by following a shortest (safe) path from w to b. Actually, the walk is not a complete phase (no new nodes are explored), but it is always the initial part of a phase.

Let $\tau = \langle v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)} \rangle$ be a tour on G of length l. We assume with no loss of generality that $\pi(1) = 1$. A τ-BASED EXPLORATION SCHEME $\mathcal{E}^{\tau}_{G',S,s}$ on G' consists of the following sequence of steps:

1. **walk**($b_{1,\pi(2)}$), where $b_{1,\pi(2)}$ is the node adjacent to s on the bridge $1 \leftrightarrow \pi(2)$;
2. for each $i = 2, \ldots, n$:
   (a) **walk**($b_{\pi(i),\pi(i-1)}$), where $b_{\pi(i),\pi(i-1)}$ is the node adjacent to $I_{\pi(i)}$ on the bridge $\pi(i-1) \leftrightarrow \pi(i)$;
   (b) **a-split**($b_{\pi(i),\pi(i-1)}, v'_{\pi(i)}, v''_{\pi(i)}, b_{\pi(i),\pi(i+1)}$), where $b_{\pi(i),\pi(i+1)}$ is the node adjacent to $I_{\pi(i)}$ on the bridge $\pi(i) \leftrightarrow \pi(i+1)$ (or bridge $\pi(n) \leftrightarrow 1$ if $i = n$).

In other words, the two agents walk together along the bridges, they separate to visit the two nodes of each unexplored island, and meet again on the first node of the next bridge.
Given the tour τ in G, the τ-based exploration scheme $\mathcal{E}^{\tau}_{G',S,s}$ can be obviously constructed in linear time.
In the following lemma we compute the cost of the BHS based on $\mathcal{E}^{\tau}_{G',S,s}$.

Lemma 4.1.4 *Given a tour $\tau = \langle v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)} \rangle$ on G of length l, the τ-based exploration scheme $\mathcal{E}^{\tau}_{G',S,s}$ satisfies* cost$(\mathcal{E}^{\tau}_{G',S,s}) = 2 \cdot l$.

Proof. By Lemma 4.1.3, we can compute *cost*$(\mathcal{E}^{\tau}_{G',S,s})$ as the execution time of $\mathcal{E}^{\tau}_{G',S,s}$ in the case $B = \emptyset$. The *walk* in (1) requires 1 step. For the i-th iteration in (2) $(i = 2, \ldots, n)$:

- the *walk* in $(2.a)$ requires $2 \cdot d(v_{\pi(i-1)}, v_{\pi(i)}) - 2$ steps;
- the phase defined in $(2.b)$ requires 2 steps;

The exploration scheme $\mathcal{E}^{\tau}_{G',S,s}$ ends in $b_{\pi(n),1}$, and hence the surviving agents have to get back to s. By Lemma 4.1.2, the distance from $b_{\pi(n),1}$ to s is $2 \cdot d(v_{\pi(n)}, v_1) - 1$. Therefore: *cost*$(\mathcal{E}^{\tau}_{G',S,s}) = 1 + 2 \sum_{i=2}^{n} d(v_{\pi(i-1)}, v_{\pi(i)}) + 2 \cdot d(v_{\pi(n)}, v_1) - 1 = 2 \cdot l$. □

Corollary 4.1.5 *Let (G, d) be an instance of the* TSP(1,M) *problem, and let (G', S, s) be the corresponding instance of the* BHS problem*, where the graph G' is constructed as explained before. Moreover, let τ^* be an optimal solution for (G, d) and let $\mathcal{E}^*_{G',S,s}$ be an optimal solution for (G', S, s). Then* $\text{cost}(\mathcal{E}^*_{G',S,s}) \leq 2 \cdot \text{cost}(\tau^*)$.

We can classify each phase of any exploration scheme in G', according to the number and the location of the nodes explored during such phase. A phase ϕ is a:

2s-phase : if the two nodes of the same island are explored during ϕ;

2d-phase : if two nodes in two distinct islands are explored during ϕ;

1-phase : if only one node is explored during ϕ.

Definition 4.1.1 *Given an exploration scheme $\mathcal{E}_{G',S,s}$, we define the* PHASE GRAPH *of $\mathcal{E}_{G',S,s}$, the following directed multigraph $P(\mathcal{E}_{G',S,s})$. The graph $P(\mathcal{E}_{G',S,s})$ has the nodes $v_2, \ldots, v_n$ corresponding to the islands $I_2, \ldots, I_n$ in G', plus one further node which we call x. The following edges are added to $P(\mathcal{E}_{G',S,s})$:*

- *a directed edge $\langle v_i, x\rangle$ ($\langle x, v_i\rangle$) is added for each node in island I_i which is explored during a 1-phase by* Agent-1 *(*Agent-2*);*
- *a directed edge $\langle v_i, v_j\rangle$ is added for each 2d-phase exploring a node of island I_i with* Agent-1 *and a node of island I_j with* Agent-2*;*
- *a directed self-loop $\langle v_i, v_i\rangle$ is added if the nodes of island I_i are explored by a 2s-phase.*

Lemma 4.1.6 *Given any exploration scheme $\mathcal{E}_{G',S,s}$, each node of the phase graph $P(\mathcal{E}_{G',S,s})$ has degree (= in-degree + out-degree) equal to* 2*.*

Proof. It follows from Definition 4.1.1 that, for any node v_i in $P(\mathcal{E}_{G',S,s})$, there is an outgoing edge for each node in I_i of G' which is explored by *Agent*-1, and there is an incoming edge for each node in I_i of G' which is explored by *Agent*-2. Since each island I_i ($i = 2, \ldots, n$) has two unexplored nodes, the statement follows. □

The graph $P(\mathcal{E}_{G',S,s})$ is thus a set of connected components. In the underlying undirected multigraph, these components are either cycles or isolated nodes.
Now we give a new characterization of an exploration scheme in G'.

Definition 4.1.2 *An exploration scheme $\mathcal{E}_{G',S,s}$ is* REGULAR *if and only if each agent explores exactly one node of each island I_j, with $j = 2, \ldots, n$.*

Notice that any τ-based exploration scheme is regular; we can observe that each node in $P(\mathcal{E}^{\tau}_{G',S,s})$ is an isolated node (the only adjacent edge is a self-loop).
Indeed, we can prove a tighter relation between regular exploration schemes and their corresponding phase graph.

Lemma 4.1.7 *An exploration scheme $\mathcal{E}_{G',S,s}$ is regular if and only if, in the corresponding phase graph $P(\mathcal{E}_{G',S,s})$, for each node v_i, $indeg(v_i) = 1$ and $outdeg(v_i) = 1$.*

Proof. By Lemma 4.1.6, each node v_i in $P(\mathcal{E}_{G',S,s})$ has degree 2. Hence, three cases may happen:

1. $indeg(v_i) = 1$ and $outdeg(v_i) = 1$: in this case one node of island I_i is explored by *Agent*-1 (the outgoing edge) and the other one is explored by *Agent*-2 (the incoming edge). Therefore the island is "regularly" explored by both agents.

2. $indeg(v_i) = 0$ and $outdeg(v_i) = 2$: in this case both nodes of I_i are explored by *Agent*-1; the island is not "regularly" explored.

3. $indeg(v_i) = 2$ and $outdeg(v_i) = 0$: in this case both nodes of I_i are explored by *Agent*-2; the island is not "regularly" explored.

□

Lemma 4.1.8 *For any exploration scheme $\mathcal{E}_{G',S,s}$ there exists an equivalent regular one that can be found in linear time.*

Proof. It suffices to prove that we can find in linear time a finite sequence of phase-swaps in $\mathcal{E}_{G',S,s}$, which transforms $\mathcal{E}_{G',S,s}$ into a regular exploration scheme. By Lemma 4.1.7, this corresponds to transform $P(\mathcal{E}_{G',S,s})$ into a graph where, for each node v_i, $indeg(v_i) = 1$ and $outdeg(v_i) = 1$. We can observe that each phase-swap in $\mathcal{E}_{G',S,s}$ produces a change in the orientation of the corresponding edge in $P(\mathcal{E}_{G',S,s})$. Since $P(\mathcal{E}_{G',S,s})$ is composed by a set of cycles and isolated nodes, we can swap the edges in the cycles according to a fixed orientation (e.g., by a lexicographical ordering of nodes), and thus make regular the graph $P(\mathcal{E}_{G',S,s})$, and the corresponding exploration scheme. □

Lemma 4.1.9 *Given an exploration scheme $\mathcal{E}_{G',S,s}$, we can find in linear time a tour τ on G such that, for the τ-based exploration scheme $\mathcal{E}^{\tau}_{G',S,s}$, $\text{cost}(\mathcal{E}^{\tau}_{G',S,s}) \leq \text{cost}(\mathcal{E}_{G',S,s})$.*

Proof. By Corollary 3.5.4 and Lemma 4.1.8, we can assume without loss of generality that $\mathcal{E}_{G',S,s}$ is a regular exploration scheme. By regularity, *Agent*-1 explores a node of each island in G'. Let $I_{\mathbb{X}} = \langle I_{\pi(2)}, \ldots, I_{\pi(n)} \rangle$ be the sequence of the islands in G' in the order by which *Agent*-1 explore their nodes. Let τ be the tour in G corresponding to $I_{\mathbb{X}}$ (i.e., $\tau = \langle v_1, v_{\pi(2)}, \ldots, v_{\pi(n)} \rangle$), and let $l = cost(\tau)$. We show that the τ-based exploration scheme $\mathcal{E}^{\tau}_{G',S,s}$ is such that $cost(\mathcal{E}_{G',S,s}) \geq cost(\mathcal{E}^{\tau}_{G',S,s})$. Consider the case $B = \emptyset$ (Lemma 4.1.3). *Agent*-1 starts from s, visits islands in

$I_{\mathbb{X}}$ and then gets back to s. By Lemma 4.1.2, the length of this path is at least $2 \cdot l$, and hence the execution time of $\mathcal{E}_{G',S,s}$ is not shorter than $2 \cdot l$. Therefore, $cost(\mathcal{E}_{G',S,s}) \geq 2 \cdot l \geq cost(\mathcal{E}^{\tau}_{G',S,s})$. □

Now we can relate approximability of TSP(1,M) with approximability of BHS problem.

Lemma 4.1.10 *Let (G, d) be an instance of the* TSP(1,M) *problem, and let (G', S, s) be the corresponding instance of the* BHS problem *problem. Moreover, let τ^* be an optimal tour in G, and let $\mathcal{E}^*_{G',S,s}$ be an optimal exploration scheme for G'. Let $\varepsilon > 0$. If we can find in polynomial time an exploration scheme $\mathcal{E}_{G',S,s}$ such that* $\text{cost}(\mathcal{E}_{G',S,s}) \leq \text{cost}(\mathcal{E}^*_{G',S,s})(1+\varepsilon)$*, then we can find in polynomial time a tour τ in G such that* $\text{cost}(\tau) \leq \text{cost}(\tau^*)(1+\varepsilon)$.

Proof. Let us suppose that, given the instance (G', S, s), we can construct in polynomial time an exploration scheme $\mathcal{E}_{G',S,s}$ such that its cost is at most $1+\varepsilon$ times the cost of an optimal exploration scheme. By Lemma 4.1.9, we can find an exploration scheme $\mathcal{E}^{\tau}_{G',S,s}$, based on a tour τ in G, such that $cost(\mathcal{E}^{\tau}_{G',S,s}) \leq cost(\mathcal{E}_{G',S,s}) \leq cost(\mathcal{E}^*_{G',S,s})(1+\varepsilon)$. Supposing that the length of the tour τ is l, then, by Lemma 4.1.4: $cost(\mathcal{E}^{\tau}_{G',S,s}) = 2 \cdot l$. Supposing that the length of the optimal tour τ^* is l^*, then, by Corollary 4.1.5: $cost(\mathcal{E}^*_{G',S,s}) \leq 2 \cdot l^*$. Therefore, by hypothesis:

$$2 \cdot l \;=\; cost(\mathcal{E}^{\tau}_{G',S,s}) \;\leq\; cost(\mathcal{E}^*_{G',S,s})(1+\varepsilon) \;\leq\; 2 \cdot l^*(1+\varepsilon)\,.$$

Hence, $l \leq l^*(1+\varepsilon)$. □

The main theorem immediately follows from Lemma 4.1.1 and Lemma 4.1.10.

Theorem 4.1.11 *The* BHS problem *problem is not approximable in polynomial time within a factor of $1+\varepsilon$ for any $\varepsilon < \frac{1}{388}$, unless **P=NP**.*

4.2 A 6-approximation algorithm

Let $G = (V, E)$ be a network to be explored in search of a black hole, with a set S of safe nodes, and a starting point s. Let $u = |U|$ be the number of unexplored nodes in G. In this section we describe an algorithm to produce an exploration scheme $\mathcal{E}_{G,S,s}$, whose cost is at most 6 times the cost of an optimal scheme for (G, S, s). This result improves the 9.3-approximation algorithm presented in [11]. The algorithm presented in that paper follows a rather simple approach. First a minimum Steiner tree of G is computed, where $U \cup \{s\}$ is the set of terminal (required) nodes. Let T be such computed tree. Note that the Minimum Steiner Tree problem is **APX**-hard ([7]), hence, if we denote as y the number of Steiner nodes in T and as y^* the number of Steiner nodes in an optimal solution for the problem, then the best result we can

obtain by a polynomial time algorithm is bounded by: $u+y \leq \alpha(u+y^*)$, where α is a constant value larger than 1. The exploration scheme $\mathcal{E}_{G,S,s}$ is obtained from a depth-first traversal of the internal nodes of T. Each time an internal node is visited for the first time by the agents, all its unexplored children are explored by means of a *probe* phase. Assuming that both agents are in the same node p and that v is an unexplored node adjacent to p, by **probe**(v) we denote a phase during which *Agent*-1 traverses edge (p, v), visits v and returns to node p to meet the other agent waiting there. Moreover, we denote by **probe-and-visit**(v) a part of exploration scheme obtained by concatenating **probe**(v) with **walk**(v). Each *probe* phase requires 2 time units, while each *probe-and-visit* sequence requires 3 time units. The cost of $\mathcal{E}_{G,S,s}$ is thus upper bounded by $2(u+y)+2u \leq 4(u+y)$. The authors fix the approximation ratio by observing that any exploration scheme in G has cost at least $\frac{2}{3}(u+y^*)$. Therefore:

$$\textit{cost}(\mathcal{E}_{G,S,s}) \leq 4(u+y) \leq \alpha \cdot 4(u+y^*) \leq \alpha \cdot 6 \cdot \textit{cost}(\mathcal{E}^*_{G,S,s})$$

If we use the best known approximating algorithm for the Minimum Steiner Tree problem ([35]), then $\alpha = \frac{\ln 3}{2} < 1.55$, and hence $\textit{cost}(\mathcal{E}_{G,S,s}) < 9.3 \cdot \textit{cost}(\mathcal{E}^*_{G,S,s})$.

Although we use a similar tree based approach for the exploration, the tree constructed by our algorithm has a different minimization target from the one described before: instead of (globally) minimizing the number of Steiner (initially safe) nodes to traverse, we want to minimize the relative distances of the terminal (unexplored) nodes the agents have to visit.

We define the following complete weighted graph $\widehat{G}$. The set of nodes in $\widehat{G}$ corresponds to the nodes in $U \cup \{s\}$. The weight of edge (v_i, v_j) (or, equivalently, the distance from v_i to v_j) in $\widehat{G}$, is the shortest path distance from v_i to v_j in G (considering both safe and unexplored nodes). An example of $\widehat{G}$ is given in Figure 4.2. Note that since weights in $\widehat{G}$ are derived from a shortest path metric, they satisfy triangle inequality. Let T be a minimum spanning tree of $\widehat{G}$ rooted at s, and let $\textit{cost}(T)$ be its cost, i.e., the sum of the weights of all its edges. Let $L_T = \langle v_{\pi(0)} \equiv s, v_{\pi(1)}, \ldots, v_{\pi(u)} \rangle$ be the depth-first ordering of the nodes in T, and let L_G be the sequence obtained from L_T by replacing each pair of adjacent nodes $v_{\pi(i)}, v_{\pi(i+1)}$ with the shortest path in G from $v_{\pi(i)}$ to $v_{\pi(i+1)}$. Since the distance from $v_{\pi(i)}$ to $v_{\pi(i+1)}$ is at most the (weighted) cost of path $v_{\pi(i)}, \ldots, v_{\pi(i+1)}$ in T, the length of L_G is at most $2\textit{cost}(T) - d(v_{\pi(u)}, s)$. An example of the construction of $\widehat{G}$ and the corresponding sequences L_T and L_G is reported in Figure 4.2.

We now construct the exploration scheme $\mathcal{E}_{G,S,s} = (\mathbb{X}, \mathbb{Y})$ for G. Initially $\mathbb{X} = \mathbb{Y} = L_G$. Then, the pairs of adjacent steps $\langle x_i, x_{i+1} \rangle$ and $\langle y_i, y_{i+1} \rangle$ are considered from $i = 1, \ldots, k$. If $x_i = y_i = v'$ and $x_{i+1} = y_{i+1} = v''$, where v'' is an unexplored node occurring for the first time in the sequences, we replace $\langle v', v'' \rangle$ in $\mathbb{X}$ with the sequence $\langle v', v'', v', v'' \rangle$ and we replace $\langle v', v'' \rangle$ in $\mathbb{Y}$ with the sequence $\langle v', v', v', v'' \rangle$. This is to assure that each time the agents have to visit an unexplored node v'', such visit is replaced by a **probe-and-visit**(v'') phase. Since u is the number

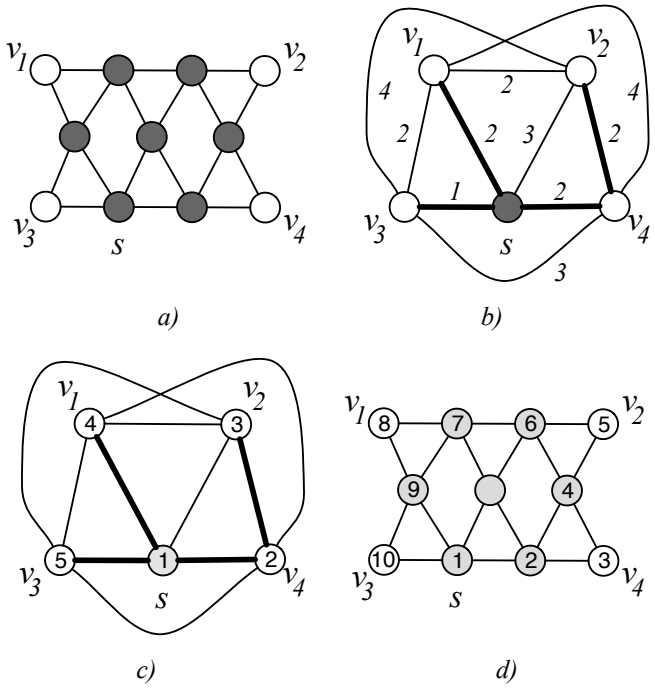

Figure 4.2: In $b)$ the graph $\widehat{G}$ obtained from an instance (G, S, s) (in $a)$). The nodes in S are filled with gray color. In $b)$ the bold lines denote a minimum spanning tree T of $\widehat{G}$ rooted at S. In $c)$ it is presented the ordering of the nodes in the sequence L_T. In $d)$ it is presented the ordering of the nodes in the corresponding sequence L_G.

of unexplored nodes, $2u$ steps are added to exploration scheme $\mathcal{E}_{G,S,s}$. The length of $\mathcal{E}_{G,S,s}$ is therefore at most $2cost(T) - d(v_{\pi(u)}, s) + 2u$, while the execution time in the case $B = \emptyset$ is at most $2cost(T) + 2u$ since the surviving agents have to get back from $v_{\pi(u)}$ to s.

Lemma 4.2.1 *The exploration scheme $\mathcal{E}_{G,S,s}$ is feasible and can be constructed in polynomial time. Moreover,* $\text{cost}(\mathcal{E}_{G,S,s}) \leq 2\text{cost}(T) + 2u$.

Proof. Constraints 1 and 2 can be easily checked by observing that sequence L_G (from which $\mathbb{X}$ and $\mathbb{Y}$ are derived) is a concatenation of paths in G, starting from s and ending in $v_\pi(u)$. All the nodes in U are in $\widehat{G}$, in L_T and thus in L_G; moreover the insertion of *probe* phases does not alter the set of visited nodes, hence the agents visit all the unexplored nodes (Constraint 3). Finally, we can observe that, except for the *probe* phases, the agents always move together along safe nodes, hence the explored territory increases only during the u *probe* phases; satisfaction of Constrain 4, and thus feasibility of $\mathcal{E}_{G,S,s}$, straightforwardly follows.
Graph $\widehat{G}$ can be constructed by computing the all-pair shortest paths in G; by using the best known algorithm ([37]), this operation has cost $O(n^\omega \log n)$, where $O(n^\omega)$ is the cost of a matrix product computation. This is the dominating cost of the whole algorithm, since the computation of the spanning tree T of $\widehat{G}$, as the computation of L_T and L_G can be all performed in linear time.
For what concerns the cost of $\mathcal{E}_{G,S,s}$, we can observe that, since the exploration traverses the edges of a tree rooted at s, Corollary 3.5.6 holds and thus $B = \emptyset$ yields the worst execution time. □

Let us consider now an optimal exploration scheme $\mathcal{E}^*_{G,S,s} = (\mathbb{X}^*, \mathbb{Y}^*)$. In computing $cost(\mathcal{E}^*_{G,S,s})$ we consider, as lower bound, the execution time of $\mathcal{E}^*_{G,S,s}$ in the case $B = \emptyset$. Let $L' = \langle x_k, \ldots, s\rangle$ be the shortest path in G from the last node x_k in $\mathbb{X}^*$ to the starting node, excluding the endpoints x_k and s. Let $L'' = \mathbb{X}^* \circ L' \circ \mathbb{Y}^* \circ L' \circ \langle s\rangle$. The sequence L'' starts from s, visits all the nodes in U and ends in s. The length of L'' (we denote it as $|L''|$) is at most twice the execution time of $\mathcal{E}^*_{G,S,s}$ in the case $B = \emptyset$, since L'' is the concatenation of the paths the two agents follow during the exploration in such case; hence $2cost(\mathcal{E}^*_{G,S,s}) \geq |L''|$. Let L^* be the minimum (shortest) tour in G starting from s and visiting all the nodes in U, and let $|L^*|$ be its length; obviously, $|L''| \geq |L^*|$.
Due to its optimality, L^* has the following structure:

$$L^* = \langle s\rangle \circ P(s, u_1\rangle \circ P(u_1, u_2\rangle \circ \ldots \circ P(u_u, s\rangle$$

where $\langle u_1, \ldots, u_u\rangle$ is the sequence of unexplored nodes in the order they are visited for the first time in L^*, and $P(x, y\rangle$ is the shortest path from node x (excluded) to node y in G. Since weights in G satisfy triangle inequality, the length of L^* is equal to the length of the minimum traveling salesman tour in $\widehat{G}$, which is, by a well known

relation, not less than the cost of the minimum spanning tree T of $\widehat{G}$. Therefore, $|L^*| \geq \mathit{cost}(T)$, and

$$\mathit{cost}(\mathcal{E}^*_{G,S,s}) \geq \frac{\mathit{cost}(T)}{2}. \tag{4.1}$$

Moreover, since the agents cannot explore more than two nodes every two steps, the trivial lower bound still holds:

$$\mathit{cost}(\mathcal{E}^*_{G,S,s}) \geq u. \tag{4.2}$$

We compute the approximation ratio of the algorithm presented in this section, by picking the numerator from Lemma 4.2.1 and by choosing a suitable balance for Equations (4.1) and (4.2) as the denominator. Therefore:

$$\frac{\mathit{cost}(\mathcal{E}_{G,S,s})}{\mathit{cost}(\mathcal{E}^*_{G,S,s})} \;\leq\; \frac{2\,\mathit{cost}(T) + 2\,u}{\frac{2}{3}\,\frac{\mathit{cost}(T)}{2} + \frac{1}{3}\,u} \;=\; 6\,. \tag{4.3}$$

Theorem 4.2.2 *The* BHS problem *problem is approximable within* 6.

Chapter 5

The Restricted BHS Problem

In this chapter we investigate a particular subcase of the BHS problem defined in Section 3.4. Here we assume that the nature (but not the topology) of the network is completely unknown and thus the only node initially known to be safe is the starting node, i.e., $S = \{s\}$. We can therefore formulate the problem as follows.

RESTRICTED MINIMUM COST BHS PROBLEM (rBHS problem)

Instance : a pair (G, s), where $G = (V, E)$ is a connected undirected graph, and $s \in V$.

Solution : a feasible EXPLORATION SCHEME $\mathcal{E}_{G,s} = (\mathbb{X}, \mathbb{Y})$ for (G, s).

Measure : *cost*$(\mathcal{E}_{G,s})$, i.e., the cost of the BHS based on $\mathcal{E}_{G,s}$.

Goal : minimization.

This problem has been presented and studied in [12]. It is worthwhile to remember that, when studying a particular subcase of an **NP**-hard problem, while the upper bounds (approximation algorithms) are still valid (but better upper bounds can possibly be found), the lower bounds (**NP**-hardness and non-approximability results) may not hold anymore. Indeed, in this chapter we will prove that the rBHS problem remains **NP**-hard, even when restricted to networks for which a planar embedding can be found (Section 5.1). In Section 5.2 we show that the rBHS problem problem is **APX**-hard. In Section 5.3 we present a $\frac{10}{7}$-approximating algorithm applicable when the network can be represented as a tree. In Section 5.4 we show that a $\frac{27}{8}$-approximating algorithm for general networks can be obtained by applying the algorithm described in Section 5.3 on a suitable spanning tree of the network. In Section 5.4.3 we discuss further about the spanning-tree approach we used, show its limitations and present different approaches. A preliminary version of part of these results can be found in [29].

5.1 NP-Hardness for arbitrary planar networks

In this section we prove the **NP**-hardness of the rBHS problem in planar graphs by providing a reduction from a specific version of the Hamiltonian Cycle problem to the decision version of the rBHS problem.

Hamiltonian Cycle problem for cubic planar graphs (cpHC problem)

Instance : a cubic planar 2-edge-connected graph $G = (V, E)$, and an edge $(x, y) \in E$;

Question : does G contain a Hamiltonian cycle that includes edge (x, y)?

Decision Black Hole Search problem for planar graphs (dBHS problem)

Instance : a planar graph $G' = (V', E')$, with a starting node $s \in V'$, and a positive integer X;

Question : does there exist an exploration scheme $\mathcal{E}_{G',s}$ for G' starting from s, such that *cost*$(\mathcal{E}_{G',s}) = X$?

Lemma 5.1.1 *The* cpHC problem *is* ***NP****-complete.*

Proof. The **NP**-completeness of the cpHC problem problem without the extra requirement that the Hamiltonian cycle passes through a given edge was proved in [23]. The version with that extra requirement is obviously in **NP**, and we can define the following simple reduction. Let a given cubic planar graph G be an instance of the original problem. We obtain an instance $\tilde{G}$ of the cpHC problem in the following way. Let D be any node in G and let A, B and C be its neighbors. We add to G six new nodes and replace the edges adjacent to D with the edges as in Figure 5.1(a). Edge (x, y), the second part of the input instance, is picked as in Figure 5.1(a). Observe that planarity and 2-edge connectivity are preserved in $\tilde{G}$. It should be clear that if graph $\tilde{G}$ has a Hamiltonian cycle containing edge (x, y), then graph G has a Hamiltonian cycle as well. Figure 5.1(b) shows that the implication in the other direction is also true: if graph G has a Hamiltonian cycle, then graph $\tilde{G}$ has a Hamiltonian cycle containing edge (x, y). □

We describe now a polynomial time reduction from the cpHC problem to the dBHS problem. We want to achieve the property that if G can be traversed in a number of steps equal to the number of nodes in it, then G' can be explored in a number of time units equal to the number of unexplored nodes in it. Therefore, our goal is to construct a graph which facilitates its exploration through split phases, without any extra walk. Let $G = (V, E)$ and $(x, y) \in E$ be an instance of the cpHC problem. We construct the corresponding instance of the dBHS problem, i.e., a graph G', a starting node s, and an integer X, by modifying graph G in the following steps.

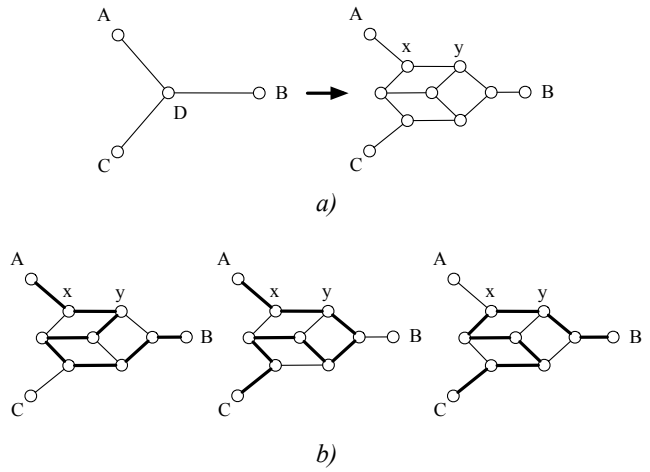

Figure 5.1: a) Reduction from the cpHC problem with no fixed edge to the cpHC problem with a fixed edge (x, y). b) Extensions of Hamiltonian cycles in graph G to Hamiltonian cycles in graph $\tilde{G}$ passing through edge (x, y).

1. Replace in G the edge (x, y) with the edges (x, s) and (s, y), where $s \notin V$ is a new node, obtaining graph $\bar{G}$.

2. Let $\mathcal{F}$ be the set of the faces of an arbitrary planar embedding of graph $\bar{G}$. We identify each face $f \in \mathcal{F}$ with the sequence of the consecutive edges adjacent to this face (starting with any edge adjacent to f and traversing the boundary of f in either of the two directions).

3. For each face $f \in \mathcal{F}$ and each edge (v, w) adjacent to f, add one new node $z_f^{(v,w)}$ and two edges $(v, z_f^{(v,w)})$ and $(w, z_f^{(v,w)})$. Since graph G is planar and 2-edge connected, each edge e in graph $\bar{G}$ is adjacent to exactly two different faces f' and f'' in $\mathcal{F}$. The two nodes $z_{f'}^e$ and $z_{f''}^e$ in G' added for edge e are called the *twin nodes* for edge e.

4. For each face $f = \langle e_1, e_2, \ldots, e_q \rangle \in \mathcal{F}$ add the *shortcut edges* $(z_f^{e_1}, z_f^{e_2})$, $(z_f^{e_2}, z_f^{e_3}), \ldots, (z_f^{e_q}, z_f^{e_1})$.

5. For each node $v \in V \cup \{s\} \setminus \{x\}$, add a new node v^F, called the *flag node* of node v, and an edge (v, v^F).

6. Let G' be the obtained graph. Set X to $n' - 1 = 5n + 2$, where $n' = n + 1 + 2(e+1)+n = 5n+3$ is the number of nodes in G' and n and e are, respectively, the number of nodes and edges in G (in a cubic graph, $e = (3/2)n$).

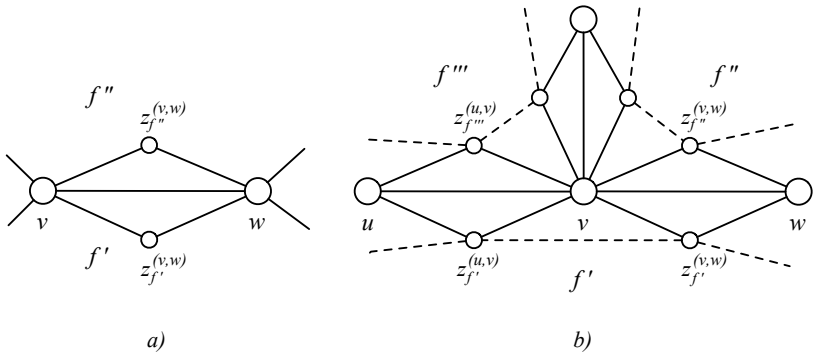

Figure 5.2: In *a)*, the two twin nodes for the edge (v, w); in *b*), the twin nodes for the edges (u, v) and (v, w) and their neighborhood (shortcut edges are dashed).

The construction of graph G' is illustrated in Figure 5.2. Graph G' is planar and can be constructed in linear time. The nodes in G' inherited from graph $\bar{G}$ are called the *original nodes*.

The two following lemmas state two properties of graph G' which we use in further arguments.

Lemma 5.1.2 *Let $\langle u, v, w\rangle$ be a path in graph $\bar{G}$. Then there is a path $\langle u, z', z'', w\rangle$ in G' bypassing node v (that is $v \notin \{z', z''\}$).*

Proof. Since the degree of each node in $\bar{G}$ is at most 3, there must be a face $f \in \mathcal{F}$ to which both edges (u, v) and (v, w) are adjacent. By construction, there exist two adjacent nodes $z_f^{(u,v)}$ and $z_f^{(v,w)}$ in G and thus the sequence of nodes $\left\langle u, z_f^{(u,v)}, z_f^{(v,w)}, w\right\rangle$ is a path in G'. □

Lemma 5.1.3 *Each twin node in G' has degree* 4.

Proof. For each twin node $z_f^{(u,v)}$, the nodes u and v are added as neighbors during step 3, and the two twin nodes $z_f^{(u,w)}$ and $z_f^{(z,v)}$ are added during step 4 of the construction of G'.

Lemmas 5.1.4 and 5.1.5 prove that graph G has a Hamiltonian cycle passing through edge (x, y) if and only if there is an exploration scheme for (G', s) with cost $X = 5n + 2$.

Lemma 5.1.4 *If graph G has a Hamiltonian cycle that includes edge (x, y), then there exists an exploration scheme $\mathcal{E}^*_{G',s}$ on graph G' from the starting node s, such that* $\text{cost}(\mathcal{E}^*_{G',s}) = 5n + 2$.

Proof. Let $\{v_1 = y, e_1, v_2, \ldots, e_{n-1}, v_n = x, e_n, v_1 = y\}$ be such Hamiltonian cycle in G. Consider the exploration scheme $\mathcal{E}^*_{G',s}$ defined by the following sequence of phases:

1. **b-split**(s, s^F, y), where s^F is the flag node of s;
2. **a-split**(s, z_1, z_2, y), where z_1 and z_2 are the twin nodes of the edge (s, y);
3. for each node v_i of the Hamiltonian cycle, with $(i = 1, \ldots, n-1)$:
   (a) let v_j be the third neighbor of v_i, other than v_{i-1} and v_{i+1}; if $j > i$ then **b-split**(v_i, z_1, z_2), where z_1 and z_2 are the twin nodes of (v_i, v_j);
   (b) **b-split**(v_i, v_i^F, v_{i+1}), where v_i^F is the flag node of v_i;
   (c) **a-split**(v_i, z_1, z_2, v_{i+1}), where z_1 and z_2 are the twin nodes of the edge (v_i, v_{i+1});
4. **a-split**(x, z_1, z_2, s), where z_1 and z_2 are the twin nodes of the edge (x, s).

Let us compute the length of $\mathcal{E}^*_{G',s}$. Since *a-split* and *b-split* phases have length 2 and increase the explored territory by 2 nodes (see Section 3.5), the overall number of phases is $(5n + 2)/2$ and hence $\mathcal{E}^*_{G',s}$ has length $5n + 2$. Notice that this is also the exploration time for $\mathcal{E}^*_{G',s}$, in the case $B = \emptyset$, since $\mathcal{E}^*_{G',s}$ ends in s.

Now we have to check that this is also the cost of $\mathcal{E}^*_{G',s}$, i.e. there is no allocation of the black hole that yields a larger exploration time. Unfortunately, graph G' does not have the properties for which Lemma 3.5.5 can be applied; by removing, for instance, an original node v adjacent to s, the remaining neighbors of v may increase their distance from s. However, we can still prove such a result for $\mathcal{E}^*_{G',s}$. We first observe that the set of meeting points in $\mathcal{E}^*_{G',s}$ is $\{v_i : 1 \leq i \leq n\} \cup \{s\}$.

Claim 1 *Consider the meeting step when the agents are to meet at a node v_i ($1 \leq i \leq n$). If a black hole has been just discovered, then the remaining exploration time for this case is not greater than the remaining exploration time for the case $B = \emptyset$.*

Proof. If the black hole is the flag node v_i^F (phase $3.b$) or one of the twin nodes for the edge (v_{i-1}, v_i) or for the edge (v_i, v_j) (phase $3.c$ or $3.a$), then the surviving agent can reach s by following the remaining part of the Hamiltonian Cycle, and hence the remaining cost is at most: $n + 1 - i$. If the black hole is at node v_{i+1} (phase $3.b$), then, by Lemma 5.1.2, there is a path of length 4 in G' from v_i to v_{i+2} bypassing node v_{i+1} (where v_{i+2} is node s, if $i + 1 = n$). Therefore the surviving agent can reach node v_{i+2} (or s) by using this safe path and then, as before, he can follow the remaining part of the Hamiltonian Cycle to reach s. The remaining cost is at most $n + 2 - i$. If $B = \emptyset$, then the remaining cost is at least: $2(n + 1 - i) \geq n + 2 - i$. This concludes the proof of the claim.

Observe that the exploration scheme $\mathcal{E}^*_{G',s}$ is optimal since, by Lemma 3.5.2, the exploration of $5n + 2$ nodes requires at least $5n + 2$ time units. □

Lemma 5.1.5 *If there exists an exploration scheme $\mathcal{E}_{G',s}$ on G' starting from s such that* cost$(\mathcal{E}_{G',s}) = 5n + 2$*, then the graph G has a Hamiltonian cycle that includes edge (x, y).*

Proof. By Lemma 3.5.2, each phase of $\mathcal{E}_{G',s}$ has length at least two and cannot enlarge the explored territory by more than two nodes. Since G' has $5n + 2$ unexplored nodes, $\mathcal{E}_{G',s}$ must end in s, and each of its phases must be either an *a-split* or a *b-split*. Consider now the sequence $M_{\mathcal{E}}$ of the meeting points established for $\mathcal{E}_{G',s}$ at the end of each *a-split*, excluding the last one which is s. Each meeting point v_i in $M_{\mathcal{E}}$ other than s must have at least degree 5 since one neighbor is needed for the initial exploration of v_i, two unexplored neighbors are needed for the *a-split* that ends in v_i and two further unexplored neighbors are needed for the *a-split* that leaves v_i. For this reason only the original nodes of G' can be in $M_{\mathcal{E}}$ (flag nodes have degree 1 and twin nodes, by Lemma 5.1.3 have degree 4).

Claim 2 *The nodes x and y must be the two endpoints of $M_{\mathcal{E}}$, node s cannot be in $M_{\mathcal{E}}$, and each original node must be in $M_{\mathcal{E}}$.*

Proof. Since s is the only initially safe node, the very first phase has to be a *b-split* from s. The first *a-split* in $\mathcal{E}_{G',s}$ is from s to x or y, while the last *a-split* (ending in s) starts from the other of these two nodes x, y. If s is also an intermediate meeting point, then we need another *a-split* to s. Since each of these four phases requires two unexplored neighbors, s has to have degree at least 8, but, by construction, its degree is only 7. Contradiction. Finally, for each node v in G, its flag node v^F has to be explored with a *b-split* having as meeting point node v. Hence v must be in $M_{\mathcal{E}}$.

Now we prove that the sequence $M_{\mathcal{E}}$ defines a Hamiltonian cycle on G by showing that it has these two further properties:

a) each node of G appears at most once in $M_{\mathcal{E}}$;

b) if nodes v_i and v_j are consecutive in $M_{\mathcal{E}}$, then the edge (v_i, v_j) must be in G.

To prove *a)*, it suffices to count the number of distinct neighbors needed by a node v_i in $M_{\mathcal{E}}$. At least one neighbor is needed for the initial exploration of v_i (two neighbors, if an *a-split* explores v_i). Then, for each occurrence of v_i in $M_{\mathcal{E}}$, two unexplored neighbors are needed for the *a-split* that ends in v_i, and two additional unexplored neighbors are needed for the *a-split* that leaves v_i. Moreover the flag node v_i^F has to be explored with a *b-split* from v_i, hence another unexplored neighbor of v_i is needed. If the node v_i occurs k times in $M_{\mathcal{E}}$, then the total number of neighbors needed by v_i is at least $1 + 4k + 2 = 3 + 4k$. Since each original node in G' has only 10 neighbors (as G is a cubic graph), it must be $k \leq 1$, thus each node appears at most once in $M_{\mathcal{E}}$.

Now we prove property *b)* of $M_{\mathcal{E}}$. According to the structure of G', *a-split* operations having original nodes as meeting points, can either explore two twin nodes of an original edge (in this case property *b)* is satisfied since the meeting point is

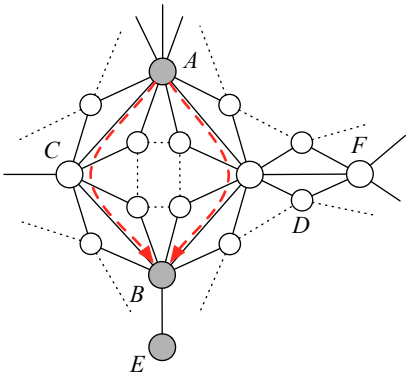

Figure 5.3: A big *a-split* from A to B. Flag nodes are not shown, the shaded nodes are already explored.

adjacent in G to the previous one), or explore two original nodes of G' and meet in another original node which may not be adjacent to the previous meeting point, thus violating property *b)*.

Suppose that this latter kind of split (a *big a-split*) happens from a node A to a node B; see Figure 5.3. In order to do this, A must have two unexplored original neighbors (C and D in the figure) both having B as a neighbor. B must be already explored, therefore the last original neighbor of B (E in the figure) must have already been a meeting point (we can suppose without loss of generality that the one from A to B is the first *big a-split* in $M_{\mathcal{E}}$). At this point no other *big a-splits* can be performed from B (all its original neighbors are now explored) and, by property *a)*, E cannot be again a meeting point, thus the sequence $M_{\mathcal{E}}$ can have either C or D as the next meeting point. Supposing that C is that one, consider the instant when D becomes a meeting point. We cannot get to D with a *big a-split*, since D does not have two neighbors in G that are unexplored, hence also F has been already a meeting point. Now all the original neighbors of D have already been a meeting point in $M_{\mathcal{E}}$, and none of them can be s, thus there is no way to leave D without violating property *a)*. Therefore there cannot be any *big a-split* in $\mathcal{E}_{G',s}$, and thus property *b)* is verified.

We have proved that, if there exists an exploration scheme such that *cost*$(\mathcal{E}_{G',s}) = 5n + 2$, then G has a Hamiltonian cycle ($M_{\mathcal{E}}$) that includes edge (x, y). □

Lemma 5.1.1, together with Lemma 5.1.4 and Lemma 5.1.5, imply the following theorem.

Theorem 5.1.6 *The* dBHS problem *is NP-hard.*

5.2 Approximation lower bound

Can we do something better than proving **NP**-hardness for the rBHS problem? The rather simple reduction described in Section 4.1 requires the bridge nodes to be initially safe, and hence it cannot be straightforwardly applied to the restricted case. Indeed, we could not prove the same approximation lower bound for the BHS problem. However, we can still prove **APX**-hardness of the rBHS problem problem by deriving it from the **APX**-hardness of the TSP(1,2) problem. We first recall Lemma 6.3 from [10]:

Lemma 5.2.1 *Assume we are given an instance of* TSP(1,2) *on the n-node complete graph $\overline{G}$, in the form of the subgraph G of $\overline{G}$ containing the edges of weight 1. Assume that G has max degree 3. Assume that we know that its minimum cost TSP tour is either of cost n or at least $(1+\varepsilon_0)n$, for some fixed ε_0. Then there exists such a constant ε_0 for which it is **NP**-hard to decide which of the two cases holds. The claim holds for $\varepsilon_0 = \frac{1}{786}$. If G is cubic then the claim holds for $\varepsilon_0 = \frac{1}{1290}$.*

With a small abuse of notation we define the cost of a tour in G as the cost of the corresponding TSP tour in the complete graph $\overline{G}$. We show a polynomial-time reduction algorithm $\mathcal{A}$ from TSP(1,2) to rBHS problem, which takes as input an instance G of TSP(1,2), computes an instance (G', s) of rBHS problem, and has the following property.

Lemma 5.2.2 *Let $0 < \varepsilon < \varepsilon_0/7$, let G be an n-node cubic graph (an instance of* TSP(1,2)*), and let (G', s) be the corresponding instance of* rBHS problem *computed by the reduction algorithm $\mathcal{A}$. Then the following two conditions hold.*

1. *If the optimal cost of a tour in G is equal to n, then the optimal cost of an exploration scheme for (G', s) is at most $\frac{7}{2}n + 1$.*
2. *There exists $n_0 = n_0(\varepsilon_0, \varepsilon)$ such that for $n \geq n_0$, if the optimal cost of a tour in G is at least $n(1+\varepsilon_0)$, then the optimal cost of an exploration scheme for (G', s) is greater than $\left(\frac{7}{2}n + 1\right)(1+\varepsilon)$.*

This lemma implies that for $0 < \varepsilon < \varepsilon_0/7$ and $n \geq n_0$, if we have an n-node cubic graph G and we know that the optimal cost of a tour in G either is equal to n or is at least $n(1+\varepsilon_0)$, then we can decide which of these two cases happens, if we have an $(1+\varepsilon)$-approximation of the optimal cost of an exploration scheme for (G', s). Thus Lemmas 5.2.1 and 5.2.2 imply the following theorem.

Theorem 5.2.3 *It is **NP**-hard to compute $(1+\varepsilon)$-approximate exploration schemes for the* rBHS problem *problem for any $\varepsilon < \frac{1}{9030}$.*

Description of the reduction algorithm $\mathcal{A}$.

Let an n-node cubic graph $G = (V, E)$ be the input instance of TSP(1,2). The construction of the instance (G', s) of rBHS problem is inspired by the construction presented in Section 4.1. The main differences are that here we do not add bridges corresponding to edges of weight 2 and that, obviously, all nodes but the starting node s are initially unexplored. To facilitate their exploration, all the bridge nodes are connected to s. More precisely, the construction of (G', s) proceeds as follows. We pick an arbitrary node in G (say v_1) as the starting node ($s \equiv v_1$) and we add it to G' (as before, this is island I_1). For each node v_i in G, $2 \leq i \leq n$, we add in G' a pair of unexplored nodes v'_i, v''_i (as before, we denote this pair as island I_i). For each edge (v_i, v_j) in G, we put in G' an unexplored node $b_{i,j}$ (*bridge node*), connected to v'_i, v''_i (if $i > 1$), to v'_j, v''_j (if $j > 1$) and to s. If the number of bridge nodes (that is, the number of edges in G) is odd, then we add another unexplored node b_s adjacent to s (to ensure that s is adjacent to an even number of unexplored nodes). Note that s is adjacent to all bridge nodes and is not adjacent to any "island" nodes.

Proof of Lemma 5.2.2.

Let G be an n-node cubic graph. Since G has $e = \frac{3}{2}n$ edges, the total number of nodes in G' is $\frac{7}{2}n - 1 + odd(e)$, and all of them but one are initially unexplored. The function $odd : \mathbb{N} \mapsto \mathbb{N}$ is defined as $odd(x) = x \pmod 2$, i.e., $odd(x)$ is equal to 1, if x is odd, and to 0 otherwise. As in Section 4.1, we define for a tour $\tau = \langle v_1, v_{\pi(2)}, \ldots, v_{\pi(n)} \rangle$ in G, the τ-based exploration scheme $\mathcal{E}^{\tau}_{G',s}$ for (G', s), which explores two by two the nodes of each island in the order $\langle I_{\pi(2)}, \ldots, I_{\pi(n)} \rangle$. Here, however, the scheme first explores the bridge nodes.

More formally, the scheme $\mathcal{E}^{\tau}_{G',s}$ has the following sequence of steps.

1. While there are two unexplored nodes b', b'' adjacent to s: **b-split**(s, b', b'').
2. For each $i = 2, \ldots, n$:
   (a) **walk**(b'), where b' is either the bridge node $b_{\pi(i-1),\pi(i)}$, if nodes $v_{\pi(i-1)}$ and $v_{\pi(i)}$ are adjacent in G, or any bridge node adjacent to I_i otherwise.
   (b) **a-split**$(b', v'_{\pi(i)}, v''_{\pi(i)}, b'')$, where b'' is either the bridge node $b_{\pi(i),\pi(i+1)}$, if $i < n$ and nodes $v_{\pi(i)}$ and $v_{\pi(i+1)}$ are adjacent in G, or any bridge node adjacent to I_i otherwise.

Note that the first **walk** operation, for $i = 2$, has length 1. For each $3 \leq i \leq n$, the **walk** operation has length either 0, if nodes $v_{\pi(i-1),\pi(i)}$ are adjacent in G, or 2, if nodes $v_{\pi(i-1),\pi(i)}$ are not adjacent in G. Therefore, if the tour τ has cost $n + d$ (that is, contains d edges of weight 2), then the exploration scheme $\mathcal{E}^{\tau}_{G',s}$ has length at most $\frac{3}{2}n + odd(e) + 1 + 2d + 2(n-1) \leq \frac{7}{2}n + 2d$. The execution time for the case $B = \emptyset$ is at most $\frac{7}{2}n + 2d + 1$, since $\mathcal{E}^{\tau}_{G',s}$ ends in a bridge node, which is adjacent to

s. This is also the cost of the BHS based on $\mathcal{E}^{\tau}_{G',s}$. When an agent realizes that there is a black hole, then this agent must be at a meeting point, and each meeting point is either node s or a bridge node, which is adjacent to s. Hence, if the cost of tour τ is n, then $d = 0$ and the cost of $\mathcal{E}^{\tau}_{G',s}$ is at most $\frac{7}{2}n + 1$, so the first part of Lemma 5.2.2 holds.

To prove the second part of Lemma 5.2.2, consider an arbitrary exploration scheme $\mathcal{E}_{G',s}$. By using a similar approach as in Section 4.1, we can find, through a sequence of phase swaps, a "regular" exploration scheme $\mathcal{E}'_{G',s}$, equivalent to $\mathcal{E}_{G',s}$, where each agent explores exactly one node of each island I_j for $j = 2, \ldots, n$, and $\mathit{cost}(\mathcal{E}'_{G',s}) = \mathit{cost}(\mathcal{E}_{G',s})$. We assume by symmetry that scheme $\mathcal{E}'_{G',s}$ is such that *Agent*-1 explores nodes v'_j, $j = 2, \ldots, n$, and that $\langle v'_{\pi(2)}, \ldots, v'_{\pi(n)} \rangle$ is the order in which *Agent*-1 explores these nodes. We consider the tour $\tau = \langle v_1, v_{\pi(2)}, \ldots, v_{\pi(n)} \rangle$ in G. Let d be the number of weight 2 edges in τ. Thus the number of indexes i, $2 \leq i \leq n-1$, such that $(v_{\pi(i)}, v_{\pi(i+1)})$ is not an edge in G is at least $d-2$. Consider any of these indexes i and two consecutive phases ϕ_{j_i} and ϕ_{j_i+1} in $\mathcal{E}'_{G',s}$, where ϕ_{j_i} is the phase during which node $v'_{\pi(i)}$ is explored by *Agent*-1. If phase ϕ_{j_i} is a ***split*** (an ***a-split*** or a ***b-split***), then the meeting point at the end of this phase is a bridge node x adjacent to island $I_{\pi(i)}$. The next phase ϕ_{j_i+1} is either phase $\phi_{j_{i+1}}$ when *Agent*-1 explores node $v'_{\pi(i+1)}$, or a phase when *Agent*-1 explores a bridge node y, or a phase when *Agent*-1 does not explore any new node. In the first case, phase ϕ_{j_i+1} is not a ***split*** because node $v'_{\pi(i+1)}$ is not adjacent to node x (there is no bridge node between islands $I_{\pi(i)}$ and $I_{\pi(i+1)}$). In the second case, phase ϕ_{j_i+1} is not a ***split*** because node y is not adjacent to node x (no two bridge nodes are adjacent). In the third case, phase ϕ_{j_i+1} is not a ***split*** because *Agent*-1 does not explore any new node. Thus at least one of the two phases ϕ_{j_i} and ϕ_{j_i+1} is not a split, so at least $(d-2)/2$ phases in scheme $\mathcal{E}'_{G',s}$ are not splits.

The cost of any exploration scheme is at least the number of unexplored nodes plus the number of phases other than splits. Therefore, we have

$$\mathit{cost}(\mathcal{E}'_{G',s}) \;\geq\; \frac{7}{2}n - 2 + \frac{d-2}{2} \;=\; \frac{7}{2}n - 3 + \frac{d}{2}.$$

This implies that if $\mathit{cost}(\mathcal{E}'_{G',s}) \leq \left(\frac{7}{2}n + 1\right)(1+\varepsilon)$, then

$$d \leq 7\varepsilon n + 2(4+\varepsilon),$$

and

$$\begin{aligned} \mathit{cost}(\tau) &= n + d \\ &\leq n + 7\varepsilon n + 2(4+\varepsilon) \\ &\leq n(1+\varepsilon_0) - (\varepsilon_0 - 7\varepsilon)n + 2(4+\varepsilon) \\ &< n(1+\varepsilon_0), \end{aligned}$$

provided that $\varepsilon < \varepsilon_0/7$ and $n \geq n_0 = \lceil 2(4+\varepsilon)/(\varepsilon_0 - 7\varepsilon) + 1 \rceil$.

5.3 A $\frac{10}{7}$-approximation algorithm for tree networks

In this section we focus our attention on the rBHS problem in the case that a network has a tree topology. The motivation is twofold. First, the design of exploration schemes in trees is simpler than in the case of arbitrary graphs since, roughly speaking, there are less "degrees of freedom" and moreover their analysis is simplified by application of Lemma 3.5.5. The second reason for considering trees approach is that, as we will see, the approach we followed for the general graph approximating algorithm, is to first compute a suitable spanning tree of the graph, and then to explore it by using the algorithm for trees and traversing only the tree edges of the graph.
These motivations make desirable to have an efficient algorithm for trees. Unluckily, we still do not have an answer to the following open question.

Open Question 1 *Does there exists a polynomial time algorithm for the* rBHS problem *in the case that the network can be represented as a tree ?*

The question was first posed by Czyzowicz *et. al.* in [12]. Indeed, they showed a linear time algorithm for constructing optimal exploration schemes for trees where each internal node has at least 2 children, called BUSHY TREES. The optimality of such algorithm arises from the property that, since each internal node has at least 2 children, each agent has always a close reachable unexplored node and hence does never have to wait for the other agent. As a result, the explored territory increases by two nodes in each phase, possibly excluding the last one, and optimality of such algorithm can be proved. Unfortunately, the algorithm does not extend to arbitrary trees. In [12], the authors also provide a simple general algorithm for arbitrary trees. The algorithm is guaranteed to produce exploration schemes whose cost is at most $\frac{5}{3}$ larger than the optimal cost. We can see our algorithm as a generalization of the bushy tree algorithm to arbitrary trees. It still produces optimal exploration schemes for bushy trees, but it improves the approximation ratio[1] for arbitrary trees from $\frac{5}{3}$ to $\frac{10}{7}$. We call our algorithm *Search-Tree(T,s)*.

Let T be a rooted n-node tree and let the starting node s be its root. We assume that $n \geq 2$. Algorithm *Search-Tree(T,s)* uses the following order L_T of the nodes of T other than the root (that is, all unexplored nodes in T). We first order the children of each node according to the number of descendants: a child with more descendants comes before a child with fewer descendants and the ties are resolved arbitrarily. Thus from now on T is an *ordered* rooted tree. Let $I_T = \langle w_1, w_2, \ldots, w_b \rangle$ be the sequence of the internal nodes of T in the depth-first order. The order L_T is a copy of the sequence I_T where each node w_i is replaced by the (ordered) list of its children. Observe that L_T contains indeed all nodes of tree T other than the root, and each of these nodes occurs in L_T exactly once. We denote the i-th node in the order L_T as v_i

[1] In this case we are abusing of the term *approximation ratio* since we still do not know whether the problem is **NP**-hard

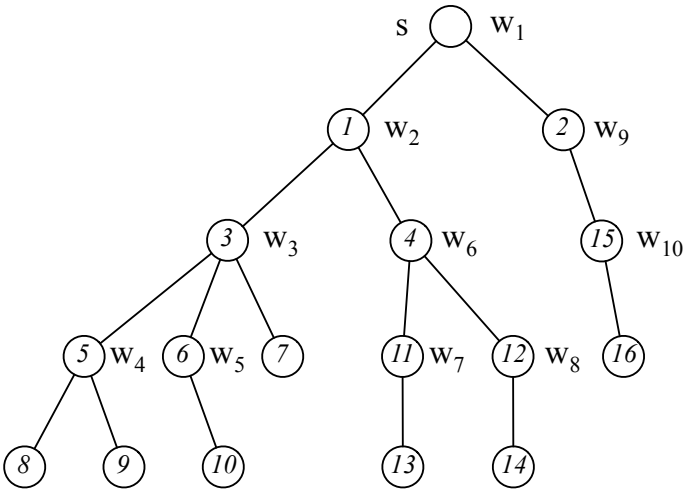

Figure 5.4: An ordered rooted tree T. The value inside each node is the position of the node in the L_T order. The internal nodes are also marked to show their depth-first order $I_T = \langle w_1, w_2, \ldots, w_{10} \rangle$.

and call it the i-th node of the tree. The odd (even) nodes of T are the nodes at the odd (even) positions in L_T. We denote the parent of node v_i as p_i. An example tree T and the corresponding order L_T of its nodes are given in Figure 5.4.

The two lemmas below, which follow from the construction of the sequence L_T, will be used to prove that algorithm *Search-Tree* returns feasible exploration schemes for trees.

Lemma 5.3.1 *In the sequence L_T, let the j-th node v_j be the parent of the i-th node v_i. Then $j < i$, and $i = j + 1$ if and only if node v_j does not have a sibling and node v_i is its first child.*

Proof. The parent p_j of node v_j precedes node v_j in the depth-first order I_T of the internal nodes. Thus all children of p_j, including node v_j, precede all children of v_j, including node v_i, in the sequence L_T, so $j < i$.

If node v_j does not have a sibling, then v_j must be immediately after p_j in the sequence I_T. In this case, when the sequence L_T is created from $I_T = \langle \ldots, p_j, v_j, \ldots \rangle$, the occurrence of node p_j in I_T is replaced by (its only child) v_j, while the occurrence of node v_j in I_T is replaced by the ordered list of its children. Thus if node v_i is the first child of node v_j, then v_i is immediately after v_j in the sequence L_T, that is, $i = j + 1$.

If node v_j has a right sibling r, then node r is after node v_j and before node v_i in L_T, so $i > j + 1$. If node v_j has a left sibling l, then node l must have at least

one child since the siblings are ordered according to the number of descendants and node v_j has at least one descendant. The children of node l are after node v_j and before node v_i in L_T, so $i > j + 1$. If node v_i is not the first child of node v_j, then all left siblings of v_i are after node v_j and before node v_i in L_T, so also in this case $i > j + 1$. □

Lemma 5.3.2 *Let v_i and v_{i+1} be two consecutive nodes in the sequence L_T, and let p_i and p_{i+1} be their parents. Then either nodes v_i and v_{i+1} are siblings, so $p_i = p_{i+1}$, or node p_{i+1} is the next node after node p_i in the depth-first order I_T of the internal nodes of T.*

Proof. Assume that nodes v_i and v_{i+1} are not siblings. Node p_i must occur in I_T before node p_{i+1}. If there was another (internal) node between p_i and p_{i+1} in I_T, then the children of this node would be between nodes v_i and v_{i+1} in L_T. □

We classify all nodes of tree T other than the root s into the following three disjoint types:

- *type-1* nodes: the leaf nodes;
- *type-3* nodes: the internal nodes with at least one sibling;
- *type-4* nodes: the internal nodes (other than the root) without siblings.

As we will see, we informally say that, in the exploration scheme which we construct for tree T a *type-t* node can be viewed as contributing t steps to the total cost. Note that there are no *type-2* nodes. We denote by x_t the number of *type-t* nodes.

We consider first the case when T does not have any *type-4* node and has an odd number $n = 2q + 1 \geq 3$ of nodes (that is, tree T has an even number of unexplored nodes $v_1, v_2, \ldots, v_{2q}$). *Agent*-1 will be exploring the odd nodes of T while *Agent*-2 will be exploring the even nodes.

For nodes u and r in tree T, let $P(u, r\rangle$ be the sequence of the nodes on the tree path from u to r excluding the first node u. If $u \equiv r$, then $P(u, r\rangle$ is the empty sequence. The exploration sequences $\mathbb{X}_T$ and $\mathbb{Y}_T$ for *Agent*-1 and *Agent*-2, respectively, are

$$\begin{aligned} \mathbb{X}_T &= \langle s\rangle \circ \phi_1^1 \circ \phi_2^1 \circ \cdots \circ \phi_q^1, \\ \mathbb{Y}_T &= \langle s\rangle \circ \phi_1^2 \circ \phi_2^2 \circ \cdots \circ \phi_q^2; \end{aligned}$$

where

$$\begin{aligned} \phi_j^1 &= P(p_{2j-2}, p_{2j-1}\rangle \circ \langle v_{2j-1}, p_{2j-1}\rangle \circ P(p_{2j-1}, p_{2j}\rangle, \\ \phi_j^2 &= P(p_{2j-2}, p_{2j-1}\rangle \circ P(p_{2j-1}, p_{2j}\rangle \circ \langle v_{2j}, p_{2j}\rangle. \end{aligned}$$

In the above formulae, operation "$\circ$" denotes concatenation of sequences; we define $p_0 \equiv s$. Note that the corresponding sub-sequences ϕ_j^1 and ϕ_j^2 in $\mathbb{X}_T$ and $\mathbb{Y}_T$ have the

same length and end at the same node p_{2j}. Indeed, we will show that ϕ_j^1 and ϕ_j^2 form the j-th phase of the exploration scheme $\mathcal{E}_{T,s} = (\mathbb{X}_T, \mathbb{Y}_T)$ (Lemma 5.3.3). Figure 5.5 provides an exhaustive enumeration of the possible relative locations of nodes v_{2j-2}, v_{2j-1}, v_{2j}, p_{2j-2}, p_{2j-1} and p_{2j}, leading to different types of sequences ϕ_j^1 and ϕ_j^2.

The idea behind the definition of sequences $\mathbb{X}_T$ and $\mathbb{Y}_T$ can be explained in the following way. If we remove from all ϕ_j^1 and ϕ_j^2 the segments $\langle v_{2j-1}, p_{2j-1}\rangle$ and $\langle v_{2j}, p_{2j}\rangle$, then both sequences $\mathbb{X}_T$ and $\mathbb{Y}_T$ coincide with the sequence

$$\langle s\rangle \circ P(p_0, p_1\rangle \circ P(p_1, p_2\rangle \circ \cdots \circ P(p_{2q-1}, p_{2q}\rangle.$$

Lemma 5.3.2 implies that this sequence is the depth-first traversal of the internal nodes of tree T ending when the last internal node is visited for the first time. Thus *Agent*-1 (*Agent*-2) follows the depth-first traversal of the internal nodes of T, and whenever it comes to an internal node p for the first time, it visits all children of p which are odd (even) nodes in T before continuing the traversal.

We prove now that $\mathcal{E}_{T,s} = (\mathbb{X}_T, \mathbb{Y}_T)$ is a feasible exploration scheme for tree T. We can easily check that $\mathcal{E}_{T,s}$ satisfies the feasibility Constraints 1–3. The lemma below identifies the phases of scheme $\mathcal{E}_{T,s}$ and states that each phase satisfies the conditions given in Constraint 4.

Lemma 5.3.3 *For each $j = 1, 2, \ldots, q$, the sub-sequences ϕ_j^1 and ϕ_j^2 within $\mathbb{X}_T$ and $\mathbb{Y}_T$ form the j-th phase of the feasible exploration scheme $\mathcal{E}_{T,s} = (\mathbb{X}_T, \mathbb{Y}_T)$; such phase satisfies the conditions stated in the feasibility Constraint 4.*

Proof. Let $m(0) = 0$, and for $j = 1, 2, \ldots, q$, let $m(j)$ denote the step in $\mathcal{E}_{T,s}$ where the sub-sequences ϕ_j^1 and ϕ_j^2 end. That is, the sub-sequences ϕ_j^1 and ϕ_j^2 occur within $\mathbb{X}_T$ and $\mathbb{Y}_T$, respectively, at the steps $\langle m(j-1)+1, \ldots, m(j)\rangle$. We prove by induction that for each $j = 1, \ldots, q$, the following statements are true.

1. The explored territory at step $m(j)$ is $S_{m(j)} = \{s, v_1, \ldots, v_{2j}\}$.
2. The sequence of steps $\langle m(j-1)+1, \ldots, m(j)\rangle$ in scheme $\mathcal{E}_{T,s}$ (where the sub-sequences ϕ_j^1 and ϕ_j^2 occur) is a phase and satisfies Constraint 4.

Note that, $S_{m(0)} = S_0 = \{s\}$. For the base step ($j = 1$), observe that

$$\begin{aligned} \phi_1^1 &= P(p_0, p_1\rangle \circ \langle v_1, p_1\rangle \circ P(p_1, p_2\rangle = \langle v_1, s\rangle, \\ \phi_1^2 &= P(p_0, p_1\rangle \circ P(p_1, p_2\rangle \circ \langle v_2, p_2\rangle = \langle v_2, s\rangle, \end{aligned}$$

because $p_0 = p_1 = p_2 = s$. Thus $m(1) = 2$, $S_{m(1)} = \{s, v_1, v_2\}$, and the steps $\langle 1, 2\rangle$ form a phase satisfying Constraint 4 (this phase is *b-split*(s, v_1, v_2)) so both Statements 1 and 2 hold.

Consider now any index j, $1 \leq j \leq q$ and assume that both Statements 1 and 2 are true for $j-1$. This assumption implies that $S_{m(j-1)} = \{s, v_1, v_2, \ldots, v_{2j-2}\}$ and

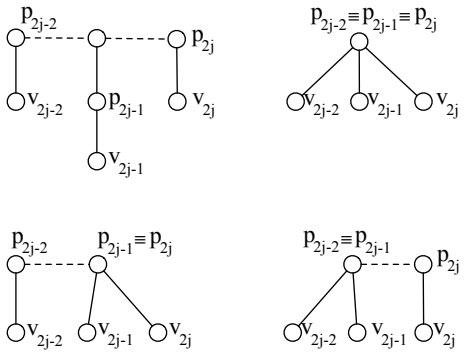

Figure 5.5: Different relative locations in T of three nodes v_{2j-2}, v_{2j-1} and v_{2j} consecutive in the L_T order and their parents p_{2j-2}, p_{2j-1} and p_{2j}. The dashed lines represent paths in the tree (which may be possibly empty in the first diagram).

that step $m(j-1)$ is a meeting step. (If $j \geq 2$, then $m(j-1)$ is a meeting step as the last step of the phase $\langle m(j-2)+1, \ldots, m(j-1)\rangle$. If $j = 1$, then step $m(j-1) = 0$ is by definition a meeting step.) By the definition of sequences $\mathbb{X}_T$ and $\mathbb{Y}_T$, the agents are at step $m(j-1)$ at the node p_{2j-2} (the parent of the node v_{2j-2}, or s if $j = 1$). Now *Agent*-1 and *Agent*-2 follow the sequences of nodes ϕ_j^1 and ϕ_j^2, respectively. Lemma 5.3.1 implies that the nodes p_{2j-2} and p_{2j-1} are in $S_{m(j-1)}$. Lemma 5.3.1 also implies that $p_{2j} \in S_{m(j-1)}$: if $p_{2j} \neq s$, then p_{2j} has a sibling, so p_{2j} is a node v_k for some $k \leq 2j-2$. Applying again Lemma 5.3.1, we conclude that all nodes in the sequences $P(p_{2j-2}, p_{2j-1}\rangle$ and $P(p_{2j-1}, p_{2j}\rangle$ must be in $S_{m(j-1)}$ as well, since each node in any of these two sequences is an ancestor of at least one of the nodes p_{2j-2}, p_{2j-1} and p_{2j}. Thus the only nodes in ϕ_j^1 and ϕ_j^2 which are not in $S_{m(j-1)}$ are node v_{2j-1} in ϕ_j^1 and node $v_{2j} \neq v_{2j-1}$ in ϕ_j^2. Therefore $S_{m(j)} = S_{m(j-1)} \cup \{v_{2j-1}, v_{2j}\}$ (so Statement 1 holds for j) and the sequence of steps $\langle m(j-1)+1, \ldots, m(j)\rangle$ satisfies Constraint 4. It remains to show that step $m(j)$ is the first meeting step after the meeting step $m(j-1)$, that is, to show that step $m(j)$ is the first step after step $m(j-1)$ when the explored territory increases.

Follow the agents' routes at steps $m(j-1)+1, \ldots, m(j)$ (see the diagrams in Figure 5.5). At the end of step $m(j-1)$ both agents are at the node p_{2j-2}, then they traverse together the (possibly empty) sequence of nodes $P(p_{2j-2}, p_{2j-1}\rangle$, not increasing the explored territory, and then they separate and meet again for the first time at step $m(j)$ at the node p_{2j}. At that step the explored territory increases from $S_{m(j-1)}$ to $S_{m(j)}$. Thus the sequence of steps $\langle m(j-1)+1, \ldots, m(j)\rangle$ is a phase in $\mathcal{E}_{T,s}$, so Statement 2 holds for j. This concludes the proof of the inductive step.

The lemma follows immediately from Statements 1 and 2. □

Lemma 5.3.4 *Let T be a tree rooted at s which has an odd number $n = 2q+1 \geq 3$ of nodes and does not have any* type-4 *nodes. The exploration scheme $\mathcal{E}_{T,s} = (\mathbb{X}_T, \mathbb{Y}_T)$ is feasible, can be constructed in time $O(n)$, and* cost$(\mathcal{E}_{T,s}) = x_1 + 3x_3$*, where x_t denotes the number of* type-t *nodes in T.*

Proof. The feasibility of the exploration scheme $\mathcal{E}_{T,s}$ follows from Lemma 5.3.3. The execution time of this scheme in the case when there is no black hole is equal to the length of $\mathcal{E}_{T,s}$ plus the distance from p_{2p} to s, that is, the length of the sequence $\mathbb{Y}_T \circ P(p_{2q}, s\rangle$ minus 1. By Corollary 3.5.6, this is also the cost of $\mathcal{E}_{T,s}$.
To obtain the length of the sequence $\mathbb{Y}_T \circ P(p_{2q}, s\rangle$, we separate it into two sub-sequences:

$$\langle s\rangle \circ P(p_0, p_1\rangle \circ P(p_1, p_2\rangle \circ \cdots \circ P(p_{2q-1}, p_{2q}\rangle \circ P(p_{2q}, s\rangle, \quad \text{and}$$
$$\langle v_2, p_2\rangle \circ \langle v_4, p_4\rangle \circ \cdots \circ \langle v_{2q}, p_{2q}\rangle.$$

Lemma 5.3.2 implies that the first sub-sequence is the depth-first traversal of the b internal nodes of T, so its length is $2b-1$. The length of the second sequence is $2q = n-1$. Thus the cost of the exploration scheme $\mathcal{E}_{T,s}$ is $(2b-1)+(n-1)-1 = (n-1)+2(b-1) = (x_1+x_3)+2x_3 = x_1+3x_3$.

Sequences $\mathbb{X}_T$ and $\mathbb{Y}_T$ can be constructed in time linear in the length of these sequences, so linear in the size of tree T. □

Now we generalize algorithm *Search-Tree* to trees T, which may have and even number of nodes and *type-4* nodes. For each *type-4* node v in T, we add a new leaf l as a sibling of v. If the total number of nodes, including the added nodes, is even, then we add one more leaf to an arbitrary internal node. The obtained tree T' is rooted at s, has an odd number of nodes and does not have any *type-4* nodes, so it satisfies the requirements of Lemma 5.3.4. We obtain an exploration scheme $\mathcal{E}_{T,s} = (\mathbb{X}_T, \mathbb{Y}_T)$ for tree T from the exploration scheme $\mathcal{E}_{T',s} = (\mathbb{X}_{T'}, \mathbb{Y}_{T'})$ for tree T' by replacing the traversals of the added edges with waiting. More precisely, if a node l is an added leaf, its parent is a node p, and l is an odd (even) node in tree T', then we replace the unique occurrence of l in $\mathbb{X}_{T'}$ (in $\mathbb{Y}_{T'}$) with p.

Lemma 5.3.5 *Let T be a tree rooted at s with $n \geq 2$ nodes. The exploration scheme $\mathcal{E}_{T,s} = (\mathbb{X}_T, \mathbb{Y}_T)$ for T is feasible, can be constructed in time $O(n)$ and*

$$\text{cost}(\mathcal{E}_{T,s}) \leq x_1 + 3x_3 + 4x_4 + odd(x_1 + x_3).^2 \tag{5.1}$$

Proof. The feasibility of the exploration scheme $\mathcal{E}_{T,s}$ and its construction in linear time follow from Lemma 5.3.4. Let β be equal to 1 if the extra node was added to the tree to have an odd number of nodes, and 0 otherwise. The cost of scheme $\mathcal{E}_{T,s}$ is

[2] Recall that $odd(x) = x \pmod 2$.

equal to the cost of scheme $\mathcal{E}_{T',s}$, which, by Lemma 5.3.4 is equal to $x'_1 + 3x'_3$. The number of leaves in tree T' is $x'_1 = x_1 + x_4 + \beta$, while the number of *type-3* nodes in T' is $x'_3 = x_3 + x_4$ (each *type-4* node in tree T becomes a *type-3* node in tree T'). Thus the cost of scheme $\mathcal{E}_{T,s}$ is equal to $x'_1 + 3x'_3 = x_1 + 3x_3 + 4x_4 + \beta$. The statement is proved by observing that the extra leaf is added if and only if $x'_1 + x'_3 = x_1 + x_3 + 2x_4$ is odd. □

We now prove that the algorithm defined so far is $\frac{10}{7}$-approximating, i.e. the cost of the exploration scheme $\mathcal{E}_{T,s}$ produced for any tree T is at most $\frac{10}{7}$ times larger than the optimal one.
We partition *type-3* and *type-4* nodes into two subsets each. We define as *type-3'* the nodes of *type-3* having only one descendant in T, and as *type-3"* the remaining nodes of *type-3*. By x'_3 and x''_3 we denote the number of *type-3'* and *type-3"* nodes respectively. As before, $x_3 = x'_3 + x''_3$. Analogously, we define as *type-4'* the nodes of *type-4* having only one descendant in T, and as *type-4"* the remaining nodes of *type-4*. By x'_4 and x''_4 we denote the number of *type-4'* and *type-4"* nodes respectively, and $x_4 = x'_4 + x''_4$.

We can reformulate Lemma 5.2 in [12], in the following terms:

Lemma 5.3.6 *For any exploration scheme $\mathcal{E}_{T,s}$ on T:*

$$\text{cost}(\mathcal{E}_{T,s}) \geq x_1 + 2x'_3 + 2x'_4 + 3x''_3 + 3x''_4 + odd(x_1 + x_3 + x_4).$$

Proof. Let us first consider the number of edge traversals that have to be performed by any BHS on T in the case that there are no black holes in T. In the following we denote an edge as (u, v), if u is the parent of v in T. We can observe that each edge (u, v) has to be traversed at least twice in order to move v into the explored territory. If v is a *type-1* node, then no further traversals are needed. Consider now an edge (u, v) where v is an internal node. Let l be the number of descendants of v. If $l \geq 2$ (i.e., v is a *type-4"* or a *type-3"* node) we can distinguish two cases. If, during any phase after exploration of v, edge (u, v) is traversed always by only one agent, then at least $2l \geq 4$ additional traversals are required (an agent has to traverse (u, v) two times for every descendant of v). If otherwise there is at least one phase after exploration of v where the edge is traversed by both agents, then at least 4 additional traversals of (u, v) are required for the exploration of the descendants of v (both agents traverse (u, v) and return). In this case the total minimum number of traversals of (u, v) is 6. If $l = 1$ (i.e., v is a *type-4'* or a *type-3'* node), then the branch of two edges having u as upper node can be traversed in the following way. 2 traversals are required for the exploration of node v. If during any phase after exploration of v, edge (u, v) is traversed always by only one agent, then at least 4 additional edge traversals on this branch are required. If there is at least one phase

after exploration of v in which this edge is traversed by both agents, then at least 6 additional edge traversals on this branch are required (both agents traverse edge (u, v), then one of them explores the lower edge and finally they return). Therefore the total minimum number of traversals on each of such branches is 6. The proof is completed by observing that in any BHS the two agents need at least $\lceil \frac{x}{2} \rceil$ time units to perform x traversals. □

We can make the following observations:
Observation 1: For each node of *type-3'* and *type-4'*, there exists a distinct child of *type-1* in T, hence:

$x'_3 + x'_4 = \alpha x_1$, where $0 \leq \alpha \leq 1$

Observation 2: For each of the nodes of *type-4'* there exists in T a distinct ancestor node of *type-3"* (unless T is a path, but in this case the produced exploration scheme $\mathcal{E}_{T,s}$ would be optimal). Hence:

$x'_4 = \beta x''_3$, where $0 \leq \beta \leq 1$

We now determine a ratio between the upper bound defined by Lemma 5.3.5 and the lower found stated by Lemma 5.3.6. Unfortunately, the two "oddities" in the upper and lower bound are not equivalent, and hence we have to distinguish two cases, according to the values of $o_u = odd(x_1 + x_3)$ and $o_l = odd(x_1 + x_3 + x_4)$.

1. $o_u = o_l$ or $o_l = 1$: in this case

$$\begin{aligned} \frac{cost(\mathcal{E}_{T,s})}{cost(\mathcal{E}^*_{T,s})} &\leq \frac{x_1 + 3x'_3 + 4x'_4 + 3x''_3 + 4x''_4}{x_1 + 2x'_3 + 2x'_4 + 3x''_3 + 3x''_4} \\ &= \frac{x_1 + 3\alpha x_1 + \beta x''_3 + 3x''_3 + 4x''_4}{x_1 + 2\alpha x_1 + 3x''_3 + 3x''_4} \\ &= \frac{(1 + 3\alpha)x_1 + (3 + \beta)x''_3 + 4x''_4}{(1 + 2\alpha)x_1 + 3x''_3 + 3x''_4} \leq \frac{4}{3} \end{aligned}$$

   since $\frac{1+3\alpha}{1+2\alpha} > \frac{4}{3} \Leftrightarrow \alpha > 1$.

2. $o_u = 1$ and $o_l = 0$: in this case $odd(x_4) = 1$ and $odd(x_1+x_3) = 1$. Note that if $x'_3+x'_4 = x_1$ ($\alpha = 1$) and $x'_4 = x''_3$ ($\beta = 1$) then $x'_4 = x_1-x'_3 = x''_3$ and hence $x_1 = x_3$. This is a contradiction since we assumed $o_u = odd(x_1 + x_3) = 1$. Therefore at least one of the two conditions must hold: $x'_4 = \beta x''_3 - 1$ or

$x'_3 + x'_4 = \alpha x_1 - 1$. If the former holds, then:

$$\begin{aligned}\frac{cost(\mathcal{E}_{T,s})}{cost(\mathcal{E}^*_{T,s})} &\leq \frac{x_1 + 3x'_3 + 4x'_4 + 3x''_3 + 4x''_4 + 1}{x_1 + 2x'_3 + 2x'_4 + 3x''_3 + 3x''_4} \\ &= \frac{x_1 + 3\alpha x_1 + \beta x''_3 - 1 + 3x''_3 + 4x''_4 + 1}{x_1 + 2\alpha x_1 + 3x''_3 + 3x''_4} \\ &= \frac{(1+3\alpha)x_1 + (3+\beta)x''_3 + 4x''_4}{(1+2\alpha)x_1 + 3x''_3 + 3x''_4} \leq \frac{4}{3}\end{aligned}$$

If the latter holds, then:

$$\begin{aligned}\frac{cost(\mathcal{E}_{T,s})}{cost(\mathcal{E}^*_{T,s})} &\leq \frac{x_1 + 3x'_3 + 4x'_4 + 3x''_3 + 4x''_4 + 1}{x_1 + 2x'_3 + 2x'_4 + 3x''_3 + 3x''_4} \\ &= \frac{x_1 + 3\alpha x_1 - 3 + \beta x''_3 - 1 + 3x''_3 + 4x''_4 + 1}{x_1 + 2\alpha x_1 - 2 + 3x''_3 + 3x''_4} \\ &= \frac{(1+3\alpha)x_1 + (3+\beta)x''_3 + 4x''_4 - 2}{(1+2\alpha)x_1 + 3x''_3 + 4x''_4 - 2}\end{aligned}$$

Since $odd(x_4) = 1$, $x_4 \geq 1$. If $x'_4 \geq 1$ then $x_1 \geq 2$ and $x''_3 \geq 1$, hence

$$\frac{cost(\mathcal{E}_{T,s})}{cost(\mathcal{E}^*_{T,s})} \leq \frac{4x_1 + 4x''_3 + 4x''_4 - 2}{3x_1 + 3x''_3 + 3x''_4 - 2} \leq \frac{10}{7}$$

We wish to remark that the cost of the exploration scheme $\mathcal{E}_{T,s}$ is at most $4/3 + O(1/n)$ times the optimal cost of an exploration scheme for T, and that both the algorithm and its worst case analysis could be further improved. For example, for the first diagram in Figure 5.5, *Agent*-2 obviously does not have to go to node p_{2j-1} on its way to explore node v_{2j}. If it omitted node p_{2j-1}, then the phase would have one step less (the agents would meet at the end of this phase in the predecessor of p_{2j} in the path $P(p_{2j-1}, p_{2j}\rangle$) and this local gain could reduce in some cases the overall cost of the search. Moreover, the worst case in the analysis holds for the tree in Figure 5.6. We can observe that, while the exploration scheme produced by the algorithm has cost 10, the optimal exploration scheme on such tree has cost 8 and not 7 as stated by the provided lower bound.

5.4 A $\frac{27}{8}$-approximation algorithm for arbitrary networks

We consider the following natural approach to the rBHS problem in an arbitrary graph G. First select a spanning tree in G and then explore the graph by traversing the tree edges. In [12], the authors hint to a similar approach. Their tree exploration algorithm generates exploration schemes where both agents traverse the tree together

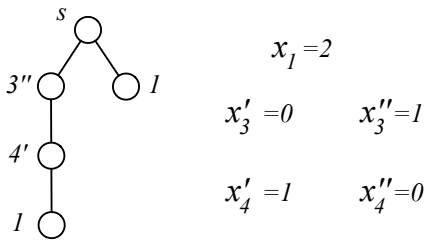

Figure 5.6: The tree yielding the worst case in the analysis of *Search-Tree* algorithm. The numbers represent the types of the nodes.

in, say, the depth-first order and explore each new node v with a two-step *probe phase*: one agent waits in the parent p of v while the other goes to v and back to p. Such schemes explore an n-node tree within $4(n-1)-2l$ steps, where l is the number of leaves in the tree. This approach guarantees an approximation ratio of 4, since any exploration of an n-node graph requires at least $n-1$ steps.
Here we want to use algorithm *SearchTree* presented in previous section, and exploit the bound given by formula (5.1. In Section 5.4.1 we present a heuristic algorithm *Generate-Tree*(G,s) for the problem of computing a rooted spanning tree T of graph G which gives a relatively small value of that formula.

Our *Spanning-Tree Exploration* (*STE*) algorithm returns, for a given graph G and a starting node s, the exploration scheme computed by *Search-Tree*(T_G,s), where T_G is the spanning tree computed by *Generate-Tree*(G,s). In Section 5.4.2 we show that the *STE* algorithm guarantees an approximation ratio of at most $\frac{27}{8}$. In Section 5.4.3 we remark on other possible variants of exploring graphs via spanning trees, and show the intrinsic limitations of such approach.
Note that if G is a path with s as an end node, then the optimal exploration scheme is obvious. Therefore we assume throughout this section that graph G is not of this form.

5.4.1 Generating a good spanning tree of a graph

We describe now our heuristic algorithm *Generate-Tree*(G,s) for computing a spanning tree T_G of a graph $G=(V,E)$ rooted at a node $s \in V$ whose objective is to minimize the formula (5.1). We believe that computing a rooted spanning tree which minimizes this formula is NP-hard, since the related problem of computing a spanning tree which maximizes the number of leaves is NP-hard [22]. In Section 5.4.2 we show that the exploration scheme constructed by algorithm *Search-Tree*(T_G,s) for the spanning tree T_G computed by algorithm *Generate-Tree*(G,s) has cost at most $\frac{27}{8}$ times worse than the cost of an optimal BHS for graph G. It is interesting to notice

that, if we were provided with an algorithm able to find a bushy spanning tree for any given graph, then we would have a BHS algorithm for arbitrary networks with an approximation ratio of 2. For a bushy tree we have $x_4 = 0$ and $x_3 \leq x_1 - 1$. Hence, the ratio between formula (5.1) and the optimal cost is bounded by:

$$\begin{aligned} \frac{x_1 + 3x_3 + odd(x_1 + x_3)}{x_1 + x_3 + odd(x_1 + x_3)} &= \\ \frac{x_1 + 3\alpha(x_1 - 1) + odd(x_1 + x_3)}{x_1 + \alpha(x_1 - 1) + odd(x_1 + x_3)} &= \\ \frac{x_1(1 + 3\alpha) - 3\alpha + odd(x_1 + x_3)}{x_1(1 + \alpha) - \alpha + odd(x_1 + x_3)} &\leq 2 \end{aligned}$$

since $\alpha \in [0, \ldots 1]$. Unfortunately, not every graph contains a bushy spanning tree, and moreover we are not able to answer to the following open question:

Open Question 2 *Does there exist a polynomial time algorithm able to find, given a graph G, a spanning tree T_G of G, which maximizes the number of "bushy" nodes, i.e., of internal nodes having degree at least 3?*

Our heuristics follows the same route. The minimization goal of Algorithm *Generate-Tree*(G, s) is pursued by trying to avoid creation of *type-4* nodes. More precisely, the algorithm grows in a greedy manner a spanning tree T, starting from node s, avoiding creation of internal nodes with only one child. A single child is a *type-4* node, unless it is a leaf. For the computation of the algorithm, let V_T denote always the set of nodes in the current tree T and let $\overline{V}_T = V \setminus V_T$ be the set of nodes not yet in T; initially $V_T = \{s\}$. With respect to tree T, each node in V is either an *internal node*, or a *leaf*; it is an *external node* if it belongs to the set $\overline{V}_T$. An *external neighbor* of a node $u \in V$ is a neighbor of u in graph G which belongs to $\overline{V}_T$.

The pseudocode of algorithm *Generate-Tree* is given below. The algorithm consists of two parts. During part 1, the algorithm iteratively extends the current tree T rooted at s for as long as there is an *expandable leaf* in T or there is an *expandable external node* in $\overline{V}_T$. An *expandable leaf* in tree T is a leaf which has at least two external neighbors. An *expandable external node* (with respect to T) is a node in $\overline{V}_T$ which has at least one neighbor in T and at least two external neighbors, or has at least three external neighbors. The loop in part 1 of the algorithm maintains the following invariant: for the current tree T, there is no edge in G between an internal node and an external node. That is, each edge in G between the sets V_T and $\overline{V}_T$ is adjacent to a leaf of T.

If there is an expandable leaf in tree T, then extend T by selecting an arbitrary expandable leaf u and attaching to it all its external neighbors (see the left diagram in Figure 5.7). If there is no expandable leaf in T but there is an expandable external node, then we extend T in the following way. Let $P = (u_1, u_2, \ldots, u_k)$ be a path

in G consisting of external nodes such that node u_1 is the only node in P adjacent to T and node u_k is the only expandable external node in P. Let u_0 be a node in T adjacent to u_1 and let $w_1, w_2, \ldots, w_k$ be the neighbors of u_k which are neither in T nor on P. According to the invariant of the loop, node u_0 must be a leaf in tree T. We extend tree T by attaching path P to node u_0 and nodes $w_1, w_2, \ldots, w_k$ as children of u_k. Path P, as a part of the new extended tree and a part of the final tree T_G, is called a *mid-tree path*. The middle diagram in Figure 5.7 illustrates the expansion of the tree using mid-tree paths.

Let T_1 denote the tree T at the end of part 1 of the algorithm. Since no expandable external node is left, each connected component of the subgraph of graph G induced by the set of external nodes must be now a path. Moreover, for each such path P, no node of P other than an end node is adjacent to T_1 (or otherwise such a node would be an expandable external node) but at least one end node of P is adjacent to tree T_1 (since G is connected). Let $\mathcal{P}$ denote the collection of these paths. If a path $P \in \mathcal{P}$ has at least two nodes and both end nodes are adjacent to T_1, then we replace P in $\mathcal{P}$ with paths P' and P'' obtained from P by removing the middle edge (or any of the two middle edges, if P has an odd number of nodes). Now for each path $P = (w_1, w_2, \ldots, w_k) \in \mathcal{P}$ where w_1 is adjacent to T_1 (exactly one end node of P is adjacent to T_1), we extend T by attaching P to a neighbor of w_1 in T_1, which must be a leaf in T_1 (see the last diagram in Figure 5.7). If path P has at least two nodes, then we call this path without the last node w_k a *leaf path*. When all paths from $\mathcal{P}$ are attached to tree T, tree T becomes a spanning tree T_G of G, and this tree is returned by the algorithm.

The whole algorithm *Generate-Tree* can be easily implemented to run in polynomial time, and it actually can be implemented to run in linear time on the number of nodes in G. An example of a spanning tree produced by the algorithm is given in Figure 5.8. The next two lemmas summarize the properties of the algorithm which are important in our analysis.

Lemma 5.4.1 *Consider any iteration of the loop in part 1 of algorithm Generate-Tree(G, s), and the current tree T at the beginning of this iteration. The following two properties hold.*

1. *No internal node of T is adjacent in G to any external node.*
2. *Each leaf in T has a sibling, unless this is the first iteration of the loop (when T contains only the root s).*

Proof. At the beginning of the first iteration of the loop, tree T does not have any internal node, so both Statements 1 and 2 are obviously true. Let T' be the tree T at the beginning of one iteration of the loop other than the last one, and let T'' be the tree T at the beginning of the next iteration. Assume inductively that Statements 1 and 2 are true for tree T'. Tree T'' is obtained from tree T' by adding children to an

Algorithm 1: Algorithm Generate-Tree (G, s)

$V \leftarrow$ set of nodes in G; $E \leftarrow$ set of edges in G;
$T \leftarrow \emptyset$; {the edges of the current tree}
let V_T denote the set of nodes in T (initially $V_T = \{s\}$), and let $\overline{V}_T = V \setminus V_T$;
{*Part 1: grow T until there is no expandable leaf or expandable external node.*}
loop
  if there exists an expandable leaf in T **then**
    $u \leftarrow$ an expandable leaf in T;
    $W \leftarrow$ the set of neighbors of u in $\overline{V}_T$;
    $T \leftarrow T \cup \{(u, w) : w \in W\}$;
  else if there exists an expandable external node in $\overline{V}_T$ **then**
    $P = (u_1, \ldots, u_k) \leftarrow$ a path in G such that each $u_i \in \overline{V}_T$, u_1 is the only node on P adjacent to T and u_k is the only expandable external node on P;
    $u_0 \leftarrow$ a leaf in T adjacent to u_1;
    $W \leftarrow$ the set of neighbors of u_k which are neither in T nor on P;
    $T \leftarrow T \cup \{(u_0, u_1)\} \cup P \cup \{(u_k, w) : w \in W\}$;
    { P is a *mid-tree path* in T }
  else
    exit the loop;
  end if
end loop
{*Part 2: attach to T the remaining paths.*}
$T_1 \leftarrow T$;
$\mathcal{P} \leftarrow$ the set of connected components (paths) in the subgraph induced by $\overline{V}_T$;
for all $P = (u_1, u_2, \ldots, u_j) \in \mathcal{P}$, where $j \geq 2$ and u_1 and u_j adjacent to T_1 **do**
  let $P' = (u_1, \ldots, u_k)$ and $P'' = (u_{k+1}, \ldots, u_j)$, where $k = \lfloor j/2 \rfloor$;
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{P\} \cup \{P', P''\}$;
end for
for all $P = (w) \in \mathcal{P}$ **do**
  $u \leftarrow$ a leaf in T_1 adjacent to w; $T \leftarrow T \cup \{(u, w)\}$;
end for
for all $P = (u_1, u_2, \ldots, u_k) \in \mathcal{P}$, where $k \geq 2$ and u_1 adjacent to T_1 **do**
  $u_0 \leftarrow$ a leaf in T_1 adjacent to u_1; $T \leftarrow T \cup \{(u_0, u_1)\} \cup P$;
  { path $(u_1, u_2, \ldots, u_{k-1})$ is a *leaf path* in T }
end for
return T.

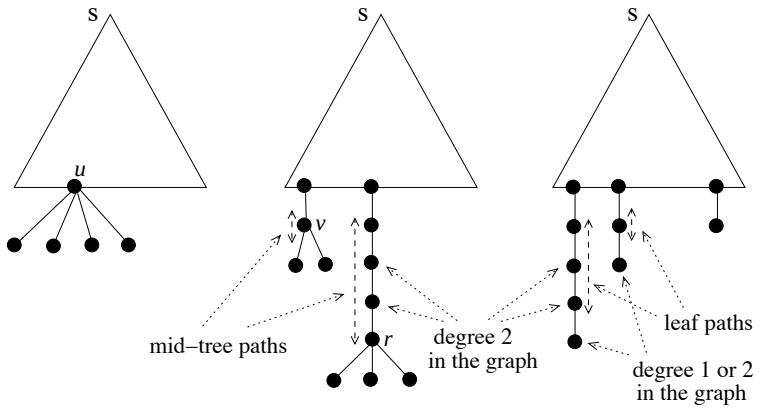

Figure 5.7: Expansion of the tree during the computation of algorithm *Generate-Tree*: in part 1 of the algorithm using an expandable leaf u (the left diagram) and using mid-tree paths to expandable external nodes v and r (the middle diagram); and in part 2 of the algorithm (the right diagram).

expandable leaf (lines 7–9 in the pseudocode) or, if T' does not have an expandable leaf, by adding a mid-tree path and children of the last node on this path (lines 11–14).

Consider the first case: tree T'' is obtained from T' by adding children to an expandable leaf u. Node u is the only new internal node in T'' and its children are the only new leaves. All neighbors of node u are now in T'', so Statement 1 is true for T''. Node u gets at least two children since u is an expandable leaf in tree T', so also Statement 2 is true for T''.

Consider now the second case: tree T' does not have an expandable leaf and tree T'' is obtained from tree T' by attaching a mid-tree path $P = (u_1, \ldots, u_k)$ to a leaf u_0 and attaching all remaining neighbors of u_k (the neighbors neither in tree T' nor in path P) as children of u_k. We check first that the new internal nodes $u_0, u_1, \ldots, u_k$ in tree T'' have all their neighbors in T''. Clearly node u_k has all its neighbors in tree T''. Node u_0 cannot have neighbors outside of T' other than node u_1 since node u_0 is not an expandable leaf in T'. If $k \geq 2$, then node u_1 cannot have neighbors outside T' other than u_2 since u_1 is not an expandable external node. If $k \geq 3$, then for each $i = 2, \ldots, k-1$, node u_i is not adjacent to T' and is not an expandable external node, so nodes u_{i-1} and u_{i+1} can be its only neighbors in graph G. Thus each new internal node in T'' has all its neighbors in T'', so Statement 1 holds for T''.

The new leaves in T'' are the children of u_k. Since u_k is an expandable external node (with respect to T'), it gets at least two children in T''. Indeed, if $k = 1$, then, by definition of expandable external node, node u_1 must have at least two external neighbors, which become its children in T''. If $k \geq 2$, then node u_k is not adjacent to tree T', so it must have at least 3 external neighbors. One of them is node u_{k-1} while

the remaining ones are the children of u_k in T''. Thus Statement 2 holds for T''. □

Lemma 5.4.2 *Let T_1 denote the tree T at the end of part 1 of Algorithm Generate-Tree(G, s) and let $\mathcal{P}$ denote the set of connected components of the subgraph G' of graph G induced by the external nodes (with respect to T_1).*

1. *For each connected component of subgraph G', the edges of this component form a (simple) path.*
2. *For each path $P \in \mathcal{P}$,*
   (a) *the internal nodes of P are not adjacent to tree T_1;*
   (b) *at least one end node of P is adjacent to tree T_1.*

Proof. There is no expandable external node with respect to T_1. Thus each node in subgraph G' has degree at most 2 in G', since otherwise such a node would be an expandable external node. Therefore each connected component of G' is either a path (possibly a single node) or a cycle. However, if a connected component of G' were a cycle, then there would be a node on this cycle adjacent to tree T_1, since graph G is connected, and this node would be an expandable external node.

For a path P which is a connected component of subgraph G', if a node in P other than an end node were adjacent to tree T_1, then this node would be an expandable external node. Since graph G is connected, at least one end node of P must be adjacent to tree T_1. □

We look now at the *type-4* nodes in T_G to see how they were created and what their properties in graph G are. We view the mid-tree paths and the leaf paths in T_G in the direction from the root toward the leaves. That is, the first node on such a path is the node closest to the root.

Lemma 5.4.3 *A node in tree T_G is a* type-4 *node if and only if it belongs to a mid-tree path or a leaf path.*

Proof. We have to examine all the possible extensions of the current tree T to a new tree T' during the computation of algorithm *Generate-Tree*.

In line 9 of the algorithm, node u changes its status from *type-1* in tree T to *type-3* in tree T' (Property 2 in Lemma 5.4.1 implies that u has a sibling in tree T) and all new nodes in tree T' are *type-1* nodes. In line 14, node u_0 changes its status from *type-1* in tree T to *type-3* in tree T', the new nodes $u_1, u_2, \ldots, u_k$, which form a mid-tree path, are *type-4* nodes in tree T', and the leaves attached to u_k are *type-1* nodes in tree T'. In line 28, node u changes its status from *type-1* in tree T to *type-3* in tree T' (Property 2 of Lemma 5.4.1 implies that u has a sibling in the tree T_1 constructed during the first part of the algorithm) and the new node w is a *type-1* node in tree T'. In line 31, node u_0 changes its status from *type-1* in tree T to *type-3* in tree T', the

new nodes $u_1, u_2, \ldots, u_{k-1}$, which form a leaf path, are *type-4* nodes in tree T', and the leaf u_k attached to u_{k-1} is a *type-1* node in tree T'.

Thus a node in the final tree T_G is a *type-4* node if and only if this node has been added to the growing tree as a part of a mid-tree path or a leaf path. □

Lemma 5.4.4 *Each node on a mid-tree path in tree T_G other than the first node and the last node has degree 2 in G.*

Proof. Let T be the tree during the computation of algorithm *Generate-Tree* when a mid-tree path $P = (u_1, u_2, \ldots, u_k)$ is selected in line 11. For each $i = 2, 3, \ldots, k-1$, node u_i is a non-expandable external node with two external neighbors u_{i-1} and u_{i+1}, so by definition of the expandable external nodes, node u_i is not adjacent to any node in T and nodes u_{i-1} and u_{i+1} must be its only external neighbors. □

Lemma 5.4.5 *Let $(u_1, \ldots, u_{k-1})$ be a leaf path in tree T_G, and let u_k be the leaf in T_G attached to u_{k-1}. Then the following properties hold.*

1. *Each node $u_2, u_3, \ldots, u_{k-1}$ has degree 2 in G.*

2. *Node u_k has degree at most 2 in G.*

3. *If node u_k has degree 2 in G and the length of the leaf path is at least 2 ($k \geq 3$), then both neighbors of u_k in G have degree 2.*

Proof. Let T_1 be the tree constructed in the first part of the algorithm, and let $P = (u_1, \ldots, u_{k-1}, u_k)$, $k \geq 2$, be one of the paths considered in lines 30–31. Path $(u_1, u_2, \ldots, u_{k-1})$ is a leaf path in the final tree T_G. There is no expandable external node with respect to tree T_1, so for each $i = 2, 3, \ldots, k-1$, node u_i is a non-expandable external node with two external neighbors u_{i-1} and u_{i+1}. The definition of the expandable external nodes implies that node u_i is not adjacent to any node in T_1 and nodes u_{i-1} and u_{i+1} must be its only external neighbors. Thus the degree of nodes u_i in G is 2.

Node u_k is a non-expandable external node, so it may be adjacent to at most one external node other than u_{k-1}. However, node u_k cannot be adjacent to T_1 because if it were, then path P would have been split into two paths in lines 24–25. Thus the degree of node u_k in G is at most 2.

If node u_k has degree 2 in G, then path P has been obtained by splitting a path $(u_1, \ldots, u_k, u_{k+1}, \ldots, u_j)$ of external nodes in lines 24–25, where $2k \leq j \leq 2k+1$. If $k \geq 3$, and hence $j \geq k+2$, neither of nodes u_{k-1} and u_{k+1} is adjacent to tree T_1 and, as non-expandable external nodes, they may have only two external neighbors each. Thus both u_{k-1} and u_{k+1} have degree 2 in G. □

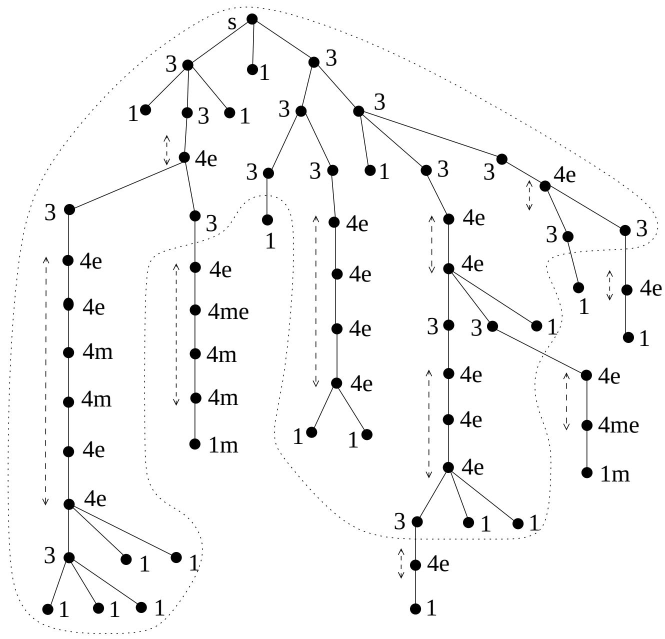

Figure 5.8: An example of spanning tree produced by Algorithm *Generate-Tree*. Each node of the tree (excluding the root) is labeled with the corresponding type. The part of the tree produced during Part 1 of the algorithm is enclosed in the dotted curve. Arrows denote mid-tree paths and leaf paths.

5.4.2 Approximation ratio of the *STE* Algorithm

Lemma 5.3.5 implies that the cost of the exploration scheme $\mathcal{E}_{G,s}$ computed by the *STE* algorithm for a graph G and a starting node s is

$$cost(\mathcal{E}_{G,s}) \leq x_1 + 3x_3 + 4x_4 + 1, \tag{5.2}$$

where x_t is the number of the *type-t* nodes in the tree T_G computed by algorithm *Generate-Tree*(G, s). The cost of the optimal exploration scheme is at least $n - 1 = x_1 + x_3 + x_4$, so any upper bound on x_4 in a form of a linear function of x_1 and x_3 would give immediately an upper bound on the approximation ratio of algorithm *STE* as a constant less than 4. However this simple approach cannot work by itself since the ratio $x_4/(x_1 + x_3)$ can be arbitrarily large not only for tree T_G, but for the best possible spanning tree as well. For example, if graph G is a path, then in its unique spanning tree all nodes except node s, its two neighbors and the end points of the path are *type-4* nodes.

Our analysis, which examines closer the *type-4* nodes in tree T_G, can be viewed as consisting of the following three steps. We first identify some nodes in graph G

which "slow down" the optimal BHS in graph G so that its cost must be greater than the ideal $n-1$ (Lemma 5.4.6). We then show which *type-4* nodes in T_G must be among those "slowing down" nodes (Lemma 5.4.7). Finally we give a bound on the number of the other *type-4* nodes as a linear function of x_1 and x_3 (Lemma 5.4.8).

A node in graph G is a *type-d* node if its degree is at most 2 and the degrees of its neighbors are also at most 2.

Lemma 5.4.6 *The cost of an optimal exploration scheme $\mathcal{E}^*_{G,s}$ for (G,s) is*

$$\text{cost}(\mathcal{E}^*_{G,s}) \geq n - 1 + \frac{1}{2}x_d \tag{5.3}$$

Proof. Informally, no BHS can explore *type-d* nodes at the average rate of one node per one step, requiring at least one additional step per two *type-d* nodes. Formally, consider any BHS and the case when there is no black hole. Each phase of the search when a *type-d* node v and another node u (which may be also a *type-d* node) are explored must consist of at least 3 steps. To see this, check that the distance from either v or u (or both) to the meeting point at the end of this phase must be at least 2. Thus:

1. there are at least $(n-1)/2 + \alpha$ phases in total, where $\alpha \geq 0$ is the number of phases when only one node is explored, and each phase consists of at least 2 steps;

2. there are at least $(x_d - \alpha)/2$ phases when a *type-d* node is explored together with another node, and each of these phases consists of at least 3 steps.

Hence the total number of steps is at least $n - 1 + 2\alpha + (x_d - \alpha)/2 \geq n - 1 + x_d/2$. □

Lemma 5.4.3 says that the *type-4* nodes in tree T_G are the nodes on the mid-tree paths and the leaf paths. We further categorize these nodes in the following way. A *type-4e* node is a node which is one of the first two or the last two nodes of a mid-tree path or the first node of a leaf path. A *type-4me* node is the second node of a leaf path. All other nodes on the mid-tree paths and the leaf paths are *type-4m* nodes. We also introduce *type-1m* for the leaves attached to the leaf paths having length at least 2 (see the example in Figure 5.8). These definitions and Lemmas 5.4.4 and 5.4.5 immediately imply the following lemma.

Lemma 5.4.7 *Each* type-4m *or* type-1m *node in tree T_G is a* type-d *node in G.*

The next lemma gives bounds on the number of *type-4e* and *type-4me* nodes in tree T_G.

Lemma 5.4.8 *The number of* type-4e *nodes and the number of* type-4me *nodes in tree T_G satisfy the following relations.*

$$x_{4e} \leq 3x_1 + x_3 - 2, \tag{5.4}$$

$$x_{4me} = x_{1m}. \tag{5.5}$$

Proof. The fact that there are exactly as many *type-4me* nodes as *type-1m* nodes follows immediately from the definitions of these types. To show that Inequality (5.4) holds, denote by z' and z'' the number of the mid-tree paths and the number of the leaf paths in T_G, respectively. The definition of *type-4e* nodes imply that

$$x_{4e} \leq 4z' + z''. \tag{5.6}$$

The last node of a mid-tree path is a branching node in tree T_G (a node with at least two children) so $z' \leq x_1 - 1$ since T_G has at most $x_1 - 1$ branching nodes. We also have $z' + z'' \leq x_3 + 1$ since the parents of the first nodes of mid-tree paths and leaf paths must be distinct and each of them is either a *type-3* node or the root. Thus

$$4z' \leq 3(x_1 - 1) + x_3 + 1 - z'', \tag{5.7}$$

and Inequalities (5.6) and (5.7) give Inequality (5.4). □

We can now state our final theorem.

Theorem 5.4.9 *For any graph G and any starting node s, the ratio of the cost of an exploration scheme $\mathcal{E}_{G,s}$ computed by the STE algorithm to the cost of an optimal exploration scheme $\mathcal{E}^*_{G,s}$ is at most $\frac{27}{8}$.*

Proof. Starting from the bounds (5.2) and (5.3), we have

$$\begin{aligned}
&\frac{27}{8}\mathit{cost}(\mathcal{E}^*_{G,s}) - \mathit{cost}(\mathcal{E}_{G,s}) \geq \\
&\quad \geq \frac{27}{8}(n - 1 + \frac{1}{2}x_d) - (x_1 + 3x_3 + 4x_4 + 1) && (5.8)\\
&\quad \geq \frac{27}{8}(x_1 + x_3 + x_{4e} + x_{4me} + \frac{3}{2}x_{4m} + \frac{1}{2}x_{1m}) && (5.9)\\
&\qquad -(x_1 + 3x_3 + 4x_{4e} + 4x_{4me} + 4x_{4m} + 1) \\
&\quad = \frac{19}{8}x_1 + \frac{3}{8}x_3 - \frac{5}{8}(x_{4e} + x_{4me}) + \frac{17}{16}x_{4m} + \frac{27}{16}x_{1m} - 1 \\
&\quad \geq \frac{19}{8}x_1 + \frac{3}{8}x_3 - \frac{5}{8}(3x_1 + x_3 - 2 + x_{1m}) + \frac{17}{16}x_{4m} + \frac{27}{16}x_{1m} - 1 && (5.10)\\
&\quad = \frac{1}{4}(2x_1 - x_3) + \frac{17}{16}(x_{4m} + x_{1m}) + \frac{1}{4} \geq 0. && (5.11)
\end{aligned}$$

Inequality (5.8) follows from (5.2) and (5.3), Inequality (5.9) follows from Lemma 5.4.7, and Inequality (5.10) follows from (5.4) and (5.5). Finally the inequality in line (5.11)

holds because $x_3 \leq 2x_1 - 1$. To see that this is a valid bound on x_3, we can bound separately the number of *type-3* nodes which have only one descendant leaf and the number of the other *type-3* nodes. The number of *type-3* nodes which have only one descendant leaf is at most x_1, the number of leaves. Each *type-3* node which has at least 2 descendant leaves is either a branching node in T_G, or is the parent of the first node of a mid-tree path and the last node of this path is a branching node. Thus the number of *type-3* nodes which have at least 2 descendant leaves is at most the number of branching nodes in T_G, which is at most $x_1 - 1$. □

5.4.3 Additional comments on exploring a graph via a spanning tree

The approximation algorithm for the rBHS problem in arbitrary graphs which we presented in the previous section is based on the following two-part approach.

1. Find a suitable spanning tree T_G of the input graph G.
2. Using an algorithm for constructing exploration schemes for trees, construct an exploration scheme for T_G, and take it as an exploration scheme for G.

In this section we further investigate advantages and limits of this technique. We already mentioned in the beginning of Section 5.4 the simple 4-approximation algorithm for the rBHS problem given in [12]. One can also obtain a c-approximation algorithm for a constant $c < 4$ using other ways of selecting a spanning tree than our algorithm *Generate-Tree*(G, s). In a preliminary paper ([29]) we actually gave a different version of Algorithm *Generate-Tree*, which was based on greedily selecting a maximal forest of bushy trees and then connecting the trees into one spanning tree. However, we could only show that that method led to an approximation ratio of $\frac{7}{2}$.

Another possible good candidate is a spanning tree T which "locally" maximizes the number of leaves. The tree is k-local maximum in the sense that no exchange of at most k tree edges for non-tree edges, for some constant k, can give a new spanning tree with more leaves than in T. Such a "locally maximized" spanning tree can be computed in polynomial time starting from any spanning tree. One can show that locally maximized spanning trees for $k = 2$, together with our *Search-Tree* algorithm, give an algorithm for the rBHS problem with an approximation ratio of $\frac{43}{12} > \frac{7}{2}$.

We would like to mention that the straightforward algorithm for searching a tree outlined in the first paragraph of Section 5.4, together with good spanning-tree selection algorithms, can also give approximation algorithms with ratios less than 4, but greater than the approximation ratios which can be obtained using the *Search-Tree* algorithm. For example, the straightforward tree-searching algorithm gives approximation ratios of $\frac{29}{8}$ and $\frac{23}{6}$ for the rBHS problem, if used together with the spanning trees computed by our *Generate-Tree* algorithm, and the locally maximized spanning trees, respectively.

Even though the spanning tree based approach seems very natural (and we found indeed difficult to analyze more general approaches), we can show that no graph exploration using this technique can guarantee a better approximation ratio than $3/2$, even if we restrict the analysis to planar graphs.

Let $G_c = (V, E)$ be an even-length cycle of $c + 1$ nodes $v_0, v_1, v_2, \ldots, v_c$ and edges $(v_0, v_1), (v_1, v_2) \ldots, (v_{c-1}, v_c), (v_c, v_0)$. A new graph G'_c is obtained from G_c by using a similar construction as the one for the NP-hardness proof given in Section 5.1. The starting node s is assumed to be v_0. Then, a pair of twin nodes is added to each edge (v_i, v_{i+1}) and to (v_c, v_0); a flag node is appended to each node v_i, for $i = 0, \ldots, c - 1$. We can easily find a planar embedding of the obtained graph, in which the two corresponding twin nodes are on the two faces delimited by cycle $\langle s, v_1, v_2, \ldots, v_c\rangle$. Let us call as f' and f'' the inner and the outer of such faces, respectively. Unlike construction in Section 5.1, here the shortcut edges alternate between the two faces. Formally, for each $i = 0, 2, 4, \ldots, c - 1$, we add an edge between twin node $z_{f'}^{(v_i,v_{i+1})}$ and twin node $z_{f'}^{(v_{i+1},v_{i+2})}$ and an edge between twin node $z_{f''}^{(v_{i+1},v_{i+2})}$ and twin node $z_{f''}^{(v_{i+2},v_{i+3})}$, where indexes of v are computed $(\mathrm{mod}\ c + 1)$. An example of graph G'_c, for $c = 7$, is shown in Figure 5.9. Graph G'_c has $4c+3$ nodes and by modifying appropriately the exploration scheme given in the proof of Lemma 5.1.4, one can show that the cost of an optimal exploration scheme for G'_c is $4c + 2$.

Consider the spanning tree of G'_c as shown in Figure 5.9. In the terminology and notation from Section 5.3, this tree has $x_3 = c - 1$ *type-3* nodes (the nodes $v_1, v_2, \ldots, v_{c-1}$) and $x_1 = 3c + 3$ *type-1* nodes. Lemma 5.3.4 implies that the cost of the exploration scheme computed for this tree by algorithm *Search-Tree* given in Section 5.3 is exactly $x_1 + 3x_3 = 6c$. We show below that the cost of any exploration scheme for any spanning tree of G'_c is at least $6c - 2$, so at least $3/2 - O(1/n)$ times higher than the optimal cost.

We recall Lemma 5.3.6 and the partitioning of nodes defined there.

Lemma 5.4.10 *For any spanning tree T of G'_c rooted at s, $x'_3 + x'_4 + 2(x''_3 + x''_4) \geq 2c - 4$.*

Proof. All nodes in $V \setminus \{v_c\} = \{v_1, v_2, \ldots, v_{c-1}\}$ must be internal nodes in T since they have to be parents of their flag nodes. Let z be the total number of *type-3'* and *type-4'* nodes in $V \setminus \{v_c\}$. The remaining $c-1-z$ nodes in $V \setminus \{v_c\}$ are either of type $3''$ or of type $4''$. Let v_i and v_j be two nodes in cycle G_c, such that $i+2 \leq j \leq c-1$. If the shortcut edges bypassing them are not in T, then at most one of them can have only one descendant (*type-3'* or *type-4'*). To see this, observe that a path from s to a node v_k, $i < k < j$, must pass through one of the nodes v_i and v_j or through one of their shortcut edges. This means that if neither of the shortcut edges bypassing nodes v_i and v_j is in T, then either v_i or v_j is an ancestor of at least two nodes (one is v_k and the other is the flag node). Therefore, at least $z - 2$ shortcut edges belong to T.

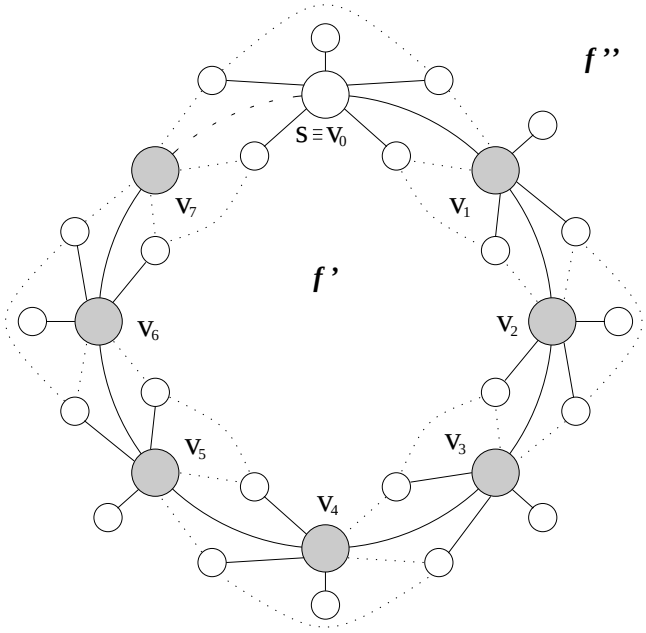

Figure 5.9: Graph G'_7 and its "good" spanning tree (solid edges).

Note that, for each of these edges, at least one of the endpoints is an internal node. Hence, we have

$$x'_3 + x'_4 + 2(x''_3 + x''_4) \geq z + 2(c - 1 - z) + z - 2 = 2c - 4.$$

□

Lemmas 5.3.6 and 5.4.10 imply that the cost of any exploration scheme for any spanning tree of G'_c is at least $6c - 2$. The theorem follows straightforwardly.

Theorem 5.4.11 *There exists an infinite class of planar graphs $\mathcal{G}$, for which any spanning tree based exploration scheme on $G \in \mathcal{G}$ cannot achieve a better approximation ratio than $\frac{3}{2} - O\left(\frac{1}{n}\right)$*

5.5 Network topologies that facilitate black hole search

In this section we show how better approximation ratios can be obtained if the graph has some particular properties.
For example, it is shown in [30] that any n-node graph with the minimum node degree at least 3 has a spanning tree with at least $n/4 + 2$ leaves, and a polynomial-time algorithm for computing such a spanning tree is given. This gives a c-approximation

algorithm for the rBHS problem for such graphs, where c is $\frac{7}{2}$, if the straightforward tree-searching algorithm is used, or $\frac{13}{4}$, if algorithm *Search-Tree* is used. It is also shown in [30] that for graphs with the minimum degree at least k one can compute in polynomial time spanning trees with at least $(1 - O((\log k)/k))n$ leaves. This gives a $(1 + O((\log k)/k))$-approximation algorithm for the rBHS problem for this class of graphs.

In the following subsection we provide an optimal algorithm suited for the special case where the network is a ring.

5.5.1 Optimal exploration schemes for ring networks

In [12] it is presented an optimal algorithm for the rBHS problem in networks whose topology is a line. Here we extend such result to ring networks. The approach is rather similar; however we can exploit here the fact that there are always two distinct paths from any node to the starting point.

We are given a cycle G_c of c nodes $\langle v_0, v_1, \ldots, v_{c-1}\rangle$ (where c is sufficiently large), and we assume, without loss of generality, that the starting point is v_0. We provide an optimal exploration scheme $\mathcal{E}_{G_c,v_0}$ for such cycle by explicitly finding, among all the possible exploration schemes, the one yielding the lower bound on the cost of any exploration scheme for G_c.
First we give some preliminary definitions.

Definition 5.5.1 *Given a graph $G = (V, E)$ and an exploration scheme $\mathcal{E}_{G,s}$, a node v is called a* LIMIT *of an explored territory S_i if $v \in S_i$ and there exists a node u adjacent to v in G such that $u \in V \setminus S_i$.*

Remark 5.5.1 *The explored territory of any exploration scheme $\mathcal{E}_{G,s}$ is always a connected set. Therefore, in ring networks each step of $\mathcal{E}_{G,s}$ has two limit nodes, excluding the steps of the first phase (in which the only limit is v_0), and the last step (in which there are no limit nodes).*

Now we define three basic exploration strategies, then we will show that we can always express an optimal exploration scheme only by means of them.
Here we extend the notation $P(\cdot,\cdot\rangle$ introduced in Section 5.3. For nodes v_i and v_j in the ring G_c, $P(v_i, v_j\rangle$ is the sequence of the nodes on the shortest path composed of only safe nodes, from v_i to v_j excluding the first node v_i. We can observe that, excluding the last phase, there is always a single safe path between nodes v_i and v_j in G_c. Such path has length $(i - j) \mod c$ (if $i > j$) or $(j - i) \mod c$ (if $j > i$).
We assume that v_i and v_j are the current limits of the explored territory (with $i < j$ or, possibly, $i = j = 0$ for the first phase), and that v_m is the meeting point of the previous phase. The phases denoted as *f-probe*, *b-probe* and *c-split* are defined in the following way:

f-probe (or *forward probe*): the explored territory is expanded in the clockwise direction:

$$\begin{aligned}\phi^1 &= P(v_m, v_i\rangle \circ \langle v_{i+1}, v_i\rangle \\ \phi^2 &= P(v_m, v_i\rangle \circ \langle v_i, v_i\rangle.\end{aligned}$$

The meeting point at the end of the phase is node v_i. The length of the phase is $(i-m) \mod c + 2$.

b-probe (or *backward probe*): the explored territory is expanded in the counterclockwise direction:

$$\begin{aligned}\phi^1 &= P(v_m, v_j\rangle \circ \langle v_{j-1}, v_j\rangle \\ \phi^2 &= P(v_m, v_j\rangle \circ \langle v_j, v_j\rangle.\end{aligned}$$

The meeting point at the end of the phase is node v_j. The length of the phase is $(m-j) \mod c + 2$.

c-split : the explored territory is expanded in both directions:

$$\begin{aligned}\phi^1 &= P(v_m, v_{i+1}\rangle \circ P(v_{i+1}, v_{m'}\rangle \\ \phi^2 &= P(v_m, v_{j-1}\rangle \circ P(v_{j-1}, v_{m'}\rangle;\end{aligned}$$

the meeting point $v_{m'}$ is such that:

$$\left(i - m + i - m'\right) \mod c = \left(m - j + m' - j\right) \mod c$$

and hence $m' = (j + i - m) \mod c$.
The length of such phase is $(i-j) \mod c + 2$.

Definition 5.5.2 *An exploration scheme $\mathcal{E}_{G_c,v_0}$ for a ring network G_C is called* PROPER, *if and only if the first phase is a* c-split *and each of the remaining phases, excluding possibly the last one, is either a* c-split *or a* f-probe *or a* b-probe.

Our next target is to show that any (optimal) exploration scheme on a ring, can be easily transformed into a proper one, having at most the same cost. This allows us to narrow the range of candidates for the lower bound on the cost of any exploration scheme. We first need a preliminary lemma.

Lemma 5.5.1 *Let (G_c, v_0) be an instance of the* rBHS problem, *and let $\mathcal{E}'_{G_c,v_0}$ and $\mathcal{E}''_{G_c,v_0}$ be two solutions of equal length. If, for every meeting step i of $\mathcal{E}'_{G_c,v_0}$, $S'_i \subseteq S''_i$ and $x'_i = x''_i = y'_i = y''_i$ then* cost$(\mathcal{E}''_{G_c,v_0}) \leq$ cost$(\mathcal{E}'_{G_c,v_0})$.

Proof. Indeed, we prove a stronger property, that is, for any possible value of B, the execution time of $\mathcal{E}''_{G_c,v_0}$ is not larger than the execution time of $\mathcal{E}'_{G_c,v_0}$. It is easy to observe that, since the two exploration schemes have the same length and end in the same node, their execution time for the case $B = \emptyset$ is exactly the same. This holds also for the case in which the black hole is one of the nodes explored during the last phase. Suppose, by contradiction, that there exists a node v_b in G_c such that, for the case $B = \{v_b\}$, the execution time of $\mathcal{E}''_{G_c,v_0}$ is larger than the execution time of $\mathcal{E}'_{G_c,v_0}$. Let i be the step in which v_b becomes explored in $\mathcal{E}'_{G_c,v_0}$ and let j be the step in which v_b becomes explored in $\mathcal{E}''_{G_c,v_0}$. We are thus supposing that:

$$i + d(x'_i, v_0) < j + d(x''_j, v_0),$$

where distances are computed in the subgraph of G_c induced by $V \setminus \{v_b\}$. By hypothesis, $v_b \in S''_i$ and $x''_i = y''_i$, hence $j \leq i$. The contradiction follows by observing that $d(x''_j, v_0) \leq d(x''_j, x''_i) + d(x''_i, v_0) \leq i - j + d(x''_i, v_0) = i - j + d(x'_i, v_0)$, by hypothesis.

Lemma 5.5.2 *Let $\mathcal{E}_{G_c,v_0}$ be an exploration scheme on a ring network G_c such that the first phase is not a* c-split*. Then, we can replace such phase with a* c-split *without increasing the cost of the exploration scheme.*

Proof. In the first phase of $\mathcal{E}_{G_c,v_0}$ either v_1 or v_c or both may be explored. Assume that the first meeting step is i (with $i \geq 2$). By Constraint 4, $x_i = y_i = v_0$. We replace the first i steps of $\mathcal{E}_{G_c,v_0}$ with a *c-split* followed by $i - 2$ steps in which both agents are in node v_0. Observe that, for each meeting step in $\mathcal{E}_{G_c,v_0}$, the explored territory is not decreased and both agents are in the same node as before, hence, by Lemma 5.5.1, the cost does not increase. □

Lemma 5.5.3 *Let $\mathcal{E}_{G_c,v_0}$ be any exploration scheme for a ring network G_c. We can transform $\mathcal{E}_{G_c,v_0}$ into a proper exploration scheme $\mathcal{E}'_{G_c,v_0}$ such that* cost$(\mathcal{E}'_{G_c,v_0}) \leq$ cost$(\mathcal{E}_{G_c,v_0})$.

Proof. Consider any phase of $\mathcal{E}_{G_c,v_0}$, excluding the first one (which, by Lemma 5.5.2, can be easily transformed into a *c-split*) and the last one. In each of these phases, the explored territory may increase by one node in clockwise direction, in counterclockwise direction or in both directions. We denote such phases as *1f-phase*, *1b-phase* and *2-phase* respectively. We show how each *1f-phase*, *1b-phase*, *2-phase* can be transformed respectively into a *f-probe*, *b-probe*, *c-split*. Assume that the meeting point before the phase is node v_m.
(*1f-phase*). Without loss of generality (in light of Lemma 3.5.3) we assume that *Agent*-1 performs the exploration, say of node v_{i+1}. Assume that the meeting point at the end of the phase is node $v_{m'}$. Therefore the sequence followed by *Agent*-1 is:

$$\phi^1 = P(v_m, v_{i+1}\rangle \circ P(v_{i+1}, v_{m'}\rangle \tag{5.12}$$

Note that if the sequence is longer, it can be replaced by sequence in 5.12, without increasing the cost of the exploration. We can hence replace the sequence followed by *Agent*-2 with the following sequence:

$$\phi^2 = P(v_m, v_i\rangle \circ \langle v_i, v_i\rangle \circ P(v_i, v_{m'}\rangle$$

It is easy to check that the explored territory remains the same as before. However, now the agents meet in node v_i after $(i-m) \mod c + 2$ steps; the phase has been transformed into a *f-probe*.
(*1b-phase*). The proof is symmetrical with respect to the *1f-phase*. We assume again that *Agent*-1 performs the exploration of node v_{j-1}. Assume that the meeting point at the end of the phase is node $v_{m'}$. Therefore the sequence followed by *Agent*-1 is:

$$\phi^1 = P(v_m, v_{j-1}\rangle \circ P(v_{j-1}, v_{m'}\rangle$$

We again replace the sequence followed by *Agent*-2 with the following sequence:

$$\phi^2 = P(v_m, v_j\rangle \circ \langle v_j, v_j\rangle \circ P(v_j, v_{m'}\rangle$$

It is easy to check that the explored territory remains the same as before. However, now the agents meet in node v_j after $(m-j) \mod c + 2$ steps; the phase has been transformed into a *b-probe*.
(*2-phase*). Again, by Lemma 3.5.3, we assume that *Agent*-1 explores node v_{i+1}, while *Agent*-2 explores node v_{j-1}, with $j > i+2$. Such phase has length at least:

$$\max[(i-m) \mod c + \left(i-m'\right) \mod c + 2, (m-j) \mod c + \left(m'-j\right) \mod c + 2]$$

We can simply observe that, if we replace this phase with a *c-split*, then the agents meet in a node $v_{m''}$ (where $m'' = (j+i-m) \mod c$) after $(i-j) \mod c + 2$ steps. If node $v_{m'}$ is the δ-th node following $v_{m''}$ in the clockwise direction ($m' = (m''+\delta) \mod c$) then the length of the original phase is at least:

$$\begin{aligned}(m-j) \mod c + \left(m'-j\right) \mod c + 2 &= \\ (m-j) \mod c + \left(m''-j\right) \mod c + \delta + 2 &= \\ (m-j) \mod c + (i-m) \mod c + \delta + 2 &= \\ (i-j) \mod c + \delta + 2.\end{aligned}$$

If, otherwise, node $v_{m'}$ is the δ-th node following $v_{m''}$ in the counterclockwise direction ($m' = (m''-\delta) \mod c$), then the length of the original phase is at least:

$$\begin{aligned}(i-m) \mod c + \left(i-m'\right) \mod c + 2 &= \\ (i-m) \mod c + \left(i-m''\right) \mod c + \delta + 2 &= \\ (i-m) \mod c + (m-j) \mod c + \delta + 2 &= \\ (i-j) \mod c + \delta + 2.\end{aligned}$$

Therefore, we can replace the *2-phase* with a *c-split* (ending in node m''), followed by a *walk* from node m'' to node m', without increasing the cost of the exploration scheme. □

By virtue of Lemma 5.5.3, we can narrow the set of candidates optimal, by considering only proper exploration schemes. Indeed, we can further restrict the structure of optimal exploration schemes.

Lemma 5.5.4 *Let $\mathcal{E}_{G_c,v_0}$ be a proper exploration scheme in which a* b-probe *follows a* f-probe *(or, vice-versa, a* f-probe *follows a* b-probe*). The exploration scheme $\mathcal{E}'_{G_c,v_0}$ obtained by replacing the two probes with a* c-split *(and possibly a walk) is such that* cost$(\mathcal{E}'_{G_c,v_0}) <$ cost$(\mathcal{E}_{G_c,v_0})$.

Proof. By symmetry, we consider only the case in which a *b-probe* follows a *f-probe*. Assume that, at the end of the phase before the *f-probe*, v_m is the meeting point and v_i and v_j are the two limit nodes ($i < j$). Since $\mathcal{E}_{G_c,v_0}$ is a proper exploration scheme, the total number of steps required by the two *probes* is $(i-m)_{\mod c} + 2 + (i-j)_{\mod c} + 2$ and the meeting point at the end of the two phases is node v_j. Suppose now that the two nodes are explored by a *c-split*. The length of such phase is $(i-j)_{\mod c} + 2$ and the meeting point at the end of the phase is node $m' = (j+i-m)_{\mod c}$. The two agents can reach node v_j as before, with further $(m'-j)_{\mod c} = (i-m)_{\mod c}$ steps. Therefore, the total length of the *c-split* is $(i-j)_{\mod c}+2+(i-m)_{\mod c}$, and hence it requires two steps less than exploring the same nodes by two *probes*. □

It should be obvious that in any optimal exploration scheme, the agents are not allowed to go back and forth on both sides of the explored territory when the explored territory becomes larger and larger (see [12] for a formal proof). Therefore, we can consider as candidate optimal, only the exploration schemes composed of the following parts:

i) a sequence of k' *c-split* phases (by Lemma 5.5.2, $k' \geq 1$);

ii) a sequence of k'' *f-probe* phases (or *b-probe*, by symmetry);

iii) one *c-split*;

iv) a sequence of k''' *b-probe* phases (or *f-probe*);

v) a final phase.

with the remark that parts *(iii)* and *(iv)* are performed only if $k''' > 0$.

It is easy to compute the length of parts *(i)-(iv)* above, we report the results in the following table, together with the limit nodes at the end of the part, and the last meeting point.

Part	Length	Limit nodes	Meeting point
(i)	$\sum_{i=1}^{k'} 2i = k'(k'+1)$	$\langle v_{k'}, v_{c-k'} \rangle$	v_0
(ii)	$(k'-1)+3k''$	$\langle v_{k'+k''}, v_{c-k'} \rangle$	$v_{k'+k''-1}$
(iii)	$2k'+k''+3$	$\langle v_{k'+k''+1}, v_{c-k'-1} \rangle$	$v_{c-k'+1}$
(iv)	$1+3k'''$	$\langle v_{k'+k''+1}, v_{c-k'-k'''-1} \rangle$	$v_{c-k'-k'''}$

The length of Part *(v)*, and thus the total execution time, depends on how many nodes are left to be explored and on whether k''' is positive or not. Therefore we distinguish four cases:

1. *1 node left and* $k''' = 0$. In this case parts *(iii)* and *(iv)* and are not in the exploration scheme; the last node to be explored is node $v_{k'+k''+1} = v_{c-k'-1}$. Hence $k'' = c - 2k' - 2$. The meeting point is node $v_{c-k'-3}$.
   In the last phase one agent (say *Agent*-1) goes to $v_{c-k'-1}$ and then to v_0 by following the shortest path, while *Agent*-2 goes directly to v_0 by walking in counterclockwise direction (by Constraint 4, it cannot use node $v_{c-k'-1}$ to return to s):

$$\begin{aligned} \phi^1 &= \langle v_{c-k'-2}, v_{c-k'-1} \rangle \circ P(v_{c-k'-1}, v_0 \rangle \\ \phi^2 &= \langle v_{c-k'-4}, v_{c-k'-5}, \ldots, v_1, v_0 \rangle \end{aligned}$$

   The length of this part is:

$$\max(\min(c-k'+1, 3+k'), c-k'-3)$$

   which is equal to $c-k'-3$ if we assume that $c > 6+2k'$.
   The total execution time is therefore:

$$\begin{aligned} & k'(k'+1) + (k'-1) + 3k'' + c - k' - 3 \\ = \; & k'^2 + 2k' - 1 + 3c - 6k' - 6 + c - k' - 3 \\ = \; & k'^2 - 5k' + 4c - 10 \end{aligned}$$

   Such formula has its minimum for $k' = 2, 3$, where its value is $4c - 16$.

2. *2 nodes left and* $k''' = 0$. Also in this case parts *(iii)* and *(iv)* are not in the exploration scheme; the last two nodes to be explored are $v_{k'+k''+1} = v_{c-k'-2}$ and $v_{k'+k''+2} = v_{c-k'-1}$. Hence $k'' = c - 2k' - 3$. The meeting point is node $v_{c-k'-4}$.
   In the last phase one agent (say *Agent*-1) explores node $v_{c-k'-2}$ and then goes to v_0 in the counterclockwise direction, while the other one (*Agent*-2) goes, in the counterclockwise direction, to $v_{c-k'-1}$ and then to v_0:

$$\begin{aligned} \phi^1 &= P(v_{c-k'-4}, v_{c-k'-2} \rangle \circ P(v_{c-k'-2}, v_0 \rangle \\ \phi^2 &= P(v_{c-k'-4}, v_{c-k'-1} \rangle \circ P(v_{c-k'-1}, v_0 \rangle \end{aligned}$$

The length of this part is:

$$\max(c-k', c+k'-2) = c+k'-2.$$

The total execution time is therefore:

$$\begin{aligned} & k'(k'+1)+(k'-1)+3k''+c+k'-2 \\ = \; & k'^2+2k'-1+3c-6k'-9+c+k'-2 \\ = \; & k'^2-3k'+4c-12 \end{aligned}$$

Such formula has its minimum for $k' = 1, 2$, where its value is $4c-14$.

3. *1 node left and $k''' > 0$.* The total length of parts *(i)-(iv)* is:

$$k'(k'+1)+(k'-1)+3k''+2k'+k''+3+1+3k''' = \\ k'^2+4k'+4k''+3k'''+3.$$

The last node to be explored is node $v_{k'+k''+2} = v_{c-k'-k'''-2}$, hence

$$c-2k'-k''-k'''-4 = 0 \tag{5.13}$$
$$k''+k''' = c-2k'-4. \tag{5.14}$$

The meeting point is node $v_{c-k'-k'''}$.
In the last phase one agent (say again *Agent*-1) explores node $v_{c-k'-k'''-2}$ and then goes to v_0 by following the shortest path, while the other one (*Agent*-2) goes directly to v_0 by walking in the clockwise direction:

$$\begin{aligned} \phi^1 &= \langle v_{c-k'-k'''-1}, v_{c-k'-k'''-2}\rangle \circ P(v_{c-k'-k'''-2}, v_0\rangle \\ \phi^2 &= \langle v_{c-k'-k'''+1}, v_{c-k'-k'''+2}, \ldots, v_{c-1}, v_0\rangle \end{aligned}$$

The length of this last phase is:

$$\max[\min(k'+k'''+4, k'+k''+4), k'+k'''-1]$$

The total execution time is therefore:

$$k'^2+5k'+4k''+4k'''+7 \qquad \text{if } k'' \geq k''' \tag{5.15}$$
$$k'^2+5k'+5k''+3k'''+7 \qquad \text{if } k'''-5 < k'' < k''' \tag{5.16}$$
$$k'^2+5k'+4k''+4k'''+2 \qquad \text{if } k'' \leq k'''-5 \tag{5.17}$$

By applying Equation 5.14 in 5.15 we obtain a total cost of $k'^2 - 3k' + 4c - 9$, whose minimum value (for $k' = 1, 2$) is $4c - 11$.
By applying Equation 5.14 in 5.17 we obtain a total cost of $k'^2 - 3k' + 4c - 14$, whose minimum value (for $k' = 1, 2$) is $4c - 16$.
By using Equation 5.13, and assuming $k'' = k''' - x$ (where $x = \{1, 2, 3, 4\}$), in Equation 5.16 we obtain as total cost:

$$\begin{aligned} k'^2 + 5k' + 8k''' + 7 - 5x &= \\ k'^2 + 5k' + 4(c - 2k' - 4 + x) + 7 - x &= \\ k'^2 - 3k' + 4c - 9 - x, & \end{aligned}$$

whose minimum value (for $k' = 1, 2$ and $x = 4$) is $4c - 15$.

4. *2 nodes left and $k''' > 0$*: The total length of parts *(i)-(iv)* is:

$$k'(k' + 1) + (k' - 1) + 3k'' + 2k' + k'' + 3 + 1 + 3k''' = \\ k'^2 + 4k' + 4k'' + 3k''' + 3.$$

The last two nodes to be explored are $v_{k'+k''+2} = v_{c-k'-k'''-3}$ and $v_{k'+k''+3} = v_{c-k'-k'''-2}$, hence

$$c - 2k' - k'' - k''' - 5 = 0 \tag{5.18}$$

The meeting point is node $v_{c-k'-k'''}$.
In the last phase one agent (say *Agent*-1) explores node $v_{c-k'-k'''-2}$ and then goes to v_0 in the clockwise direction, while the other one (*Agent*-2) goes, in the clockwise direction, to $v_{c-k'-k'''-3}$ and then to v_0 (in the counterclockwise direction):

$$\begin{aligned} \phi^1 &= P(v_{c-k'-k'''}, v_{c-k'-k'''-2}\rangle \circ P(v_{c-k'-k'''-2}, v_0\rangle \\ \phi^2 &= P(v_{c-k'-k'''}, v_{c-k'-k'''-3}\rangle \circ P(v_{c-k'-k'''-3}, v_0\rangle \end{aligned}$$

The length of this last phase is:

$$\max(4 + k' + k''', 3k' + 2k'' + k''' + 4) = 3k' + 2k'' + k''' + 4.$$

The total execution time is therefore:

$$k'^2 + 7k' + 6k'' + 4k''' + 7$$

By using Equation 5.18 we can replace k'' and obtain as total cost $k'^2 - 5k' + 6c - 2k''' - 23$. In order to minimize this formula we have to maximize the value of k'''. However, since $k'' \geq 1$, by using again Equation 5.18, $k''' \leq c - 2k' - 6$ and hence we obtain, as total cost:

$$k'^2 - 3k' + 4c - 11$$

The minimum value of this formula (for $k' = 1, 2$) is $4c - 13$.

Therefore, we can define the following exploration scheme $\mathcal{E}^*_{G_c,v_0}$ for a ring network G_c:

i) 2 *c-split* phases;

ii) $(c-6)$ *f-probe* phases;

iii) the following last phase:

$$\begin{aligned}\phi^1 &= \langle v_{c-4}, v_{c-3}, \ldots, v_{c-1}, v_0\rangle \\ \phi^2 &= \langle v_{c-6}, v_{c-7}, \ldots, v_1, v_0\rangle\end{aligned}$$

Lemma 5.5.1, together with Lemma 5.5.2, Lemma 5.5.3 and the previous computations, imply the following theorem.

Theorem 5.5.5 *Given a ring network G_c and a node v_0 as the starting point, the exploration scheme $\mathcal{E}^*_{G_c,v_0}$ for G_c has cost $4c-16$ and is optimal.*

Chapter 6

Conclusions and Further Works

Mobile agents systems are a rather new and promising concept in distributed computing. Their motivating idea is that, instead of exchanging moving informations between remote hosts, a task could be more efficiently performed by moving through hosts data and "thinking entities" able to manage them, encapsulated in form of mobile agents. In this work we analyzed, from an algorithmic perspective, a problem related to security in mobile agents systems. In particular we studied the problem of detecting a harmful host (called black hole) which destroys visiting mobile agents without leaving any trace. We showed that, among all possible security threats, black holes represent one of the most difficult to deal with. The problem has been presented and studied in the works of Dobrev *et al.* ([18, 17, 16, 19, 14]). The model considered in such papers assumes that the network is totally asynchronous, i.e., every execution step requires finite time, but this cannot be upper bounded. In this setting it was observed that if the network is not 2-connected, then the problem is unsolvable. Moreover, in an asynchronous network it is impossible to answer the question of whether a black hole actually exists, hence it is assumed that there is exactly one black hole and the task is to locate it as soon as possible.
In [11, 12] the problem is studied under the model we consider in this thesis. They assume that it is possible to fix an upper bound on the time needed by an execution step. The synchronicity assumption makes a dramatic change to the problem. A black hole can be located by two agents in any graph. Moreover the agents can decide whether there is a black hole or not. In [12] the model considered assumes that the nature of all the nodes, excluding the starting point, is unknown (*restricted problem*). This paper shows optimal algorithms for networks whose topology is a line or a bushy tree. The authors also provide a $\frac{5}{3}$-approximation algorithm for tree networks and conjecture that the black hole search problem is **NP**-hard for arbitrary networks. **NP**-hardness is proved in [11] for the *generalized* problem in which a subset of the nodes is initially known to be safe. In the same work it is also provided a 9.3-approximation algorithm for such problem.

In this thesis we provided new improved results for both the *general* and the *restricted* version of the problem.
We showed that the general problem is not polynomial time approximable within a $1+\epsilon$ factor, for any $\epsilon < \frac{1}{388}$ unless **P**=**NP**, and we improved its approximation upper bound to 6.
For the restricted problem, we proved **NP**-hardness holding also in the case that the network has a planar embedding, and **APX**-hardness in the general case. Then, we gave some approximating algorithm suited for particular network topologies. We presented an algorithm for arbitrary trees with an approximation ratio of $\frac{4}{3}+O(\frac{1}{n}) \leq \frac{10}{7}$. We provided also a $\frac{27}{8}$-approximation algorithm for arbitrary networks. We used the technique of first generating a spanning tree of the network, and then using an algorithm designed for trees to explore the graph. We discussed further about this approach, by showing its properties and limitations. Finally, we presented an optimal algorithm for ring networks.

We conclude by addressing some directions for further works. First, most of the bounds presented in this thesis are far from being tight. For what concerns the restricted problem, we believe that one could show a better upper bound for the approximation ratio of our algorithm than $\frac{27}{8}$ by further refining the analysis; however we do not expect a bound anywhere near the current lower bound on the approach (i.e., $3/2$). It seems that to obtain a more substantial improvement of the approximation ratio one would need to abandon the spanning-tree approach, but algorithms which attempt something considerably different than following a spanning tree may be very difficult to analyze. For example, one no longer would be able to assume the absence of the black hole in the worst case scenario. A still interesting question is to discover the nature of the black hole search problem in tree networks. Our conjecture is that there exists a polynomial time algorithm; however, the relative complexity of the optimal algorithm for line networks presented in [12], lead us to expect that the formulation of such algorithm is not straightforward. For what concerns the general problem, the big gap between the approximation upper and the lower bounds leaves space for further research. For instance, a speedup of the 6-approximation algorithm could possibly be obtained by replacing *probe* phases by *splits*. This however can lead to a better approximation only if we are provided with a suited spanning tree of graph $\widehat{G}$.

Further research might be devoted also to the study of problems arising by modifying the assumption we put in Section 3.2. It would be interesting, for instance, to see what non-trivial result could be shown about the complexity of computing fast black hole search schemes for many agents and possibly many black holes. If there are $k+1$ agents, where k is a parameter (not a constant) and at most k black holes, then it is not even clear how one should formalize the problem. The "oblivious" approach of giving each agent one predetermined sequence of nodes to visit does not seem adequate if there are more than two agents. On the other hand, we believe that also by changing the communication mechanism (e.g., by allowing agents to write on

whiteboards in agents platforms) we are led to completely new problems, which need totally different approaches. As we already discussed in the introduction, it would be also interesting to study the black hole search problem under the assumption that the map of the network is unknown. In this case the problem can be better modeled as an on-line problem, and a competitive analysis seems to be more suited. One can also see this as a mixed problem, where, similarly to the work presented in [17], two distinct targets (black hole search and map building) are pursued at the same time. We believe interesting to further carry out research into this kind of problems.

Bibliography

[1] ALBERS, S., AND HENZINGER, M. Exploring unknown environments. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing* (1997).

[2] ALPERN, S., BASTON, V., AND ESSEGAIER, S. Rendezvous search on a graph. *Journal of Applied Probability 36*, 1 (1999), 223–231.

[3] ARKIN, E. M., BENDER, M. A., FEKETE, S. P., MITCHELL, J. S. B., AND SKUTELLA, M. The freeze-tag problem: how to wake up a swarm of robots. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2002), Society for Industrial and Applied Mathematics, pp. 568–577.

[4] BARRIÈRE, L., FLOCCHINI, P., FRAIGNIAUD, P., AND SANTORO, N. Capture of an intruder by mobile agents. In *SPAA'02* (August 2002).

[5] BENDER, M., FERNANDEZ, A., RON, D., SAHAI, A., AND VADHAN, S. The power of a pebble: exploring and mapping directed graphs. *Information and computation 176* (2002), 1–21.

[6] BENDER, M. A., AND SLONIM, D. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (1994), pp. 75–85.

[7] BERN, M., AND PLASSMANN, P. The steiner problem with edge lengths 1 and 2. *Information Processing Letters 32* (1989), 171–176.

[8] BREISCH, R. An intuitive approach to speleotopology. *Southwestern Cavers 6*, 5 (1967), 72–78.

[9] CHESS, D. Security issues in mobile code systems. In *Mobile Agents and Security* (1998), G. Vigna, Ed., LNCS 1419, Springer-Verlag.

[10] CSABA, B., KARPINSKI, M., AND KRYSTA, P. Approximability of dense and sparse instances of minimum 2-connectivity, TSP and path problems. In *SODA*

'02: Proceedings of the thirteenth annual ACM-SIAM Symposium on Discrete algorithms (Philadelphia, PA, USA, 2002), pp. 74–83.

[11] CZYZOWICZ, J., KOWALSKI, D., MARKOU, E., AND PELC, A. Complexity of searching for a black hole. manuscript, 2004.

[12] CZYZOWICZ, J., KOWALSKI, D., MARKOU, E., AND PELC, A. Searching for a black hole in tree networks. In *Proceedings of 8th International Conference on Principles of Distributed Systems (OPODIS 2004)* (2004), pp. 34–35.

[13] DENG, X., AND PAPADIMITRIOU, C. H. Exploring an unknown graph. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science* (1990), pp. 356–361.

[14] DOBREV, S., FLOCCHINI, P., KRÀLOVIC, R., PRENCIPE, G., RUZICKA, P., AND SANTORO, N. Black hole search by mobile agents in hypercubes and related networks. In *Proceedings of 6th International Conference on Principles of Distributed Systems (OPODIS 2002)* (2002), pp. 169–180.

[15] DOBREV, S., FLOCCHINI, P., PRENCIPE, G., AND SANTORO, N. Mobile agents searching for a black hole in an anonymous ring. In *Proc. 15th Int. Symposium on Distributed Computing (DISC 2001)* (2001), Springer LNCS vol. 2180, pp. 166–179.

[16] DOBREV, S., FLOCCHINI, P., PRENCIPE, G., AND SANTORO, N. Searching for a black hole in arbitrary networks: Optimal mobile agent protocols. In *Proceedings of 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)* (2002), pp. 153–161.

[17] DOBREV, S., FLOCCHINI, P., PRENCIPE, G., AND SANTORO, N. Multiple agents rendezvous in a ring in spite of a black hole. In *Proceedings of 7th International Conference on Principles of Distributed Systems (OPODIS 2003)* (2003), Springer LNCS vol. 3144, pp. 34–46.

[18] DOBREV, S., FLOCCHINI, P., PRENCIPE, G., AND SANTORO, N. Mobile search for a black hole in an anonymous ring. *Algorithmica To appear* (2005).

[19] DOBREV, S., FLOCCHINI, P., AND SANTORO, N. Improved bounds for optimal black hole search with a network map. In *Proceedings of 11th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2004)* (2004), R. Kràlovic and O. Sỳkora, Eds., Springer LNCS vol. 3104, pp. 111–122.

[20] DUDEK, G., JENKIN, M., MILIOS, E., AND WILKES, D. Robotic exploration as graph construction. *IEEE Trans. Robot. Automat. 7* (1991), 859–865.

[21] ENGEBRETSEN, L., AND KARPINSKI, M. TSP with bounded metrics: stronger approximation hardness. Tech. Rep. 85264, University of Bonn, February 2005.

[22] GAREY, M., AND JOHNSON, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., 1979.

[23] GAREY, M., JOHNSON, D., AND TARJAN, R. The planar hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing 5*, 4 (1976), 704–714.

[24] GREENBERG, M., BYINGTON, J., AND HARPER, D. Mobile agents and security. *IEEE Communications Magazine 36*, 7 (1998), 76–85.

[25] HANUSSE, N., KRANAKIS, E., AND KRIZANC, D. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics 137* (2004), 69–85.

[26] HOHL, F. Time limited black box security: Protecting mobile agents from malicious hosts. In *Proceedings of Conference on Mobile Agent and Security* (1998), G. Vigna, Ed., Springer LNCS vol. 1419, pp. 90–111.

[27] HOHL, F. A framework to protect mobile agents by using reference states. In *Proc. 20th Int. Conf. on Distributed Computing Systems (ICDCS '00)* (2000), pp. 410–417.

[28] KLASING, R., MARKOU, E., RADZIK, T., AND SARRACCO, F. Approximation bounds for black hole search problems. In *Proceedings of 9th International Conference on Principles of Distributed Systems (OPODIS 2005)* (2005).

[29] KLASING, R., MARKOU, E., RADZIK, T., AND SARRACCO, F. Hardness and approximation results for black hole search in arbitrary graphs. In *Proc. 12th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO 2005)* (2005), Springer LNCS vol. 3499, pp. 200–215.

[30] KLEITMAN, D., AND WEST, D. Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics 4*, 1 (1991), 99–106.

[31] LYNCH, N. A. *Distributed algorithms*. Morgan Kaufmann Publishers, 1997.

[32] MORIN, J.-H. Hypernews, a hypermedia electronic-publishing environment based on agents. In *Proceedings of the 31st Annual Hawaii International Conference on System Sciences (HICSS)* (1998), pp. 58–67.

[33] OPPLIGER, R. Security issues in mobile code systems. *Computer Communications 22*, 12 (1999), 1165–1170.

[34] PANAITE, A., AND PELC, A. Exploring unknown undirected graphs. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms* (1998), ACM-SIAM.

[35] ROBINS, G., AND ZELIKOVSKY, A. Improved steiner tree approximation in graphs. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2000), Society for Industrial and Applied Mathematics, pp. 770–779.

[36] SCHELDERUP, K., AND OLNES, J. Mobile agent security – issues and directions. In *Proc. 6th Int. Conf. on Intelligence and Services in Networks* (1999), Springer LNCS vol. 1597, pp. 155–167.

[37] SEIDEL, R. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci. 51*, 3 (1995), 400–403.

[38] TEL, G. *Introduction to Distributed Algorithms*, 2nd ed. Cambridge University Press, 2000.

[39] TENNENHOUSE, D. Active networks. In *Proceedings of 2nd Symposium on Operating Systems Design and Implementation (OSDI '96)* (October 1996), USENIX, Ed.

[40] VITEK, J., AND CASTAGNA, G. Mobile computations and hostile hosts. In *Mobile Objects* (1999), D. Tsichritzis, Ed., University of Geneva, pp. 241–261.