Black Hole Search in the Network and Subway Models

Matthew Kellett

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy in Computer Science[†]

School of Information Technology and Engineering

Faculty of Engineering

University of Ottawa

[†] The Ph.D. program is a joint program with Carleton University, administered by Ottawa-Carleton Institute for Computer Science.

Abstract

In this thesis we look at mobile agent solutions to black hole search and related problems. Mobile agents are computational entities that are autonomous, mobile, and can interact with their environment and each other. The black hole search problem is for a team of these agents to work together to map or explore a graph-like network environment where some elements of the network are dangerous to the agents. Most research into black hole search has focussed on finding a single dangerous node: a black hole. We look at the problem of finding multiple black holes and, in the case of dangerous graph exploration, multiple black links as well.

We look at the dangerous graph exploration problem in the network model. The network model is based on a normal static computer network modelled as a simple graph. We give an optimal solution to the dangerous graph exploration problem using agents that start scattered on nodes throughout the network. We then make the problem more difficult by allowing an adversary to delete links during the execution of the algorithm and provide a solution using scattered agents.

In the last decade or two, types of networks have emerged, such as ad hoc wireless networks, that are by their nature dynamic. These networks change quickly over time and can make distributed computations difficult. We look at black hole search in one type of dynamic network described by the subway model, which we base on urban subway systems. The model allows us to look at the cost of opportunistic movement by requiring the agents to move using carriers that follow routes among the network's sites, some of which are black holes. We show that there are basic limitations on any solution to black hole search in the subway model and prove lower bounds on any solution's complexity. We then provide two optimal solutions that differ in the agents' starting locations and how they communicate with one another.

Our results provide a small window into the cost of deterministic distributed computing in networks that have dynamic elements, but which are not fully random.

For Jennifer

Acknowledgements

My first thanks go to my supervisors, Paola Flocchini, Nicola Santoro, and Peter Mason. Collectively, they were incredibly supportive and helpful in guiding my studies. I could not have asked for a more agreeable and professional group of people. I would like to thank Paola for taking a chance on me five years ago when she agreed to supervise my master's and for her unwavering support and encouragement since then. I would like to thank Nicola for agreeing to co-supervise me. It was honour both to be taught by and collaborate with one of the most senior people in the field. I would like to thank Peter for agreeing to represent Defence R&D Canada – Ottawa. As my thesis co-supervisor, he gave me invaluable insight into the scientific process. As my work supervisor, he gave me the leeway I needed to succeed at both work and school. I am proud to count all three of my supervisors not only as mentors but as friends.

My studies would not have been possible without the support of Julie Lefebvre, my manager at DRDC Ottawa. I would like to thank her for taking a chance on me when it was not at all clear that I would succeed.

Finally, I would like to thank my parents, George and Cherry Kellett, for their love and support. This thesis is dedicated to my partner (my wife), Jennifer Panek, with all my love.

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

A distributed system is a collection of computational entities that are connected together in some way. A distributed algorithm, or protocol, is a set of rules followed by the entities in the distributed system to perform, collectively, some task. The most common form of distributed system with which people are familiar is the computer network. The computers in the network are connected together by communications links such as wireless, Ethernet, or optical fibre. The computers work together to perform tasks such as sending and receiving e-mail or surfing the web. The task can also be at a much lower level such as discovering neighbours, routing packets, or converting universal resource locators (URLs) into numerical addresses. Computer networks are often represented by a graph like the one in Figure 1.1, where the computers are the vertices or nodes and the communication channels between them are the edges or links.

All these systems work on the same basic concept. Each computer has a copy of the algorithm and messages are sent through low bandwidth channels to neighbouring computers to execute or coordinate the task. In the literature, this

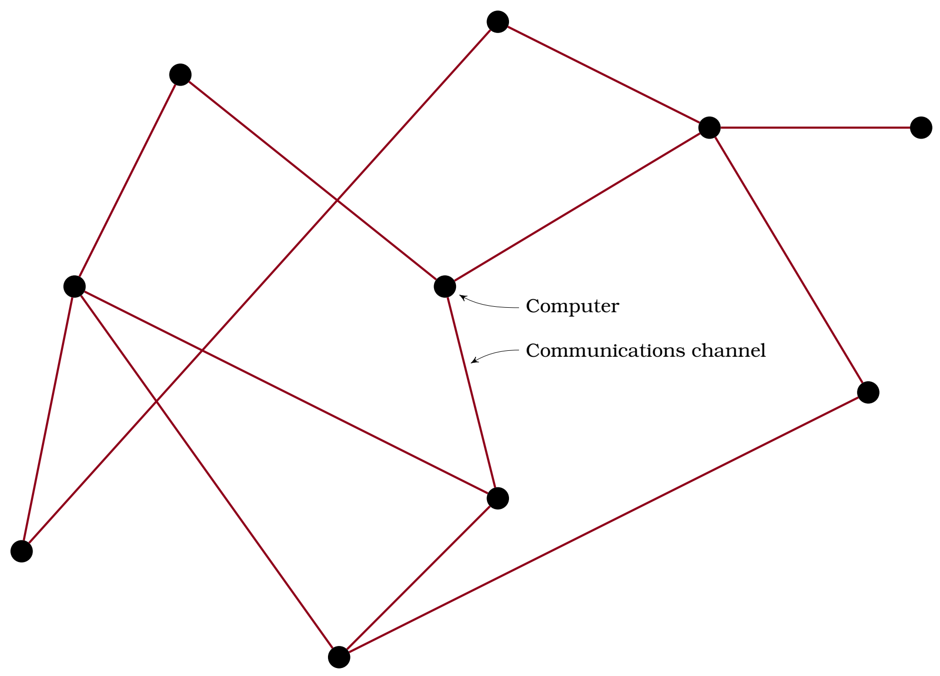

Figure 1.1: Example of a computer network represented as a graph with the computers as vertices or nodes and the communication channels between them as edges or links.

approach is called the *message passing* model and it has been the main focus of study in distributed computing since the invention of the computer network. Common problems in the message passing model include broadcast, gossiping, spanning tree construction, and election. Solutions to these problems are used as primitives that are combined to create the more complex protocols that underlie everything that is done on computer networks like the Internet.

The basic assumptions behind the message passing model are that the computational entities are stationary and they communicate by sending messages. There are, however, distributed systems where the computational entities themselves are mobile. This approach to distributed systems is referred to in the literature as the *mobile agent* model. The computational entities in the mobile agent model can be software agents moving around a computer network or autonomous robots in a planar or graph-like environment—in fact, the robot version of this model is used for the theoretical side of the robotics research. A computer virus is the most obvious example of a mobile software agent. The computers it infects do not share the same protocol, although they may share the same vulnerability. The virus contains its own protocol and moves itself around the network. Common problems in the mobile agent model include exploration, rendezvous or gathering, map construction, map verification, leader election, and intruder capture or decontamination.

Research into computer networks as distributed systems has generally made the assumption that the networks are safe. That is to say that the research assumes for simplicity's sake that nothing goes wrong while the distributed algorithm is being executed. Users of computer networks know that this assumption is not always valid. Things go wrong. Computers and equipment crash. The first question from technical support personnel is often, "Have you rebooted your computer?" In the message passing model, these problems with the network are referred to as *faults* and they have been the focus of research into fault-tolerant computing. In the mobile agent model, these faults are often referred to as *black holes* or *black links* or the network as a whole is described as a *dangerous graph*.

Another inherent assumption in much of the research into distributed algorithms is that the network does not change during the execution of the distributed algorithm; the network remains static. Again, like faults, the assumption that the network is static is often made for simplicity's sake, since it is much more difficult to deal with a network that is dynamically changing. In general, this assumption has been valid for most of the history of computer networks. The computers that make up the network have traditionally been connected by wires that were more or less fixed. Computers are added and removed and the network reconfigured, but these changes take place on a time scale that is much longer than the execution of the average distributed algorithm. Because of the time scale, intentional changes can be seen, in the same way as unintentional changes, as faults, and can be dealt with accordingly.

There are a number of classes of networks that have arisen in the last decade or two that are dynamic by nature. Wireless networks, especially mobile ad hoc networks, are prone to change frequently. Sensors in wireless sensor networks may only be connected intermittently depending on their sleep schedule and available power. Vehicular ad hoc networks are now being considered where vehicles on highways and roads would form an ever-changing peer-to-peer network for distributing information about accidents, traffic flow, and nearby services. One approach to the problem of computing in such networks is to treat these changes as faults. However, approaches to fault-tolerant computing often assume that faults occur on a time scale that makes them a rare or special event. In a dynamic network, changes in the network's topology are not rare or special—they are not faults in the traditional sense at all—they are an inherent property of the system.

In this thesis, we are interested in distributed algorithms for the mobile agent model that work in dynamic networks with dangerous elements. Specifically, we look at the problem of exploration (EXP) and its related problem of map construction. Exploration in a network environment is the problem of crossing every edge in the network or visiting every node. Map construction is the problem of having the mobile agents construct a map of the network during their exploration. Both these problems generally assume a safe network. We are interested in networks that have black holes and possibly black links. The problem of exploring a network in the presence of black holes is called black hole search (BHS). The problem of exploring a network in the presence of both black holes and black links is called dangerous graph exploration (DGE). We are generally interested in the map construction variations on these problems. Constructing a map of a dangerous network can act as a primitive to allow follow-on protocols to work in the network as if it is safe. We refer collectively to problems of map construction and exploration in the presence of dangerous elements as black hole search because it is by far the more commonly used term in the literature.

We look at black hole search in two models: the network model and the subway model. The network model is based on a normal static computer network. We provide a solution to the DGE problem in the network model and we provide another DGE-related solution that works when an adversary is allowed to delete links in the network during the execution of the algorithm. The subway model is a dynamic network model, with edges that appear periodically, based on a real-life subway system. Instead of moving themselves around, the agents in this model use carriers that move on routes between the stations or sites of the network. The subway model allows us to examine the cost of this "opportunistic" movement—moving only when transportation is available—on the black hole search problem. We provide two solutions to the BHS problem in the subway model. In one solution the agents start co-located on the same site in the network and communicate with one another through shared memory or *whiteboards* at the stations and in the other the agents start scattered on sites throughout the network and communicate through whiteboards on the carriers.

1.1 Our contributions

In this thesis, we present a comprehensive review of literature related to black hole search and its variants in the mobile agent model and related literature on exploration and dynamic networks. We present two new solutions to black hole search problems in the network model, one of which has been proven to be optimal. We introduce the subway model, which models a class of dynamic networks with periodically existing edges based on real-life subway systems. We prove the necessity of a number of assumptions needed to solve the Exp and Bhs problems in the model and prove a lower bound on the complexity of solutions to the Bhs problem. We then present two optimal solutions to the Bhs problem in the subway model, which vary in the starting locations of the agents and the means of inter-agent communication.

Most of the existing work on black hole search has focussed on the problem of a team of co-located agents—agents starting on the same node—finding a single black hole in the network. One of the first papers on the problem (Dobrev et al., 2002) (later published as a journal article (Dobrev et al., 2006c)) looks at the effects of topological knowledge on the solution to the problem in arbitrary networks. The authors prove a lower bound when the agents are completely ignorant of the topology of $\Omega(n^2)$ moves, where n is the number of nodes. The authors provide an optimal solution to the problem. The lower bound is comparable to the $O(m \log k)$-move solution to the equivalent exploration problem (in a safe graph) recently studied in (Das et al., 2007), where m is the number of links and k is the number of agents.

We extend this result in two ways. In Chapter 4, we present a solution using agents that start scattered in the network. Instead of a single black hole, we allow for any number of black holes and black links, with certain limitations. The result is an algorithm that solves the DGE problem in the network model in $O(nm)$ moves, where n is the number of nodes, and m is the number of links. Our result can be contrasted with the result for anonymous graphs that requires $O(m^2)$ moves (Chalopin et al., 2007). A preliminary version of these results was presented at IPDPS 2009 (Flocchini et al., 2009a). Based on that paper, our DGE solution was shown to be optimal in (Miklik, 2010).

In Chapter 5, we present a solution to the same scattered agent, arbitrary network DGE problem, but we allow an adversary to delete an arbitrary number of links from the network, with certain limitations, during the execution of the algorithm. We call this the DGE with link deletions (DGE-LD) problem. The result is a DGE-LD solution that uses $O(nm^2)$ moves.

Recently, the research community has started to consider the problem of mobile agent distributed algorithms in dynamic networks. One of the first works on exploration in such a network introduced the concept of a periodically varying graph (PV graph) (Flocchini et al., 2009b). In a PV graph, carriers move synchronously from station to station on routes. To explore the PV graph, the agent jumps from carrier to carrier when two carriers are co-located at the same station. The authors present a solution to exploration of carriers with homogenous routes (all route lengths are the same) that requires $O(n_C \cdot l_R)$ time and a solution for heterogenous routes (route lengths may be different) that requires $O(n_C \cdot l_R^2)$ time, where n_C is the number of carriers and l_R is the length of the longest route.

We extend the PV graph result by asking the question, what happens if an agent can step down from the carrier? The answer to this question naturally leads to the definition of the subway model, which we introduce in Chapter 2.

In Chapter 6, we introduce and prove some additional assumptions that are needed to make the Exp and Bhs problems solvable in the subway model. We then prove a lower bound on the complexity of any solution to the Bhs problem. Such a solution requires $\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, where γ is the number of stops that all carriers make on black hole sites. Carrier moves is a complexity measure similar to agent moves but specific to the subway model. Both our solutions work with an optimal number of agents, $k = \gamma + 1$.

In Chapter 7, we look at the Bhs problem in the subway model when the agents start co-located on the same site and communicate using whiteboards on the stations. We prove some basic limitations specific to this version of the model and then present an optimal solution to the Bhs problem. Our solution requires $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves.

In Chapter 8, we look at the Bhs problem in the subway model when the agents start scattered in the network and communicate using whiteboards on the carriers. We conjecture on a basic limitation specific to this version of the model and then present an optimal solution to the Bhs problem. Our solution is based on the cooperative independent exploration of the network by the agents and has a complexity of $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves.

A preliminary version of the subway model (Chapter 2) and its properties (Chapters 6 and 7), the proof of the lower bound on black hole search (Chapter 6), and the solution for black hole search with co-located agents using site whiteboards (Chapter 7) was presented at FUN 2010 (Flocchini et al., 2010b). A more detailed version of these results appears in *Theory of Computing Systems* (Flocchini et al., 2012).

Network Model		
Problem	**Result**	**Reference**
EXP by co-located agents	$O(m \log k)$	Das et al. (2007)
BHS by co-located agents	$\Theta(n^2)$	Dobrev et al. (2002, 2006c)
DGE lower bound	$\Omega(nm)$	Miklik (2010)
DGE by scattered agents	$O(nm)$	Chap. 4, Flocchini et al. (2009a)
DGE-LD by scattered agents	$O(nm^2)$	Chap. 5
DGE by scattered agents in an anonymous network	$O(m^2)$	Chalopin et al. (2007)

Subway Model		
Problem	**Result**	**Reference**
EXP in a PV graph with homogenous routes	$O(n_C \cdot l_R)$	Flocchini et al. (2009b)
EXP in a PV graph with heterogenous routes	$O(n_C \cdot l_R^2)$	Flocchini et al. (2009b)
EXP lower bound	$\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chap.6, Flocchini et al. (2010b, 2012)
BHS by co-located agents with site whiteboards	$O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chap. 7, Flocchini et al. (2010b, 2012)
BHS by scattered agents with carrier whiteboards	$O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chap. 8
Note that $k \geq \gamma + 1$ for our subway model solutions.		

Table 1.1: Thesis and related results.

The results from our solutions and related solutions are summarized in Table 1.1.

1.2 Thesis organization

We describe the organization of the thesis as whole in this section. For the reader who wishes to focus solely on one or two of the solutions presented, we also provide a thesis roadmap in Figure 1.2.

After this introduction, Chapter 2 introduces the definitions and models used in the rest of the thesis. We define the concepts of mobile agents and black holes and discuss some of their properties. We define the network model and its assumptions and we introduce the subway model. In Chapter 3, we review the literature related to black hole search in dynamic networks. In particular, we give an overview of the literature related to exploration by mobile agents in many different network models. We present a comprehensive review of literature on black hole search and dangerous graph exploration. We conclude with a review of relevant work on dynamic networks and a brief mention of work on other problems in the mobile agent model.

In the next five chapters, we present our solutions related to black hole search in the two models and prove restrictions and lower bounds on solutions to the problem in the subway model. We present an optimal solution to DGE by scattered agents in the network model in Chapter 4. We present a solution to DGE-LD by scattered agents in the network model in Chapter 5. In Chapter 6, we prove certain assumptions are needed to solve the EXP and BHS problems in the subway model and we prove lower bounds on any BHS solution. We present an optimal solution

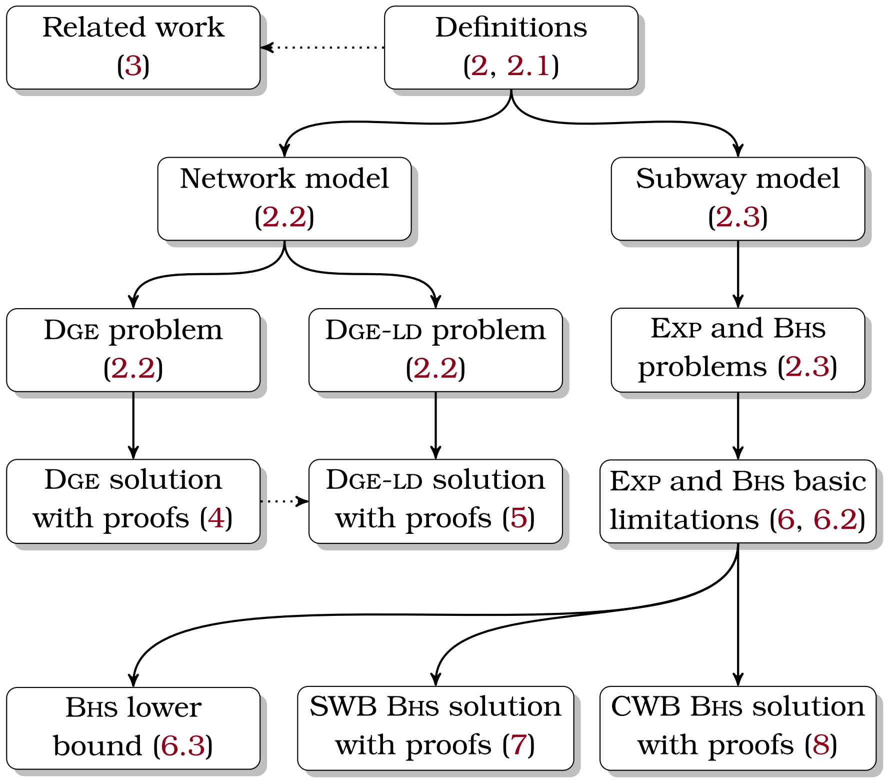
Related work (3)
Definitions (2, 2.1)
Network model (2.2)
Subway model (2.3)
DGE problem (2.2)
DGE-LD problem (2.2)
EXP and BHS problems (2.3)
DGE solution with proofs (4)
DGE-LD solution with proofs (5)
EXP and BHS basic limitations (6, 6.2)
BHS lower bound (6.3)
SWB BHS solution with proofs (7)
CWB BHS solution with proofs (8)

Figure 1.2: Thesis roadmap. Excludes the Introduction (1) and Conclusion (9). Dotted connections denote useful but optional prerequisites. SWB is site whiteboard and CWB is carrier whiteboard.

to BHS by co-located agents in the subway model with site whiteboards in Chapter 7. We present an optimal solution to BHS by scattered agents in the subway model with carrier whiteboards in Chapter 8.

We conclude in Chapter 9 with a summary of our results and a discussion of possible future work based on them.

CHAPTER 2

Definitions, Models, and Problems

In this chapter, we discuss the concepts of mobile agents and black holes and then present the two models for which we give solutions on variations to the black hole search problem for mobile agents.

We define the concepts of mobile agents and black holes in Section 2.1. We discuss the distributed algorithm view of mobile agents and give a general definition in Section 2.1.1. In Section 2.1.2, we discuss the concept of black holes as presented in one of the early works on black hole search, including their properties in asynchronous networks such as the ones studied in this thesis.

In Section 2.2, we present the network model, a straightforward model of a computer network as a simple graph. We present the assumptions made about how the network is constructed, how mobile agents work within it, and the nature of the dangerous elements that the agents must find. We define the DGE and DGE-LD problems for which we provide solutions in Chapters 4 and 5 and the additional assumptions needed to make those solutions possible.

In Section 2.3, we introduce the subway model, a dynamic network based on real-life urban subway systems. We present the assumptions made about how the network is constructed, how mobile agents work within it, and the nature of the dangerous elements that the agents must find. We define the EXP and BHS problems and summarize the additional assumptions required to solve these problems that we prove in Chapter 6. We also discuss how complexity is measured in the subway model.

The term "model" takes two very distinct meanings in this thesis and in the literature in general. The network and subway "models" described in this chapter are the complete set of assumptions under which we provide solutions to black hole search related problems. The term "model" can also be applied to certain significant or distinguishing assumptions, changes in which often require significant changes in the approach to the problem. For instance, we have already mentioned the message passing and mobile agent models. We discuss a number of these distinguishing assumptions for mobile agent solutions in Section 3.1. We use the term "model" in both senses throughout the thesis with the intended sense being made clear by the context.

2.1 Definitions

In this section, we define the concepts of mobile agents and black holes. We give a general definition of the algorithmic view of mobile agents. We also define black holes and their properties as they are presented in one of the earliest works on the black hole search problems. In subsequent sections, we define exactly how these concepts work in our network and subway models.

2.1.1 Mobile agents

A mobile agent can best be described in common terms as a mobile software agent, an entity that can move itself around a computer network at will. There are a number of subtly different versions of mobile agents that appear in the scientific literature, particularly in research on artificial intelligence and software engineering. We are interested in the distributed computing version of mobile agents, which has been the subject of research for the last decade or so. The "algorithmic" view of mobile agents and its relation to the other types of agents was formalized by Kranakis and Krizanc (2006) in 2006. Based on that view, we derive the following definition of a mobile agent.

Definition 2.1 (Mobile agent)**.** *A* mobile agent *is an* autonomous, mobile, *and* interactive *computational entity in a distributed system. Once an agent is started it follows its algorithm autonomously and without further external input. It can move freely around the system, restricted only by the nature of the system itself. It can interact with its environment to find available information about the elements of the system and to communicate with other agents.*

The traditional model of distributed computing is the message passing model. In the message passing model, each node has a copy of the algorithm and communicates with its neighbours by sending messages along the communication links connecting them. We can think of the mobile agent model as moving the algorithm into the message. In essence, the node ceases to be the computational entity in the system and the message becomes the computational entity in the form of a mobile agent. If we also transfer the "computer", the computational power, into the mobile agent, we get the robot version of mobile agents. In fact, mobile agent research is the theoretical side of robotics research, allowing researchers to consider mobile agents in the plane as well as the network. In graph-like environments, like the computer networks described by the network model below, it turns out that the message passing model and the mobile agent model are computationally equivalent (Barrière et al., 2003; Chalopin et al., 2006) when mobile agents use the whiteboard model of inter-agent communication. Despite the similarities, each model, message passing and mobile agent, lends itself to certain problems more naturally, such as is the case with problems we study here.

In this thesis, we are interested in a graph-like network environment for the mobile agents, as opposed to a planar one. We look at the problems of black hole search and dangerous graph exploration by mobile agents, which are variations on the exploration and map construction problems.

2.1.2 Black holes

It would be naïve to assume that all networks are safe. Components of networks, not just computer networks, can fail for any number of reasons, malicious or benign, logical or mechanical, and some of these failures are undetectable. Imagine a computer has crashed in a computer network but its neighbours continue sending messages to it. The black hole is based on this idea of an undetectable crash fault. Agents that move themselves to such a computer just disappear.

The first published work on black hole search looks at mobile agents finding a black hole in a ring network. Dobrev, Flocchini, Prencipe, and Santoro (2001) (published later as a journal article (Dobrev et al., 2007a)) are interested in how many agents are required to find the black hole and how many moves are required. They define a *black hole* as a node in the network that eliminates agents without leaving a discernible trace.

The authors consider an asynchronous ring network. In the asynchronous model, a mobile agent's movements and/or calculations take a finite but unpredictable amount of time (see Section 3.1 for a discussion of the asynchronous model and its effect on mobile agent algorithms in general). When a searching agent goes to see if the next node in the ring is a black hole, using the cautious walk technique described below, the asynchrony of its movements mean that another agent cannot wait for the searching agent to return. If the waiting agent waited for time t, someone playing an adversarial game against the waiting agent could ensure that the searching agent took time $t+1$ to return. The waiting agent would conclude that the node the searching agent was visiting is the black hole when in reality it is not. As a result, the authors prove the following property and its two corollaries,

Property 2.2. *It is impossible to distinguish between slow links and a black hole.*

Corollary 2.3. *It is impossible to determine if there is a black hole in a network.*

Corollary 2.4. *It is impossible to find a black hole unless the size of the network is known.*

Another consequence of having undetectable black holes in the network is that it is impossible to avoid agents being eliminated. Dobrev et al. (2001, 2007a) conclude that the obvious lower limit to the number of agents needed to find a single black hole is 2 agents, with one agent being eliminated by the black hole and the other agent witnessing it in some way. Dobrev, Flocchini, Prencipe, and Santoro (2002, 2006c) show that finding a single black hole in a network of unknown topology requires $\Delta+1$ agents, where Δ is the maximum degree of the network.

Since our models involve topology that is unknown to the agents and multiple black holes or other dangerous elements, we require many more agents to find these dangerous elements. All of the solutions presented in this thesis are optimal in terms of the number of agents needed to solve the black hole search related problems we investigate.

In order to bound the number of agents eliminated, Dobrev et al. (2001, 2007a) present the *cautious walk* technique. The technique allows the other agents to witness the searching agent's elimination without having to wait. Instead of walking around the ring from node to node blindly, the searching agent marks the link leading to the next node as *dangerous* and traverses the link. If the agent survives, it traverses the link again and marks it as *safe*. If the agent does not survive, the link leading to the node remains marked dangerous and no other agent can be eliminated via that link because no other agent will attempt to traverse it.

We use the cautious walk technique in all the solutions presented in this thesis. The technique used for our network model solutions is exactly the same as the one used in (Dobrev et al., 2001) and most other black hole search solutions in the literature. For the subway model, we develop a new cautious walk technique based on the same principles but adapted to the dynamic nature of a subway network.

2.2 Network model and problems

In this section, we describe the network model, a straightforward model of a network as a simple graph. We describe the network, the agents working in it and their abilities, and the nature of the dangerous elements. We then describe the two problems for which we provide solutions and the additional assumptions required to make them solvable.

Let $G = (V, E)$ be a simple connected undirected graph with $n = |V|$ vertices or nodes connected by $m = |E|$ edges or links. Let $E(u)$ be the edges incident to node $u \in V$ and $d(u) = |E(u)|$ be the degree of u. Two nodes $u, v \in V$ are neighbours if there is an edge $[u, v] \in E$ connecting them.

Working in G is a set A of $k = |A|$ agents. The agents all follow the same algorithm or protocol. Each has a distinct id, its own memory, and can move from any node to any neighbouring node. The agents' actions, their computations and movements, are *asynchronous* in that they take a finite but unpredictable amount of time. The agents start at varying times on scattered homebases in the network.

At each node, there is shared memory called a *whiteboard* that can be accessed in fair mutual exclusion by the agents residing on the node. The mutual exclusion property allows the agents to operate as if the links are first in, first out (FIFO) and as if the nodes have ids. Without loss of generality, we assume that the links and nodes have these properties.

The network in which the agents operate has elements, nodes and links, that are dangerous to the agents. We define them as follows:

Definition 2.5 (Black hole). *A black hole is a node that eliminates agents arriving at it without leaving a discernible trace.*

Definition 2.6 (Black link). *A black link is a link that eliminates agents traversing it without leaving a discernible trace.*

Collectively, black holes and black links are referred to as *dangerous elements*.

Let $V_B \subset V$ be the set of *black holes*. Let $E_B \subseteq E$ be the set of *black links*. All non-black hole nodes and non-black links are called *safe*. Let $E_I = \{[u, v] \in E : u, v \in V_B\}$

Notation	**Description**
G	Graph or network
V	Set of vertices or nodes
E	Set of edges or links
n	Number of vertices or nodes
m	Number of edges or links
$d(u)$	Degree of node u
A	Set of agents
k	Number of agents
G_S	Safe portion of G
V_S	Set of safe vertices or nodes
n_S	Number of safe vertices or nodes
V_B	Set of black holes
E_B	Set of black links
F_B	Set of frontier links
E_I	Set of inaccessible links

Table 2.1: Notation from the network model

be the set of *inaccessible links*, which are links, black or safe, that connect black holes. Let $F_B = \{[u,v] \in E : u \in V \setminus V_B \wedge v \in V_B\}$ be the set of *frontier links*, which are the safe links connecting safe nodes to black holes. We define the safe portion of G as $G_S = (V_S, E_S)$ where $V_S = V \setminus V_B$ is the set of safe nodes and $E_S = E \setminus (E_B \cup E_I \cup F_B)$ is the set of safe links.

A summary of notation from the model is provided in Table 2.1. An example network model graph is shown in Figure 2.1.

In Chapter 4, we look at dangerous graph exploration by a team of agents in the network model. We define the problem as follows:

Definition 2.7 (Dangerous graph exploration (DGE) problem)**.** *The dangerous graph exploration problem is for a team of agents to explore a network and within finite*

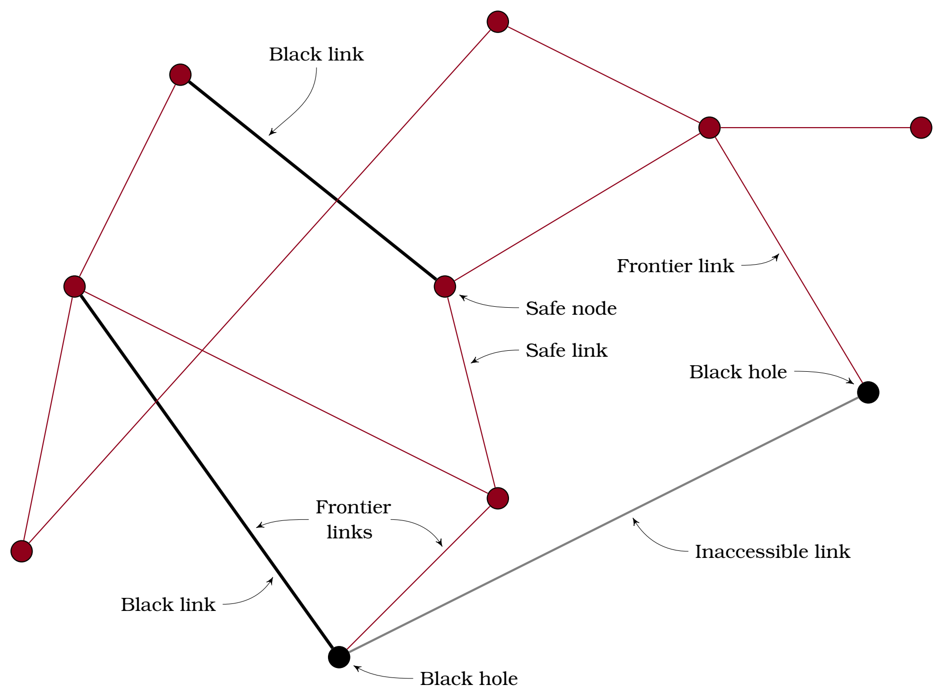

Figure 2.1: Example of a network model graph based on Figure 1.1.

time to construct a map of all the safe nodes V_S, where all links of the frontier F_B and the accessible black links $E_B \setminus E_I$ are indicated. The map is unambiguous if every safe edge is marked as such. We say that the problem is solved if at least one agent survives, and all surviving agents within finite time terminate having such a map.

Some obvious limitations follow from the problem definition. In particular, for the problem to be solvable, clearly the safe portion of the network G_S must be connected. Furthermore, since at least one agent must survive, the number of agents k must be greater than $f = |F_B| + 2|E_B \setminus (E_I \cup F_B)|$. Thus, for the solution to this problem presented in Chapter 4, we assume that G_S is connected and $k \geq f+1$. We also assume, to detect termination, that the number $n_S = |V_S| = |V \setminus V_B|$ of safe nodes is known.

In Chapter 5, we look at dangerous graph exploration by a team of agents in the network model, but this time we also allow an adversary to delete links in the network during the execution of the algorithm. Due to the deletions, making an accurate map of the network is not feasible. Instead, we want to make sure that dangerous nodes and links are marked locally as such. We define the problem as follows:

Definition 2.8 (Dangerous graph exploration with link deletions (DGE-LD) problem)**.** *The dangerous graph exploration with link deletions problem is for a team of agents to visit every accessible link in a network and, within finite time, to mark locally all frontier links F_B and accessible black links $E_B \setminus E_I$ as dangerous. During the execution of the algorithm, an adversary can delete without leaving a discernible trace any link as long as no agents are traversing it. We say that the problem is solved if, within finite time, at least one agent survives, all accessible links have been visited, and all frontier links and accessible black links are marked locally as such.*

The same limitations follow from the problem definition. The problem is only solvable if G_S remains connected, so the adversary's deletions cannot violate this assumption. There must also be enough agents. The prohibition against deleting links with agents traversing them avoids giving the adversary the power to remove agents from the network, so our solution in Chapter 5 also uses $k \geq f+1$ agents. At termination, at least one agent must survive, all accessible links must be visited, and all the dangerous links must be marked.

2.3 Subway model and problems

In this section, we describe the subway model, a model of a dynamic network based on real-life urban subway systems. We describe the network, the agents working in it and their abilities, and the nature of the dangerous elements. We then describe the problem for which we provide solutions, the additional assumptions required to make them solvable, and discuss the subway-model-specific complexity measure used to assess their efficiency.

We consider a set C of n_C *carriers* that move among a set S of n_S *sites*. A carrier $c \in C$ follows a route $R(c)$ between all the sites in its *domain* $S(c) = \{s_0, s_1, \ldots, s_{n_S(c)-1}\} \subseteq S$, where $n_S(c) = |S(c)|$. A carrier's *route* $R(c) = \langle r_0, r_1, \ldots, r_{l(c)-1} \rangle$ is a cyclic sequence of *stops*: after stopping at site $r_i \in S(c)$, the carrier moves to $r_{i+1} \in S(c)$, where all operations on the indices are modulo $l(c) = |R(c)|$, called the *length* of the route. Carriers move *asynchronously*, taking a finite but unpredictable amount of time to move between stops. We call a route *simple* if $n_S(c) = l(c)$. A *transfer site* is any site that is in the domain of two or more carriers.

A carrier's route $R(c) = \langle r_0, r_1, \ldots, r_{l(c)-1} \rangle$ defines an edge-labelled directed multigraph $\vec{G}(c) = (S(c), \vec{E}(c), \lambda(c))$, called a *carrier graph*, where $S(c)$ is the set of nodes, $\vec{E}(c)$ is the set of edges[1], and $\lambda(c)$ is the set of labels, and where there is an edge labeled $(c, i+1)$ from r_i to r_{i+1}, and the operations on indices and inside labels are modulo $l(c)$. The entire network is then represented by the edge-labelled directed multigraph $\vec{G} = (R, \vec{E}, \lambda)$, called a *subway graph*, where $R = \cup_{c \in C} R(c)$, $\vec{E} = \cup_{c \in C} \vec{E}(c)$, and $\lambda = \{\lambda(c) : c \in C\}$. Associated to the subway graph is the *transfer graph* of $\vec{G}$, which we define as the edge-labeled undirected multigraph $H(\vec{G}) = (C, E_T)$ where the nodes are the carriers and, $\forall c, c' \in C, c \neq c', s \in S$, there is an edge between c and c' labeled s iff $s \in S(c) \cap S(c')$, i.e., s is a transfer site between c and c'. In the following, where no ambiguity arises, we omit the edge labels in all graphs.

Working in the network is a team A of $k = |A|$ computational *agents* that start

[1]Although self-loops have no real-world equivalents, we allow them for theoretical completeness.

at unpredictable times. The agents move opportunistically around the network using the carriers. An agent can move from a carrier to a site (*disembark* at a stop) or from a site to a carrier (*board* a carrier), but not from one carrier to another directly. An agent on a transfer site can board any carrier stopping at it. When travelling on a carrier, an agent can count the number of stops that the carrier has passed, and can decide whether or not to disembark at the next stop.

The agents have the following additional properties. The agents are *asynchronous* in that they take a finite but unpredictable amount of time to perform computations. All agents execute the same protocol. Agents communicate with each other using shared memory in the form of a *whiteboard*, available at each site in Chapter 7 and each carrier in Chapter 8, which is accessed in fair mutual exclusion. In Chapter 7, the agents start at varying times co-located on the same site, while in Chapter 8, the agents start at varying times scattered on possibly different sites throughout the network. The site on which an agent starts is called its *homebase*.

The network in which the agents operate has elements, sites, that are dangerous to the agents. We define them as follows:

Definition 2.9 (Black hole)**.** *A black hole is a site that eliminates agents disembarking on it without leaving a discernible trace.*

Notice that a black hole in the subway model does not affect the carriers.

Among the sites, there are $n_B < n_S$ *black holes*. Let $\gamma(c) = |\{i : r_i \in R(c)$ is a black hole$\}|$ be the number of black holes among the stops of c; and let $\gamma(\vec{G}) = \sum_{c \in C} \gamma(c)$, called the *faulty load* of subway graph $\vec{G}$, be the total number of stops that are black holes. Where no ambiguity arises, we use $\gamma(\vec{G})$ and γ interchangeably. Note that, since a site might appear more than once in a route and in more than one route, in general $\gamma(\vec{G}) \geq b(\vec{G})$, where $b(\vec{G})$ is the number of black hole sites in $\vec{G}$.

Since agents rely on the carriers to move themselves around the network, we redefine Property 2.2 for the subway model as follows:

Property 2.10. *It is impossible to distinguish between slow computations by an agent and a black hole.*

In other words, we cannot tell if an agent has not boarded a carrier because it is still performing calculations on the site or because the site is a black hole.

A summary of notation from the model is provided in Table 2.2. See Section 7.2.2 for an example subway network (Table 7.2), including a view of the subway graph (Figure 7.1), its associated transfer graph (Figure 7.2), and its carrier graphs (Figure 7.3).

In Chapter 6, we look at basic limitations on exploration in the subway model. We define the problem as follows:

Definition 2.11 (Subway exploration (EXP) problem)**.** *The subway exploration (EXP) problem is for every stop in a network to be visited by an agent within finite time. We say that the problem is solved if every stop is visited by at least one agent and all agents enter a terminal state.*

In Chapters 6, 7 and 8, we look at black hole search in the subway model. Specifically, we want to find those sites that are black holes by searching the stops of all carriers. We define the problem as follows:

Definition 2.12 (Subway black hole search (BHS) problem)**.** *The subway black hole search (BHS) problem is for a team of agents to explore a subway graph and within*

Notation	**Description**
C	Set of all carriers
S	Set of all sites
R	Set of all routes
n_C	Number of carriers
n_S	Number of sites
l_R	Length of longest route
$S(c)$	Domain of carrier c
$R(c)$	Route of carrier c
$n_S(c)$	Number of sites in carrier c's domain
$l(c)$	Length of route c
s_i	The ith site in some carrier's domain
r_i	The ith stop on some carrier's route
$\vec{G}$	Subway graph
$H(\vec{G})$	Transfer graph associated with $\vec{G}$
A	Set of agents
k	Number of agents
n_B	Number of black holes
$\gamma(c)$	Number of black hole stops on carrier c's route
$\gamma(\vec{G})$ or γ	Faulty load (number of black hole stops)

Table 2.2: Notation from the subway model

finite time to determine which stops are at black hole sites. We say the problem is solved if at least one agent survives, all surviving agents enter a terminal state, and all surviving agents know which stops are black holes.

Some limitations follow from the definitions of both the Exp and Bhs problems in the subway model. We make the following assumptions that we prove in Chapter 6 are necessary for the Exp and Bhs problems to be solvable in the subway model. Each carrier is labelled with a distinct id. The agents know the number of carriers n_C. Each transfer site is labelled with the number of carriers stopping there.

Without loss of generality, since the agent knows the number of carriers and can wait for all the carriers to pass by, we assume that each transfer site is also labelled with the ids of the carriers. Finally, we assume that the standard assumptions for BHS apply: an agent's homebase is a safe site, the transfer graph must stay connected when the black holes are removed, there must be more agents than all the faulty stops ($k > \gamma$), and the agents must know the number of faulty stops γ.

We make additional assumptions depending on whether the whiteboards are at the sites or on the carriers. For site whiteboards, we make the following assumptions proven in Chapter 7. Each carrier is labelled with the length of its route. Once disembarked, an agent can board a carrier at the same point in its route. For carrier whiteboards, we make the following assumption in Chapter 8. There is a function $r_{curr}(c)$ that returns a distinct id for the current stop when the agent is on carrier c. Without loss of generality, since the agent can record every stop during the carrier's traversal of the route, we assume that there is a second function $R(c)$ that returns the set of all stops on the route. The same function can be used to determine the length of the carrier's route.

Measuring complexity: As in traditional mobile agent algorithms, the basic cost measure used to evaluate the *efficiency* of a BHS solution protocol in the subway model is the *size* of the team, that is, the number k of agents needed by the protocol. To solve the BHS problem, it is obviously necessary to have more agents than the faulty load of the network, i.e. $k > \gamma$. A solution protocol is *agent optimal* if it solves the BHS problem for $k = \gamma + 1$.

The other cost measure is the number of *carrier moves*, which is a combination of the traditional mobile agent algorithm metric of agent moves with the cost of waiting imposed on the agent by its use of opportunistic movement in the network. When an agent is riding on a carrier, which we call *agent moves*, or waiting for a carrier, which we call *agent waits*, we count each move made by that carrier as a carrier move for the agent. A solution protocol is *move optimal* in the worst case if the total number of carrier moves incurred by all agents in solving the BHS problem is the best possible.

CHAPTER 3

Related Work

In this chapter, we look at the literature that has led up to the problems we investigate in this thesis. We start by looking at specific distinguishing assumptions or models that have a significant impact on the approach a paper's authors may take towards a problem. We then look at literature on the exploration problem, black hole search problem and its variants, computing in dynamic networks, and other problems solved in the mobile agent model. We present literature related to the algorithmic view of mobile agents discussed in the last chapter. We note throughout where a particular paper has had a direct impact on the work of this thesis, which we also briefly mention below. The goal of this chapter is to give a broad overview of research into the mobile agent model with a focus on the black hole search problem.

Our work on DGE problem in Chapters 4 and 5 was inspired by the work on BHS in a network of unknown topology presented in one of the first papers on the subject (Dobrev et al., 2002, 2006c), which is discussed in Section 3.3.1. Our work on the BHS problem in the subway model in Chapters 7 and 8 was inspired by work

Notation	**Definition**
n	Number of nodes
m	Number of edges
k	Number of computational entities
D	Diameter of graph
r	Radius of graph
Δ	Maximum degree of all nodes
Δ_o	Maximum out-degree of all nodes in a directed graph
δ	Degree at every node
δ_o	Out-degree at every node in a directed graph
d	Deficiency of a directed graph

Table 3.1: Notation used for related work discussion.

on the Exp problem in periodically varying graphs introduced in (Flocchini et al., 2009b), which is discussed in Section 3.4.

Each paper discussed uses slightly different notation for expressing the cost of the complexity. For ease of reading, we convert these measures throughout this chapter into the same notation for complexity shown in Table 3.1.

3.1 Distinguishing assumptions

A model is a set of assumptions under which problem is solved. Certain assumptions are significant enough to themselves be referred to as models. Changes in these assumptions can make a marked difference in the difficulty of finding a solution, the complexity of the resulting solution, or even change the entire approach taken to the solution. We have already mentioned one of these significant assumptions: a solution to a problem in the message passing model would most likely be completely different from a solution to the same problem in the mobile agent model.

In the mobile agent model, there are a number of these significant assumptions that serve to distinguish solutions from each other. Computations and movements can be asynchronous or synchronous. Agents can start co-located on the same homebase or scattered throughout the network. They can communicate with each other using whiteboards, various types of tokens, or even face-to-face. Various elements of the model such as nodes and agents can have distinct ids or be anonymous. Agents can have memory or be oblivious. The network itself can be arbitrary or have a specific topology known to the agents.

Changes in any of these assumptions often change the nature of the problem being solved and, therefore, change the approach taken to the solution. Although not an exhaustive list, the assumptions mentioned show up frequently in the literature we review in the rest of this chapter. We briefly discuss them here and their implication to developing solutions.

Asynchronous/synchronous: One of the assumptions that has the most significant effect on how a problem is solved is the timing of computations and movements in the model. In the *asynchronous* model, all computations and movements take a finite but unpredictable amount of time. In the *synchronous* model, all agents have access to the same global clock, all computations are considered to be instantaneous, and all movement from a node to a neighbouring node takes exactly one time step. These two models are used for both message passing and mobile agent algorithms. They often call for the use of completely different techniques for solving problems. For instance, while it is impossible to determine if a network contains a black hole in the asynchronous model (Corollary 2.3), it is quite possible to do so in the synchronous model.

The asynchronous model arises naturally from the axiom of finite communication delays (Santoro, 2007), which states that in the absence of failures, communications delays are finite. Without this guarantee, very few problems, if any, would be solvable. As a result, the asynchronous model can be seen as the default model for the timing of computations in distributed networks.

Co-located/scattered agents: Where the agents start the algorithm can also have a significant effect. In the *co-located* agent model, all agents start on the same homebase. In the *scattered* agent model, the agents can start anywhere in the network, even co-located. Obviously, the impact of the choice of model depends on the problem being solved. For instance, the problem of gathering agents together on the same node would be trivial in the co-located agent model.

These assumptions certainly affect the problems of exploration, map construction, and black hole search that are the focus of this thesis. The co-located agent model is easier to deal with from an algorithmic point of view and much of the work in exploration and black hole search has focussed on this model. The scattered agent model requires the agents to somehow meet each other and/or coordinate their activities. Three of the solutions presented in this thesis are in the more difficult scattered agent model and one in the co-located agent model.

Whiteboards/tokens: Coordinating the activity of a team of agents requires some assumption of how the agents communicate with one another. Two of the most common assumptions in the study of the black hole search problem are whiteboards and tokens.

In the *whiteboard* model, each node has shared memory that is accessed in fair mutual exclusion by the agents. The whiteboards are not usually limited in their size. The fair mutual exclusion property, combined with the distinct id model discussed next, allows for certain useful properties, such as ids for co-located agents or first-in, first-out (FIFO) links, to be generated by the agents during the execution of the algorithm rather than assumed to already exist in the network. The whiteboard model is the default model for the development of mobile agent algorithms when the focus is on other aspects of the system, such as the effects on exploration of adding black holes to the system. It is the least limiting of the communication models discussed here and is used in all four of the solutions presented in this thesis.

In the *token* model, agents use tokens to mark nodes. The tokens are often assumed to be indistinguishable, meaning that there is no way of telling which agent dropped the token. The token can be dropped on the node, sometimes called the node or pure token model, or on a node's port, sometimes called the port token model. Depending on the model, agents have one token each, an unlimited supply of tokens, or something in between.

Other inter-agent communication models include the face-to-face and blackboard models. In the *face-to-face* model, two agents can exchange any amount of information when they are co-located on the same node. The face-to-face model generally only appears in solutions in the synchronous model or its variants. In the asynchronous model, when we play an adversarial game to determine the worst case performance of an algorithm, the adversary is often given the power to decide on the timing of the agents' movements and so can ensure that no two agents are ever on the same node at the same time. In the *blackboard* model, the agents communicate at the nodes using shared memory similar to the whiteboard model; however, in the blackboard case, all nodes share access to the same globally accessible memory. The blackboard model has been studied extensively as shared memory computing, but rarely in the mobile agent model.

Distinct ids/anonymous: Some problems become much harder if nodes or agents cannot be distinguished from one another. In the *distinct id* model, each node, agent, or both have ids that come from a totally ordered set. In the *anonymous* model, nodes and agents have no ids, but a node's port are still labelled in some way because of the axiom of local orientation (Santoro, 2007). Since the port labels can be used to distinguish one node from another, worst case scenarios in the anonymous model tend to involve graphs with a high degree of symmetry such as Cayley graphs. The anonymous model makes even simple problems much more difficult. Determining if an agent has already visited a specific node can take a large number of moves. At least one paper (Barrière et al., 2003) looks at the *incomparable id* model, where each node has an id but there is no ordering or only a partial ordering to the labels (e.g., Is a star greater than a square?). In general, mobile agent algorithms assume distinct ids and the anonymous and incomparable id models are assumed when it is the specific focus of the research. All the solutions presented in this thesis use the distinct id model.

Oblivious agents: Mobile agents are generally assumed to carry their algorithm and a possibly unbounded amount of persistent memory with them. Depending on the algorithm, the memory can be used to store the state of the agent or the map it is constructing. In the *oblivious* model, the agent has no memory. Each time the agent goes to perform an action, it can only react to its current situation. Solutions in the oblivious model often have the benefit of being fault-tolerant and self-stabilizing, meaning that changes in the network have little effect on the agents. The oblivious model tends to be its own focus of research. In all the solutions presented in this thesis, we assume that the agents have enough persistent memory to complete their task without being constrained.

Arbitrary/specific topology: The assumption of a specific topology can have a significant effect on how an algorithm is developed. Although not generally referred to as models, assumptions about topology do distinguish classes of solutions from one another. Agents working in specific topologies, such as trees, rings, stars, hypercubes, meshes/grids, tori, Cayley graphs, and complete graphs, can take advantage of the characteristics of the specific topologies to make their solutions easier. Whether a network has directed or undirected links is also a consideration. Since many of these topologies are common ways of connecting together networks, they are often the focus of specific research and we use topology to categorize the related work we discuss in subsequent sections.

The deletion of even a single link in many of these specific topologies is enough to make it something else. Deleting a link in a tree disconnects it, while deleting a link in a ring turns it into a tree. As a result, research into dynamic networks tends to focus on arbitrary networks where no particular topology is assumed. In certain models, such as the asynchronous model, additional assumptions about connectivity must be made to ensure that a specific problem remains solvable in the model. All of the solutions presented in this thesis assume networks with arbitrary topology. The solutions in the network model assume undirected links, while the solutions in the subway model assume directed links.

3.2 Exploration

Exploration is the problem of having a computational entity (e.g., robot, finite automaton, mobile agent) cross every edge in a network. If there is more than one agent, the problem becomes one of having every edge crossed by an agent. Some papers also vary the problem by requiring the agents only to visit every node. As we noted in the last chapter, the basic problem of black hole search is to explore a network where there are elements that are dangerous to the agents. In this section, we group the papers by topology: early work on maze solving and regular planar graphs followed by later work on directed graphs, trees and rings, and undirected graphs. For the most part, we are concerned with distributed solutions to the exploration problem. These are solutions that work with information that is available from within the network, sometimes referred to as *online* solutions. We mention below some work on variants of the travelling salesman problem, which is a way of looking at the exploration problem from the centralized perspective, sometimes referred to as *offline* solutions.

There are a number of variations of the exploration problem. *Perpetual exploration* requires the agent or agents to solve the problem completely over and over again. *Exploration with stop* requires the agent or agents to recognize when the problem has been solved and stop. *Exploration with return* requires the agent or agents not only to recognize when the problem has been solved but also to return to the node from which it or they started.

There are a number of measures that have been used for the complexity of solutions to the exploration problem. The two most common are the number of edges traversed or *moves* and the competitive ratio of the moves, which is the ratio of the number of moves used by the proposed solution compared to the optimal number of moves for the same graph. The competitive ratio can be thought of as the cost of the online solution compared to the optimal solution that is calculated offline. Since we are mainly considering online solutions, where the calculation is done by entities in the system, a number of papers give lower bounds on how close such solutions can get to the optimal offline solution.

The type of computational entity doing the exploration varies from paper to paper. The first paper related to exploration is actually about a physical maze-solving machine built by Shannon (1951). The rest of the papers refer to *travelling salesmen*: Averbakh and Berman (1996); *finite automata*: Blum and Kozen (1978); Fraigniaud et al. (2005); *robots*: Albers and Henzinger (2000); Ambühl et al. (2011); Awerbuch et al. (1999); Bender et al. (1998); Bender and Slonim (1994); Deng and Papadimitriou (1990, 1999); Dessmark and Pelc (2004); Dudek et al. (1991); Dynia et al. (2007); Flocchini et al. (2007b, 2008b, 2010a); Fraigniaud et al. (2004, 2006a, 2005, 2006b); Gasieniec et al. (2007); Panaite and Pelc (1999); and *mobile agents*: Ambühl et al. (2011); Das et al. (2007, 2005); Fraigniaud et al. (2004, 2006a); Gasieniec et al. (2007). Whatever the computational entities are called, they generally have similar capabilities and resources including having some memory, following an algorithm, moving around autonomously, and having some way of getting information from the network and/or each other. Early exploration solutions use pebbles or markers to mark where they have been, which were later formalized as the token model. Later work in the mobile agent model uses the whiteboard model.

3.2.1 Maze solving and regular planar graphs

The earliest papers related to exploration deal with the problem of solving a maze. The first came in 1951 when Shannon (1951) created a machine that could solve a 5×5 maze deterministically. The maze was defined by reconfigurable wires while the "agent" was a sensing head that worked similar to a plotter head. Not only did the machine solve the maze using an algorithm similar to depth-first search, it could also remember its solution and detect if the maze had been reconfigured while it was running. It accomplished all these things using only 75 relays.

By 1978, the problem became the searching of the maze, that is the visiting of every cell, which obviously leads to a solution to the maze. Blum and Kozen (1978) look at finite automata searching a maze and the relation between rectilinear mazes and regular planar graphs. The authors show that although mazes and regular planar graphs are related, the compass directions in a maze give a huge advantage. They show that it is possible to solve any maze with an automaton with a counter, an automaton with two pebbles, or two automata working together. Pebbles are an early version of the token model and some research assumes that they have two distinct colours, black and white. The authors then show that even in one of the simplest planar graphs, a finite cubic graph embedded on the plane, the problem could not be solved by one, two, or three automata.

The work of Blum and Kozen (1978) was one of many papers that used aspects of what was later formalized as *sense of direction*. The concept as a whole was initially reported on by Santoro (1984) in the SIGACT News in 1984 and formalized by Flocchini, Mans, and Santoro (1998) in 1998. We see sense of direction again in the black hole search literature presented in Section 3.3.

3.2.2 Directed graphs

When the attention of the research community turned to the exploration of graphs in general in the early 1990s, the focus initially was on the exploration of strongly-connected directed graphs. Chapters 6, 7, and 8 deal with a dynamic version of strongly-connected directed graphs.

Deng and Papadimitriou (1990) look at exploration of a strongly-connected directed graph. A journal version of the paper was published almost a decade later (Deng and Papadimitriou, 1999). A single robot develops a map as it goes along that allows it to unambiguously recognize nodes that it has visited before. The problem being studied is the exploration with stop of all edges. The authors find that the cost of exploring a directed graph is related to the deficiency of the graph. A directed graph's *deficiency* is the minimum number of edges that must be added to the graph to make it Eulerian. An *Eulerian graph* is one where there is a tour of all the edges that crosses each edge exactly once. The authors show that a graph with deficiency d can be explored in $O(2^{O(d \log d)}) \cdot m$ moves, while a graph with deficiency $d = 1$ can be explored in $4m$ moves.

Bender and Slonim (1994) look at the searching of strongly-connected directed graphs by two robots where every node has δ outgoing edges. The nodes are anonymous and so are indistinguishable from one another and the robots move and compute in the synchronous model. The authors present a solution based on the idea of a homing sequence, which is a polynomial-length random string that is shared by the robots. Given a sufficiently long homing sequence, the algorithm allows the robots to explore any graph in $O(\delta^2 n^5)$ moves.

Bender, Fernández, Ron, Sahai, and Vadhan (1998) study the use of pebbles by a robot exploring and mapping a strongly-connected directed graph. The nodes are indistinguishable from each other and the mapping of the network suggests that the robot stops when the exploration is complete. The authors find that if the robot knows an upper bound on the number of nodes in the graph, n, then the robot can successfully explore the network with just one pebble. However, if the robot does not know an upper bound then $\Theta(\log\log n)$ pebbles are necessary and sufficient to solve the problem.

Albers and Henzinger (2000) look at the problem of a robot exploring a strongly-connected directed graph. The authors prove lower bounds for a number of different approaches to the problem, including $2^{\Omega(d)}m$ moves for locally greedy, depth-first search, and breadth-first search, and $d^{\Omega(\log d)}m$ moves for a generalized greedy approach, where d is the deficiency of the graph. They present their own algorithm based on (Deng and Papadimitriou, 1990, 1999) that achieves an overall complexity of $O(\min\{nm, dn^2+m\})$ moves, a significant improvement over the $O(2^{O(d\log d)})m$ achieved by Deng and Papadimitriou (1990).

Fraigniaud and Ilcinkas (2004) look at the exploration of a strongly-connected directed graph that has a maximum out-degree of Δ_o. The authors distinguish between the robot model, which they define as a finite automaton with constant size memory and a source of pebbles, and the agent model, which they define as a computational entity with constant size memory and access to node whiteboards. In the robot model, they prove a lower bound of $\Omega(n\log\Delta_o)$ bits of memory for perpetual exploration of a directed graph. They give two algorithms for exploration with stop in the robot model. The first uses $O(n\Delta_o\log n)$ bits of memory with 1 pebble and is $O(\log n)$ away from optimal in constant degree digraphs; however, it runs in exponential time. The second algorithm uses $O(n^2\Delta_o \log n)$ bits of memory with $O(\log\log n)$ pebbles and runs in polynomial time. We know from Bender et al. (1998) that the second algorithm uses the optimal number of pebbles. In the mobile agent model, they show that exploration with stop is not possible if the mobile agent is oblivious, that is it has no memory. Given a constant-size memory, however, they provide a solution to exploration with return that uses $O(\log \Delta_o)$-sized whiteboards at each node.

Flocchini, Mans, and Santoro (2009b) look at exploration by a mobile agent in a class of dynamic networks called periodically varying graphs (PV graphs). PV graphs are loosely modelled on a bus system where "carriers" move synchronously between the nodes of the network on a route. The carrier is used as an abstraction for the cyclical appearance of edges along the route. The mobile agent explores the network by riding the carrier and moving from carrier to carrier when they meet at a node common to both carriers' routes. The authors present two move optimal algorithms for a single agent searching the network when the routes lengths are homogeneous and heterogeneous. This paper was the inspiration for Chapters 6, 7, and 8.

3.2.3 Trees and rings

Exploration has been studied in both trees and rings where knowledge of the structure of the network can be used to the advantage of the exploring salesman, robot, or agent.

Averbakh and Berman (1996) look at the travelling salesman problem for two salesmen in a tree. The problem is one of collective exploration where the goal is the minimize the distance covered by each salesman. The problem is known to be

NP-hard, meaning that there is no known polynomial time solution, so the focus is on finding an algorithm that has the lowest competitive ratio over the optimal solution. The authors give a solution for salesmen starting on the same node that has a competitive ratio of $\frac{1}{3}$, meaning that the salesmen cover a distance that is $\frac{1}{3}$ longer than the optimal solution. They give a solution for salesmen starting on different nodes that has a competitive ratio of $\frac{1}{2}$.

Fraigniaud, Gasieniec, Kowalski, and Pelc (2006a) (conference version (Fraigniaud et al., 2004)) look at collective exploration of a tree by a team of k mobile agents, which they also refer to as robots. The agents move synchronously and the goal is to explore the tree as quickly as possible. Because of the synchronous movement, the problem is similar to the travelling salesman problem from (Averbakh and Berman, 1996). Similarly, optimal collective exploration by k agents is NP-hard. The authors are interested in looking at how communications between the agents affect the competitive ratio of the algorithms. They present an algorithm where the agents can communicate with each other without restriction at each time step. The algorithm achieves a competitive ratio of $O\left(\frac{k}{\log k}\right)$ running time larger than optimal. They present another algorithm where the agents are only allowed to communicate using whiteboards at each node. In this case, the algorithm achieves a competitive ratio of $O\left(D+\frac{n}{\log k}\right)$ running time larger than optimal. They also prove a lower bound on the competitive ratio of $\Omega(k)$ for any algorithm in this model.

Dessmark and Pelc (2004) give solutions that cover a number of different graph types including trees. We present the results in Section 3.2.4 on the exploration of undirected graphs.

Diks, Fraigniaud, Kranakis, and Pelc (2004) look at the memory needed for different types of exploration by a robot in a tree. They give an algorithm for perpetual exploration that takes $O(\log \Delta)$ bits of memory, where Δ is the maximum degree of nodes in the graph. They give a lower bound for exploration with stop of $\Omega(\log\log\log n)$ bits of memory for some bounded degree trees and a lower bound for exploration with return of $\Omega(\log n)$ bits of memory. Finally, they present an algorithm for exploration with return of $O(\log^2 n)$ bits of memory for all trees.

Fraigniaud, Ilcinkas, and Pelc (2006b) look at the minimum amount of information available to a robot from an oracle in order to efficiently explore a tree. Depth-first search gives an upper bound of 2 on the competitive ratio for exploration. The authors are interested in the minimum number of bits of information needed to get a ratio below 2. They find that the robot needs access to an oracle with roughly $\log\log D$ bits of information.

Flocchini, Ilcinkas, Pelc, and Santoro (2007b) look at exploration by k robots in an undirected ring. The robots are identical, oblivious (have no memory), asynchronous, and cannot communicate with one another. They can, however, see the location of all the other robots in the ring. The authors show that the problem is unsolvable in general when n and k are not co-prime. They also prove that when n and k are co-prime ($\gcd(n,k) = 1$) and $k \geq 17$ then exploration is always possible in finite time. The authors prove a lower bound of $O(\log n)$ agents that are needed to explore a sufficiently large ring.

Flocchini, Ilcinkas, Pelc, and Santoro (2010a) (conference version (Flocchini et al., 2008b)) look at the problem of exploration with stop by asynchronous oblivious robots in a tree. Like the last paper, the agents cannot communicate with one another but can take a snapshot of the location of all the other robots in the tree. The authors find that there are n-node trees with $\Delta \geq 4$ that require $\Omega(n)$ agents to solve exploration successfully. However, if a tree has $\Delta = 3$ then they provide a solution that uses only $O\left(\frac{\log n}{\log\log n}\right)$ robots. They show their solution is optimal by proving a lower bound of $\Omega\left(\frac{\log n}{\log\log n}\right)$ agents. The source of the complexity is apparently the symmetry of the tree and the authors show that when a tree is not symmetric, using a specific definition, a maximum of 4 agents is required for exploration.

Ambühl, Gąsieniec, Pelc, Radzik, and Zhang (2011) (conference version (Gasieniec et al., 2007)) look at improving the results on exploration with return presented in (Diks et al., 2004). The authors present a solution that uses only $O(\log n)$ bits of memory, improving on the previous $O(\log^2 n)$ result. The algorithm works even if the model of how ports are identified is weakened so that the ports are only ordered circularly but have no port numbers.

3.2.4 Undirected graphs

Research into the exploration of undirected graphs also started in the early 1990s, but really took off in the late 1990s and the 2000s. The results presented in Chapters 4 and 5 deal with connected undirected graphs.

Dudek, Jenkin, Milios, and Wilkes (1991) look at the problem of a robot exploring a connected undirected graph. The robot is unable to distinguish the nodes of the graph from one another, but does have one or more markers that it can use to mark some nodes. The authors present a solution that allows a robot with a single marker to explore any graph in $O(nm)$ moves.

Awerbuch, Betke, Rivest, and Singh (1999) look at the problem of exploring a connected undirected graph where the robot doing the exploration must periodically return to their starting node. They refer to this problem as *piecemeal exploration* and the premise is that the robot must refuel or be checked before it can continue with the search. The authors assume that the robot can distinguish previously explored nodes and edges and that the bound B on the number of moves a robot can make during any foray into the graph is related to the radius of the graph. They present a simple algorithm that solves the problem in $O(m+n^{1.5})$ moves and a more complicate version that solves the problem in a near linear $O(m+n^{1+o(1)})$ moves. These solutions improve on the previous best known solution of $O(m+n^2)$ moves.

Panaite and Pelc (1999) study a robot exploring a connected undirected graph. The robot can distinguish previously explored nodes. The authors are interested in the penalty incurred by an algorithm solving the problem. The obvious lower bound on exploration is m moves since every edge must be explored. The penalty is the number of moves in excess of m. The authors show that "natural" approaches, such as depth first search and the greedy algorithm, have a penalty of $O(m)$ moves. They give a solution to the problem that only has a penalty of $O(n)$ moves.

Dessmark and Pelc (2004) look at a robot exploring line graphs, trees, and general undirected graphs. The authors are interested in the cost, in terms of competitive ratio, of the robot's knowledge of the topology. They show that a robot with no map can solve all graphs with a competitive ratio of 2, which is optimal for general graphs. A robot with an unanchored map, an isomorphic map that does not show the starting node, can solve line graphs in an optimal competitive ratio of $\sqrt{3}$, trees in <2 with a lower bound of $\sqrt{3}$, and an optimal 2 for general graphs. A robot with an anchored map, an isomorphic map showing the starting node, can solve line graphs in an optimal competitive ratio of $\frac{7}{5}$, trees in an optimal $\frac{3}{2}$, and an optimal 2 for general graphs.

Das, Flocchini, Kutten, Nayak, and Santoro (2007) (conference version (Das et al., 2005)) look at labelled map construction by a team of k mobile agents starting co-located on the same node in a connected undirected graph. The labelled map construction problem is where the agents collectively explore the graph and terminate with each agent having a map with the same labelling of the graph. The agents communicate with one another using whiteboards. The authors give a algorithm that solves the problem in $O(km)$ moves for graphs where n and k are co-prime ($\gcd(n, k) = 1$) and the agents know either n or k. They give a second algorithm for general graphs that solves the problem in $O(m \log k)$ moves when the agents know n and k, or $\gcd(n, k)$ and either n or k.

Fraigniaud, Ilcinkas, Peer, Pelc, and Peleg (2005) look at a finite automaton or robot exploring a connected undirected graph. The authors are interested in the relation between the number of states (the size of the robot's memory) and the degree of the graph. They show that for any K-state robot and any number $x \geq 3$, there is a planar graph of max degree $\Delta = x$ and $n = K + 1$ nodes that cannot be explored. The authors also give a lower bound on the exploration of all graphs of diameter D and maximum degree Δ by a robot with a memory of size $\Omega(D \log \Delta)$ and prove that the bound is tight by providing an algorithm.

Dynia, Lopuszański, and Schindelhauer (2007) look at robots exploring a connected undirected graph. The authors are particularly interested in the lower bound on the competitive ratio for exploration. They show that the lower bound on the competitive ratio for exploration of an unknown graph by k robots is $\Omega\left(\frac{\log k}{\log \log k}\right)$ even if there is global communication between the robots.

3.3 Black hole search

In this section, we look at the literature that has been published on black hole search, dangerous graph exploration, and related problems. A black hole is a node that eliminates agents arriving at it without leaving an observable trace. The black hole search problem is to explore a network such that, when the surviving agents terminate, they know the location of the black hole (or black holes). Often, black hole search involves map construction. The dangerous graph exploration problem extends the black hole search problem to include black links, links that act like black holes.

The presence of a dangerous element, like a black hole, changes how we have to approach exploration of the network. Dobrev, Flocchini, Prencipe, and Santoro (2001, 2007a) introduce the black hole search problem and establish some of the basic problems that must be addressed. For instance, in an asynchronous network a black hole is indistinguishable from slow links, so the agents must know something about the existence and number of black holes in the network in order to be able to terminate. As well, because agents are eliminated without a discernible trace, the agents must work in teams of at least two and use techniques like cautious walk in order to bound the number of agents that are eliminated. We discuss these properties and the cautious walk technique in more detail in Chapter 2.

Flocchini and Santoro (2006) provide a survey of black hole search results up until 2006. They also cover the intruder capture/decontamination problem, which is related to the more general graph search problem.

There are a number of papers that deal with black hole search as an optimization problem. The problem has been shown to be NP-hard by reduction to the Hamiltonian cycle problem (Czyzowicz et al., 2006). These papers develop solutions that approximate, in polynomial time, as closely as possible the optimal solution and are included here for completeness (Czyzowicz et al., 2006, 2007; Klasing et al., 2005, 2007, 2008).

3.3.1 Asynchronous arbitrary networks

Dobrev, Flocchini, Prencipe, and Santoro (2006c) (conference version Dobrev et al. (2002)) look at black hole search in connected undirected graphs. A network with a black hole is connected if remains connected when the black hole is removed. The authors are interested in the effect of topological knowledge on the cost of finding a single black hole in an arbitrary network. They look at three situations: topological ignorance, topological ignorance where the network has sense of direction labelling, and complete topological knowledge. For topological ignorance, the authors prove a lower bound of $\Omega(n^2)$ moves and present a solution that uses $\Delta + 1$ agents and an optimal $O(n^2)$ moves. For topological ignorance with sense of direction labelling, which allows an agent to tell whether it has visited a specific node before, they prove a lower bound of $\Omega(n^2)$ moves and present a solution that uses two agents and takes an optimal $O(n^2)$ moves. For complete topological knowledge, where the agents can tell unambiguously where they are in the map of the network, they prove a lower bound of $\Omega(n \log n)$ moves and present a solution that uses two agents and takes an optimal $O(n \log n)$ moves. The work on topological ignorance in this paper was the inspiration for the work presented in Chapters 4 and 5.

Dobrev, Flocchini, and Santoro (2004) look at black hole search in connected undirected graphs when the agents have a map of the network. We know from the last paper discussed (Dobrev et al., 2002, 2006c) that the lower bound on searching for a black hole with a map is $\Omega(n \log n)$. The authors show that it is possible to improve on this bound without violating it. They gives a solution that locates the black hole with two agents using just $O(n + D\log D)$ moves, which improves on the lower bound in a large number of networks without violating it. They go farther to show that it is possible to locate the black hole in $\Theta(n)$ moves in a large class of graphs where $D = O\left(\frac{n}{\log n}\right)$.

Dobrev, Flocchini, Královič, and Santoro (2006b) look at black hole search in connected undirected graphs when the agents can only communicate using tokens. The tokens used in the paper are the slightly more powerful version that can be dropped on a node or a port of a node rather than just on the node itself. The authors present a solution that requires $O(\Delta^2 m^2 n^7)$ moves, which appears unlikely to be optimal, by $\Delta + 1$ agents, which is optimal for black hole search where the agents do not know the topology.

Dobrev, Flocchini, and Santoro (2006d) look at black hole search in connected undirected graphs by two agents with a map. The authors develop a preprocessing method based on cycles that allow the agents to find the black hole in all 2-connected networks and uses only $O(n)$ moves for large classes of networks. The classes include Abelian Cayley graphs like hypercubes and multi-dimensional tori of degree three or more and non-Abelian cube graphs like butterfly and wrapped-butterfly networks.

Chalopin, Das, and Santoro (2007) look at a different problem, the rendezvous of agents scattered in a connected undirected graph where there are multiple black links and the nodes and agents are anonymous. Black links are links that act like black holes. Rendezvous is a slightly different problem than black hole search in that the goal is to have the agents all end up on the same node. The authors present an $O(m^2)$-move solution to the rendezvous problem that also solves the black hole search problem using scattered agents in an anonymous network—a model that is similar to our network model.

Flocchini, Ilcinkas, and Santoro (2011) (conference version (Flocchini et al., 2008c)) look at black hole search in connected undirected graphs when the agents can only communicate using pure or node tokens. Unlike the solution presented in (Dobrev et al., 2006b), the agents in this model can only place the token directly on the node. The authors impressively show that pure tokens are just as powerful as whiteboards by giving a solution for two agents each with a single identical token that finds a black hole in a arbitrary network in an optimal $\Theta(n \log n)$ moves.

Glaus (2009) looks at black hole search in connected undirected graphs where the agents have no knowledge of the incoming link. One of the commonly accepted axioms of distributed computing is the axiom of local orientation Santoro (2007), which states that a node knows from which port it received a message. This axiom is extended to the mobile agent/robot model by allowing the agent to know the port from which it arrived; its incoming link. Not all distributed systems provide this capability, so the author looks at the cost of the lack of this information in terms of the number of agents needed to find a black hole. He proves a lower bound and gives a matching-complexity solution that uses $\frac{\Delta^2+\Delta}{\Delta}+1$ agents.

3.3.2 Synchronous arbitrary networks

Black hole search in synchronous networks does not suffer from some of the drawbacks we see in asynchronous network. It is now possible to know that an agent has been eliminated by a black hole because the agent does not return at the appointed time. The network need no longer be connected; we can simply search the accessible portion. Finally, the agents do not need to know anything about the number of black holes in order to terminate.

Cooper, Klasing, and Radzik (2006) look at black hole search by mobile agents in a synchronous undirected graph. Let b be the number of black holes and D_b be the diameter of the portion of the network with the agents' homebase when the black holes are removed from the network. The authors prove a lower bound on black hole search with $b \leq k-2$ agents of $\Omega\left(\frac{n}{k}+D_b\right)$ steps. They give two solutions for $b \leq \frac{k}{2}$. The first solution has a complexity of $O\left(\frac{(n/k)\log n}{\log\log n + bD_b}\right)$ steps. The second has a complexity of $O\left(\frac{n}{k}\right)$ when $k = O(\sqrt{n})$ and $bD_b = O(\sqrt{n})$.

Cooper, Klasing, and Radzik (2008) look at black hole search by mobile agents in a synchronous undirected graph where an agent that finds a fault both repairs the fault and is eliminated by it. They give a solution for $b \leq \frac{k}{2}$ that takes $O\left(\frac{n/k + D\log f}{\log\log f}\right)$ steps, where $f = \min\left\{\frac{n}{k}, \frac{n}{D}\right\}$ and prove a lower bound showing that their solution is optimal.

Kosowski, Navarra, and Pinotti (2009) look at black hole search by robots in a synchronous directed graph. Let m_b be the number of edges in the network leading into black holes. The robots know the size of the network, m_b, and can communicate with each other using whiteboards. The authors show that $O(m_b 2^{m_b})$ robots is sufficient to solve the problem. They also show that when $m_b = 1$, two robots are sufficient, and when $m_b = 2$, four robots are sufficient for synchronous networks but five robots are sometimes required for asynchronous networks.

3.3.3 Specific topologies

As with exploration, many black hole search results deal with specific topologies. With the exception of the recent work in (Chalopin et al., 2011a,b), the papers listed here deal with networks in the asynchronous model.

Dobrev, Flocchini, Prencipe, and Santoro (2007a) look at mobile agents searching for a black hole in an anonymous asynchronous ring. The conference version of this article (Dobrev et al., 2001) was the first work on black holes published in the open literature. It established a number of basic properties of black holes and the information needed by the agents in order to find them that we discuss in detail in Chapter 2. The authors are interested in the cost of using many agents or just a few and whether those agents start co-located or scattered in the ring. For co-located agents, they prove a lower bound of $(n-1)\log(n-1)+O(n)$ moves regardless of the number of agents. They show that at least 2 agents are required to solve the problem and, regardless of the number of agents, $2n-4$ ideal time units are needed. For scattered agents, the authors prove a lower bound of $\Omega(n\log(n-k))$ moves if k is known and of $\Omega(n\log n)$ if k is unknown. If the ring is oriented, with each node have a consistent left-right labelling, they present an algorithm that solves the problem in an optimal $O(n\log n)$ moves with two agents. If the ring is not oriented, three agents are needed.

Dobrev, Flocchini, Královič, Ružička, Prencipe, and Santoro (2006a) look at black hole search in common interconnection networks. We have seen from previous research that it is not possible to solve black hole search in better than $\Omega(n\log n)$ moves in arbitrary asynchronous networks or even ring networks, even when the agents have full topological knowledge. The authors show that it is possible for two agents to search hypercubes, cube-connected cycles, star graphs, wrapped butterflies, chordal rings, and restricted diameter multidimensional tori and meshes in $\Theta(n)$ moves and give solutions for these networks. In fact, the solutions work even if the agents only have an awareness of the topology in which they are working, as opposed to full topological knowledge.

Dobrev, Královič, Santoro, and Shi (2006e) look at black hole search in asynchronous ring networks where the agents communicate with one another using tokens. The authors give an optimal $O(n \log n)$ solution for two agents that requires only a constant number of tokens.

Dobrev, Santoro, and Shi (2008) (conference version (Dobrev et al., 2007b)) look at black hole search by mobile agents scattered in an unoriented asynchronous ring where the agents communicate using a constant number of tokens. The authors show that it is possible to locate the black hole with just 3 agents in $O(n^2)$ moves. They then give two solutions where $k \geq 4$. The first locates the black hole in $O(kn + n \log n)$ moves. The second applies when k is constant and finds the black hole in an optimal $\Theta(n \log n)$ moves.

Shi (2009) looks at black hole search in hypercubes, tori, and complete asynchronous networks when the agents use tokens. The author looks at both co-located and scattered solutions. For co-located agents, she shows that two agents can find the black hole in all three topologies in an optimal $\Theta(n)$ moves using a constant number of tokens. For scattered agents, the author gives a solution for complete networks that uses n agents, each with a token, and takes $O(n^2)$ moves, and a solution for a oriented torus that takes $O(k^2 n^2)$ moves for $k > 3$ agents. The scattered agent solution for the hypercube is reported as future work.

Balamohan, Flocchini, Miri, and Santoro (2010) look at black hole search in asynchronous ring networks. The authors focus specifically on the ideal time required to solve the problem. They start with the time optimal solution from (Dobrev et al., 2007a), which uses $n-1$ agents and has an average and optimal worst case time complexity of $2(n-2)$. The authors improve on this result by presenting an algorithm that uses $n-1$ agents to get an average time complexity of $\frac{7}{4}n-O(1)$ and a worst case time complexity of $2(n-1)$. They then develop an optimal time algorithm that uses $2(n-1)$ agents to get an optimal average time complexity of $\frac{3}{2}n-O(1)$ and an optimal worst case complexity of $2(n-2)$. They finish with an optimal team size algorithm that uses 2 agents to get an average time complexity of $\frac{15}{2}n+O(1)$ and a worst case time complexity of $8n+O(1)$. Like (Dobrev et al., 2007a), all the algorithms presented have a move complexity of $O(n^2)$.

Balamohan, Flocchini, Miri, and Santoro (2011) re-examine the problem of black hole search by mobile agents in an asynchronous ring. We know from (Dobrev et al., 2001, 2007a) that the lower bound on any such search is $\Omega(n\log n)$ moves. The authors of Balamohan et al. (2011) start by determining the exact cost of the lower bound to be $3n\log_3 n-O(n)$ moves. They then present an algorithm that solves the problem in $3n\log_3 n+O(n)$ moves, which improves on the previous upper bound from (Dobrev et al., 2007a). The algorithm works with an optimal 2 agents in an optimal time of $O(n)$, which is an improvement on the $O(n^2)$ move complexity of the optimal time solutions presented in (Balamohan et al., 2010; Dobrev et al., 2007a).

Chalopin, Das, Labourel, and Markou (2011a) look at black hole search by anonymous agents scattered in an anonymous synchronous ring. The agents have constant-sized memory and a constant number of identical tokens. The authors show that for each combination of moveable/unmoveable tokens (tokens can be picked up again or not) and oriented/unoriented ring (the ring has a consistent left/right labelling or not), there is a minimum number of agents and tokens needed to solve black hole search: moveable tokens require 3 agents with 1 token each, regardless of orientation; unmoveable tokens in an oriented ring require 4 agents with 2 tokens each; and unmoveable tokens in an unoriented ring require 5 agents with 2 tokens each. They provide time optimal solutions to the black hole search problem that match these lower bounds.

Chalopin, Das, Labourel, and Markou (2011b) look at the problem of black hole search by finite automata with tokens in an anonymous synchronous torus. They find that it is impossible to find the black hole using a finite team of automata with a finite number of unmovable tokens. However, the authors find that 3 agents with a finite number of tokens is enough if the tokens are moveable, as long as each agent has more than one token. They then give an optimal solution to the problem using 3 agents with 2 tokens each.

3.4 Dynamic networks

There are several classes of networks that are dynamic in that they have topologies that change as a function of time and may even be disconnected at times. These networks include wireless mobile ad hoc networks where the network's topology may change dramatically over time due to the movement of the network's nodes; sensor networks where links only exist when two neighbouring sensors are awake and have power; and vehicular networks, similar to mobile ad hoc networks, where the topology changes constantly as vehicles move. We are interested in dynamic networks where the links appear and disappear in time, as opposed to changes in both links and nodes.

There is a large amount of research on these networks (which are called *delay-tolerant*, *challenged*, *opportunistic*, *evolving*, etc.) focusing mostly on broadcasting and routing in the message passing model. We note some of those works here. Bui Xuan, Ferreira, and Jarry (2003) investigate the properties of journeys, the dynamic graph equivalent of paths in static graphs, looking at how to define them in terms of time, distance, and expected arrival. O'Dell and Wattenhofer (2005) look at the algorithmic properties of flooding and routing in randomly changing dynamic graphs. Burgess, Gallagher, Jensen, and Levine (2006) present a probabilistic routing protocol for routing in randomly changing dynamic networks. Zhang (2006) looks at different probabilistic routing protocols specifically targeted at randomly changing dynamic networks such as mobile ad hoc networks. Zhang, Kurose, Levine, Towsley, and Zhang (2007) look at routing in the UMass DeiselNet, which is a network made up of buses that act as nodes in a wireless network. One deterministic approach was published recently in which Liu and Wu (2009) propose a deterministic routing algorithm for dynamic networks that have cyclical properties.

Recently, more attention has been paid to the algorithmic aspects of dynamic networks. Avin, Koucky, and Lotker (2008) look at the random walks in dynamic networks. The goal of a random walk is the same as the node version of exploration problem: to visit every node in the graph. The expected time required to achieve the goal is called the cover time. While the cover time of an undirected graph using a simple random walk is known to be polynomial in the size of the graph, the authors show that the cover time of a dynamic graph is exponential. In a simple random walk, the agent chooses its next destination uniformly at random from the current node's neighbours. The authors show that a *lazy* random walk is capable of covering a dynamic graph in polynomial time. In a lazy random walk, the agent chooses its next destination uniformly at random from among all nodes in the graph and goes there if there is an edge between the current node and the chosen node.

There are three recent works that attempt to characterize the dynamic environment and how distributed algorithms work within it. Casteigts, Chaumette, and Ferreira (2009) develop tools and methods for analyzing distributed algorithms in dynamic networks. The authors look at three message passing algorithms and show how the analysis ties the success of the distributed algorithm to topological characteristics. Casteigts, Flocchini, Mans, and Santoro (2010) look at the message passing model problem of broadcasting in networks described by time-varying graphs when the connectivity of the network is unpredictable. They provide a wide-range of solutions depending on the goal of the broadcast—shortest, fastest, or foremost, all of which have specific definitions in dynamic networks—and the nature of the network—recurrent or time-bounded recurrent. Casteigts, Flocchini, Quattrociocchi, and Santoro (2011) build on the work of (Casteigts et al., 2009) and (Casteigts et al., 2010) to propose a unified framework for classifying time-varying networks. The authors classify the subway model presented in Chapter 2, as published in (Flocchini et al., 2010b), as a Class 8 (Periodicity of edges) time-varying graph.

At least one study, by Flocchini, Mans, and Santoro (2009b), has looked at how to explore one class of these networks in the mobile agent model: periodically-varying graphs. In the periodically-varying graph (PV graph) exploration problem, agents ride carriers between sites in the network. A link only exists between sites when a carrier is passing between them. The agents explore the network by moving from carrier to carrier when they meet at a site. Unlike our subway model, the exploration is of carriers rather than of sites. The sites only serve as meeting points or time points for the carriers in the PV graph model. This paper inspired the work in Chapters 6, 7, and 8 when we asked the question of what would happen if the agent could step down from the carrier?

3.5 Other mobile agent problems

It is worth noting that there is a large amount of research related to the mobile agent model that deals with problems that are not directly related to those studied in this thesis. We mention three of them here: intruder capture, rendezvous, and leader election.

The intruder capture problem is the mobile agent model formulation of the graph search problem introduced by Parsons (1978a,b). The graph search problem is to determine how many searchers are needed to find an intruder in a particular graph. If the intruder knows the location of all the searchers and can move arbitrarily fast, the problem becomes equivalent to the decontamination of the network, since the "infection" can move anywhere that is not protected. Some early versions of the problem allow for the searchers to jump from node to node in the network and for portions of the network to be recontaminated. The mobile agent version of the problem, called intruder capture or decontamination in the literature, generally focuses on the monotone contiguous graph search problem, where monotone means that a decontaminated node or edge is never recontaminated, and contiguous means that the agents work together to create and enlarge a single contiguous decontaminated area until the entire graph is decontaminated or the intruder is captured. The literature on all variations of this problem includes the following: Barrière et al. (2002); Barrière et al. (2003); Bienstock (1991); Bienstock and Langston (1995); Bienstock and Seymour (1991); Blin et al. (2006); Breisch (1967); Chang (1991); Dendris et al. (1997); Ellis et al. (1994); Flocchini et al. (2007a, 2008a, 2005a, 2008d, 2005b); Flocchini and Santoro (2010); Fomin and Golovach (2000); Fomin and Thilikos (2008); Fomin et al. (2005); Fraigniaud and Nisse (2006a,b); Hansen and Eldredge (1988); Hanusse et al. (2004); Kinnersley (1992); Kirousis and Papadimitriou (1985, 1986); Kozen (1979); LaPaugh (1993); Lengauer (1981); Luccio et al. (2006); Luccio (2007, 2009); Makedon and Sudborough (1983); Megiddo et al. (1988); Neufeld (1996); Parsons (1978a,b); Rabin (1967); Seymour and Thomas (1993); Smith (1987); Stamatiou and Thilikos (1999); von Stengel and Werchner (1997); Suzuki et al. (1998); Suzuki and Yamashita (1992); Takahashi et al. (1995); Thilikos (2000); Yamamoto et al. (1998); Yang et al. (2004).

The rendezvous problem is for scattered agents within finite time to gather on the same node. Like exploration, solutions to the rendezvous problem are used as primitives to build more complex protocols. In the literature, “rendezvous” is often used to describe the two agent case and “gathering” is often used when there are more than two agents, although this distinction is not universal. The literature on this problem includes the following: Barrière et al. (2007); Baston and Gal (2001); Buhrman et al. (1999); Chalopin et al. (2007, 2010); Das (2008); Das et al. (2007, 2005, 2008); Dessmark et al. (2006); Dobrev et al. (2003); Flocchini et al. (2004a,b); Kranakis et al. (2006a, 2008, 2006b, 2003); Yu and Yung (1996).

The leader election problem is to choose a leader unambiguously from the agents. Leader election in the mobile agent model, especially with whiteboards, is often a consequence of solutions to other problems. For instance, all of the solutions presented in this thesis also solve the leader election problem. The problem is trivial for co-located agents in the whiteboard model or with distinct ids. With scattered agents, leader election also requires rendezvous. There is a small amount of literature that deals directly with the leader election problem and includes the following: Barrière et al. (2003, 2007); Das et al. (2006, 2008); Garcia-Molina (1982); Haddar et al. (2008).

CHAPTER 4

Dangerous Graph Exploration by Scattered Agents in the Network Model

In this chapter, we look at the DGE problem in the network model. The DGE problem (Definition 2.7) is to construct a map of a network that contains both black holes (Definition 2.5) and black links (Definition 2.6). A solution correctly solves the problem if at least one agent survives and all surviving agents terminate within finite time with a map. We present a solution that solves the DGE problem in the network model with an optimal number of agents and in an optimal number of moves.

We review the network model in Section 4.1. The model is presented in full detail in Section 2.2. We provide a solution to the DGE problem in the general setting of arbitrary networks of unknown topology with an arbitrary number of black holes and black links and with asynchronous scattered agents. The algorithm is presented in Section 4.2 and its correctness and complexity are presented in Section 4.3. We summarize the chapter in Section 4.4. A preliminary version of these results was presented at IPDPS 2009 (Flocchini et al., 2009a).

Problem	**Result**	**Reference**
DGE by scattered agents	$O(n_S \cdot m)$ $O(nm)$, if $n_S = O(n)$	Chapter 4, Flocchini et al. (2009a)
DGE by scattered agents lower bound	$\Omega(nm)$	Miklik (2010)
DGE by scattered agents in an anonymous network	$O(m^2)$	Chalopin et al. (2007)

Table 4.1: Results related to DGE solution.

We present a protocol that solves the DGE problem, as well as the *spanning tree construction*, *election*, and *rendezvous* problems. The proposed protocol does so at a worst case cost of at most $O(n_S \cdot m)$ or $O(nm)$ moves, where n_S is the number of safe nodes, n is the number of all nodes, and m is the number of edges. The preliminary version of this protocol (Flocchini et al., 2009a) was proven to be optimal by Miklik (2010). Our solution improves the existing $O(m^2)$ bound of the protocol by Chalopin et al. (2007) that used a similar model but with anonymous nodes and agents. These results are summarized in Table 4.1.

4.1 Model

We summarize the network model here. For a full description of the model, see Section 2.2, and for a summary of the notation used, see Table 2.1.

Let $G = (V, E)$ be a simple connected undirected graph with $n = |V|$ vertices or nodes connected by $m = |E|$ edges or links. Let $E(u)$ be the edges incident to node $u \in V$ and $d(u) = |E(u)|$ be the degree of u. Working in G is a set A of $k = |A|$ mobile agents that start scattered in the network at varying times, follow the same algorithm, and move asynchronously. Each node in the network is equipped with a whiteboard that can be accessed in fair mutual exclusion by agents resident on the node. In the network are a set of black holes V_B and a set of black links E_B. Let F_B be the set of frontier links, which are links that connect safe nodes to black holes. Let E_I be the set of inaccessible links, which are links that connect two black holes.

We make the following additional assumptions. The safe portion of the network, $G_S = (V_S, E_S)$, is connected. There are $k \geq f + 1$ agents, where $f = |F_B| + 2|E_B \setminus (E_I \cup F_B)|$. The agents know the number of safe nodes, $n_S = |V_S|$, which is needed for termination.

4.2 Algorithm

We present an algorithm, *ExploreDG*, that solves the DGE problem (Definition 2.7). After an overview of the algorithm, we describe how an agent performs initialization and each of its basic work activities: exploration, verification, and merging. The result of *ExploreDG* is a map of the safe portion of the network, G_S, showing the frontier links, F_B, and the accessible black links, $E_B \setminus E_I$.

4.2.1 Overview

In general, algorithm *ExploreDG* works as follows. The agents build trees out from their homebases in the *exploration* process. The *cautious walk* technique is used during exploration to ensure that only one agent is eliminated per frontier link and at most two agents are eliminated per black link. The *verification* process is used to detect when two trees are touching (a non-tree link is incident to a node in each tree). When two trees are found to be adjacent, they are merged in the *merging* process. An agent terminates the algorithm when the current tree contains n_S

nodes and there is no verification or exploration work left.

4.2.2 Initialization

When an agent a wakes up for the first time on its homebase r, it enters the *initialization* phase. It accesses the node's whiteboard to see if the node has a root marker or a *parent pointer*. If there is no root marker or parent pointer then agent a creates a root marker. A *root marker* is used to coordinate the work of the agents in the tree rooted at that node. It contains a map of the tree with all the information about what work is available in the tree and what work has already been done. It also takes the id of the node on which it was created and records the identity of the tree's verifying agent, initially *null*.

The pseudocode for algorithm *ExploreDG*, including the initialization phase, starts in Algorithm 4.1. In addition to the map, we include counters in the pseudocode representation to make explicit what work is available and what work has been done (the same information is available in the map). We also simplify the algorithm somewhat by implicitly assuming access to the whiteboard in fair mutual exclusion. Access is atomic in that the agent is assumed to have continuous exclusive access to the whiteboard until it explicitly releases it or it moves to another node.

4.2.3 Looking for work

When agent a finishes its initialization phase, it looks for work using the LOOK FOR WORK procedure. It starts by grabbing the tree's root marker. It then looks first for verification work and then for exploration work. If no work is available and the termination conditions have not been satisfied, it waits.

Algorithm 4.1 *ExploreDG*

```
Agent $a$ starts on its homebase $r$.

▷ Initialization
1: if $wb$ is blank then                          ▷ No root marker or parent pointer
       ▷ Create root marker $rm$
2:     $rm.id \leftarrow r.id$                    ▷ Take id of homebase
3:     initialize $rm.M$ with $r$ and its links   ▷ Create map
4:     $rm.c_e \leftarrow d(r)$                   ▷ Links to explore
5:     $rm.c_v \leftarrow 0$                      ▷ Links to verify
6:     $rm.c_f \leftarrow 0$                      ▷ Nodes fully explored
7:     $rm.va \leftarrow null$                    ▷ No verifying agent to start
8: end if
9: LOOK FOR WORK
```

Due to mergers, which are a consequence of verification work, it is possible for the root marker of a tree to move—even a root marker that the agent just created. As a result, each time the agent wants to find work, it must *grab* the root marker using the GRAB ROOT MARKER procedure. A node without a root marker has a parent pointer pointing in the direction of the root marker. The agent follows the parent pointers until it gains access to the root marker. We prove later that this process always works even when the root marker is being merged.

Once agent a has access to the root marker, it looks for work. It starts by checking if it should become the verifying agent for the tree. It becomes the verifying agent if there is verification work available and no existing verifying agent. Next, if agent a is the verifying agent, it checks for verification work. If there is verification work, it chooses a link to verify and follows the VERIFY LINK procedure described below. If there is no verification work, it removes itself as the tree's verifying agent and releases the whiteboard. Otherwise, if a is not the verifying agent for the tree, it checks for exploration work. If there is exploration work, it chooses a link to explore and follows the EXPLORE LINK procedure described below. If there is no exploration work, it waits by releasing the whiteboard, which allows other agents to access it. When the agent completes its work or releases the whiteboard, it grabs the whiteboard again to look for new work. Note that for a waiting agent (and a verifying agent becoming an exploring agent), this has the effect of following the root marker during mergers. The agent terminates the algorithm when there is no work available and the map on the root marker contains n_S nodes.

The pseudocode for the LOOK FOR WORK procedure can be found in Algorithm 4.2. The pseudocode for the GRAB ROOT MARKER procedure can be found in Algorithm 4.3.

We now look at the work of exploration, verification, and merging.

4.2.4 Exploration

Exploration is the work of exploring every accessible link in the network using cautious walk. Agent a has chosen to explore link $[u, v]$ using the EXPLORE LINK procedure (see Figure 4.1). It takes the tree path from the root r to the node u. It then uses the *cautious walk* technique (discussed in Section 2.1.2) to see if $[u, v]$ is safe. It marks the port on u leading to $[u, v]$ as *dangerous* and then traverses to v. If $[u, v]$ is a black link or v is a black hole then agent is eliminated, completing the exploration. The port on u remains marked as dangerous so that no other agent can enter it to be eliminated.

If the agent is not eliminated then it checks to see if v has been previously visited, meaning that it already has a root marker or parent pointer. If v has not been previously visited—it is *new*—then agent a creates a parent pointer pointing to u. It completes the cautious walk by returning to u and marking the port from u to $[u, v]$ as *explored*. It then grabs the root marker to return to r (or wherever the

Algorithm 4.2 Look for work

Agent a is on its homebase r in some tree T. The agent knows the number of safe nodes n_s, which is needed for termination.

```
10: procedure LOOK FOR WORK
11:     $rm \leftarrow$ GRAB ROOT MARKER                          ▷ Find tree's root marker
12:     while $rm.c_f < n_s$ do                   ▷ Safe nodes with unexplored links
            ▷ Verifying agent
13:         if $rm.c_v > 0 \land rm.va = null$ then           ▷ Tree needs verifying agent
14:             $rm.va \leftarrow a$                              ▷ Become verifying agent
15:         end if
            ▷ Verification
16:         if $rm.va = a$ then                                 ▷ If $a$ is verifying agent
17:             if $rm.c_v > 0$ then                            ▷ There are links to verify
18:                 $rm.c_v \leftarrow rm.c_v - 1$
19:                 choose link to verify and mark it on $rm.M$
20:                 VERIFY LINK
21:             else                                         ▷ There are no links to verify
22:                 $rm.va \leftarrow null$                        ▷ Become exploring agent
23:                 release $wb$                                   ▷ Release whiteboard
24:             end if
            ▷ Exploration
25:         else                                              ▷ If $a$ is not verifying agent
26:             if $rm.c_e > 0$ then                           ▷ There are links to explore
27:                 $rm.c_e \leftarrow rm.c_e - 1$
28:                 choose link $[u, v]$ to explore and mark it on $rm.M$
29:                 if $u$ has no more unexplored links then
30:                     $rm.c_f \leftarrow rm.c_f + 1$
31:                 end if
32:                 $a.M \leftarrow rm.M$
33:                 EXPLORE LINK
34:             else                                       ▷ There are no links to explore
35:                 release $wb$                                  ▷ Release whiteboard
36:             end if
37:         end if
38:         $rm \leftarrow$ GRAB ROOT MARKER                      ▷ Find tree's root marker
39:     end while
40: end procedure
```

Algorithm 4.3 Grab root marker

Agent a is on a node in tree T and is trying to access T's root marker.

```
41: function GRAB ROOT MARKER
42:     repeat
43:         if $wb$ has parent pointer then
44:             follow parent pointer
45:         end if
46:     until $wb$ contains $rm$
47:     return $rm$
48: end function
```

root marker is). At r, it adds the information about v to the root marker, including marking $[u, v]$ for verification and marking all of v's incident links except $[v, u]$ for exploration, completing the exploration. If v has been previously visited then agent a completes the cautious walk and grabs the root marker to return to r (or wherever the root marker is). At r, it marks $[u, v]$ for verification, completing the exploration.

The pseudocode for the EXPLORE LINK procedure can be found in Algorithm 4.4.

The use of cautious walk and an agent team size $k = f + 1$, where f is the number of possible faults that could eliminate an agent, gives us the following useful property:

Property 4.1. *At any time, there is always one agent alive.*

4.2.5 Verification

Verification is the work of determining whether every safely explored link is internal or external to the tree (see Figure 4.2). If an external link is found then the agent may try to merge the two trees connected by the link. Agent a has chosen to verify link $[u, v]$ using the VERIFY LINK procedure.

Algorithm 4.4 Exploration

Agent a on root r of tree T has chosen to explore a link $[u,v]$ incident to node $u \in T$. Let $\oplus$ represent the operation of merging maps or root markers.

```
49: procedure EXPLORE LINK
50:     traverse $T$ from $r$ to $u$
51:     walk from $u$ to $v$ via $[u,v]$              ▷ First step of cautious walk
        ▷ If not eliminated by a dangerous element
52:     if $v$ is unexplored then                          ▷ A new node
53:         add $v$ to $a.M$ marking its edges for exploration
54:         set parent pointer in direction of $u$
55:         walk from $v$ to $u$ via $[v,u]$          ▷ Second step of cautious walk
56:         $rm \leftarrow$ GRAB ROOT MARKER    ▷ Find and access tree's root marker
57:         $rm.c_e \leftarrow rm.c_e + (d(v) - 1)$
58:     else                                ▷ $v$ must be have been visited before
59:         walk from $v$ to $u$ via $[v,u]$          ▷ Second step of cautious walk
60:         $rm \leftarrow$ GRAB ROOT MARKER               ▷ Find tree's root marker
61:     end if
62:     mark $[u,v]$ in $a.M$ for verification
63:     $rm.c_v \leftarrow rm.c_v + 1$
64:     $rm.M \leftarrow rm.M \oplus a.M$                           ▷ Merge maps
65: end procedure
```

Agent a starts by checking to see if v is already in the root marker's map. If it is, the link is marked internal, completing the verification. If v is not in the map then agent a takes the tree path from the root r to u, traverses from u to v, and grabs the root marker of v's tree on its root r'. Note that all the links crossed by a have been previously explored and are therefore safe, which gives us the following property:

Property 4.2. *Verification takes place over safe links.*

The agent first checks to see if r and r' have the same id. If they do then a marks $[u,v]$ as *internal*, completing the verification.

If r and r' have different ids then it is possible that they are different trees.

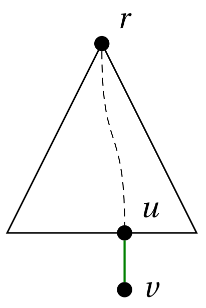

Figure 4.1: Exploration of link $[u, v]$, where $u \in T_r$. Possible movement of r due to mergers not shown.

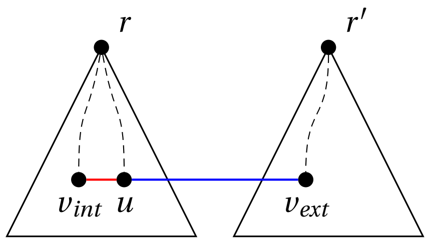

Figure 4.2: Verification of link incident to node u, where link $[u, v_{int}]$ is internal and link $[u, v_{ext}]$ is external. Possible movement of r due to mergers not shown.

There are three cases. The agent finds that the link is internal. The agent finds that r' has a higher id or that r' has a lower id and has no verifying agent. The agent finds that r' has a lower id than r and has a verifying agent.

In the internal link case, the agent finds that r's tree is a subset of r''s tree. This situation results when r has been merged into r' while a is performing the verification. Since r had a verifying agent, a, this situation can only arise when r' has a lower id than r and has a verifying agent. The agent marks $[u, v]$ as internal and becomes an exploring agent in the new tree, completing the verification.

The next case is that the agent finds that r' has a higher id or that r' has a lower id and has no verifying agent. The verification is considered completed and the agent merges the two trees using the MERGE TREES procedure discussed below.

The final case is that the agent finds that r' has a lower id than r and has a verifying agent. The agent marks $[v, u]$ for verification and becomes an exploring agent in the new tree, completing the verification. Note that a is still the verifying agent for r's tree. We prove later that r's tree is always merged.

The pseudocode for the VERIFY LINK procedure can be found in Algorithm 4.5.

Algorithm 4.5 Verification

Agent a on root r with root marker rm has chosen to verify a link $[u, v]$ between node u in its own tree and a node v, which may be in a another tree.

```
66: procedure VERIFY LINK
67:     if $v \in rm.M$ then                 ▷ Both $u$ and $v$ are in the root marker's map
68:         mark $[u, v]$ as internal in $rm.M$          ▷ The agent never leaves $r$
69:         release $wb$                                ▷ Release $r$'s whiteboard
70:     else                                        ▷ Only $u$ is in the map, not $v$
71:         $a.M \leftarrow rm.M$
72:         $a.rmid \leftarrow rm.id$
73:         release $wb$                     ▷ Release whiteboard of $u$'s tree's root
74:         traverse $T$ from $r$ to $u$
75:         walk from $u$ to $v$ via $[u, v]$
76:         $rm \leftarrow$ GRAB ROOT MARKER                ▷ Find $v$'s tree's root marker
77:         if $rm.id = a.rmid$ then                              ▷ $v$ is in $a$'s tree
78:             mark $[u, v]$ as internal in $rm.M$
79:         else
80:             if $rm.id < a.rmid$ then                    ▷ New root has lower id
81:                 if $rm.va \neq null$ then               ▷ Tree has a verifying agent
82:                     if $a.M \subset rm.M$ then      ▷ $u$ and $v$'s trees already merged
83:                         mark $[u, v]$ as internal in $rm.M$
84:                     else                          ▷ $u$ and $v$'s trees are disjoint
85:                         if $[v, u]$ is unexplored then
86:                             mark it for verification in $rm.M$
87:                         end if
88:                     end if
                        ▷ Agent $a$ becomes exploring agent
89:                 else                              ▷ Tree has no verifying agent
90:                     MERGE TREES
91:                 end if
92:             else                   ▷ New root has higher id ($rm.id > a.rmid$)
93:                 MERGE TREES
94:             end if
95:         end if
96:     end if
97: end procedure
```

4.2.6 Merging

Merging is the work of adding one tree to another. Agent a verifying edge $[u, v]$ has arrived on root r' and has chosen to merge r''s root marker with the root marker of its own tree r using the MERGE TREES procedure. The agent picks up r''s root marker and takes the tree path from r' to v. Along the path, it reverses the parent pointers to point towards v. At v, it points the parent pointer towards u. It traverses from v to u over $[v, u]$. Since r may have been merged in a's absence, a grabs the current root marker on root r'' of u's tree. It merges r''s root marker with r'''s, completing the merger. If $r = r''$ then a continues its work as a verifying agent. If $r \neq r''$ then a becomes a exploring agent in the new tree.

It is possible that agents in r''s tree are trying to grab its root marker during the merger. Since we can treat the links as if they are FIFO, these agents simply follow a until it reaches r'' and completes the merger.

Note that all the links used in merging are exactly those used in verification. As a result, Property 4.2 also applies to merging.

The pseudocode for the MERGE TREES procedure can be found in Algorithm 4.6.

4.2.7 Internal link verification problem

The verification process is about finding external links so that the trees they connect can be merged. The internal link verification problem arises from the need to verify all links, internal and external, in order to find the $k' - 1$ external links over which mergers occur, where k' is the number of homebases and $1 \leq k' \leq k$. Ideally, we would like to avoid verifying internal links altogether because we are really only interested in finding the external links. Unfortunately, as we will see, it is not

Algorithm 4.6 Merging

Agent a is verifying link $[u, v]$ and has picked up the root marker at the root of v's tree. Let r_v be the root of the tree T_v and let rm_v be the root marker a is carrying. Let $\oplus$ represent the operation of merging maps or root markers.

```
98:  procedure MERGE TREES
         ▷ At $r_v$ pick up root marker and set parent pointer towards new root
99:      pick up $rm_v$                          ▷ Remove $rm_v$ from $r_v$'s whiteboard
100:     set parent pointer in direction of $v$
101:     release $wb$                            ▷ Release $r_v$'s whiteboard
         ▷ Traverse from $r_v$ to $v$ reversing the parent pointers
102:     move to next node on $T_v$ in direction of $v$ using $rm_v.M$
103:     while not at $v$ do
104:         reset parent pointer in direction of $v$
105:         move to next node on $T_v$ in direction of $v$ using $rm_v.M$
106:     end while
         ▷ At $v$ add it to the tree
107:     reset parent pointer in direction of $u$
108:     walk from $v$ to $u$ via $[v, u]$
         ▷ In $u$'s tree find the root marker and merge it with the one $a$ is carrying
109:     $rm_u \leftarrow$ GRAB ROOT MARKER       ▷ Find $u$'s tree's root marker
110:     $rm_u \leftarrow rm_u \oplus rm_v$       ▷ Merge the two root markers
111:     discard $rm_v$
         ▷ Agent $a$ may become exploring agent if $u$'s tree was merged in $a$'s absence
112: end procedure
```

possible to avoid verifying internal links.

Let $G_i = (V_i, E_i)$ be the subgraph of G explored by the agents from homebase r_i. Let $T_i = (V_i, E_i^T)$ be the restriction of G_i to the tree links. Let $M_i = (V_i^M, E_i^M)$ be the map stored in root marker $rm_i.M$ on root r_i.

Let agent a in tree T_i be verifying a link $[u, v]$ where $u \in T_i$ and $v \in T_j$. The link would not be marked for verification unless $u \in V_i^M$, so we are interested in v. There are a number of good cases. If $i \neq j$, the link is external and either the agent becomes an exploring agent on another tree or it merges the trees. If $i = j$ and $v \in V_i^M$, the agent can confirm that $[u, v]$ is internal without ever leaving r_i just by looking at the map $rm_i.M$. Note that this case always applies to the tree links that are marked for verification.

The internal link verification problem occurs when the link is internal but v is not in the root's map. In other words, $i = j$ and $v \notin V_i^M$. The agent must make the trip all the way to the link and back to its own root with no benefit other than marking the link *internal*. The problem comes from the fact that the agent in T_i that explored v has not yet returned to r_i with that information. Unfortunately, we cannot wait for the agent to return because the network is asynchronous. The result is that the good case of $i \neq j$ and the bad case of $i = j$ and $v \notin V_i^M$ are indistinguishable unless we take some action.

Our solution is to travel to the root on the other side of the link being verified. If we play an adversarial game where the adversary can make some links "slow", we get the worst case scenario where all the internal links fall into the bad case meaning that the agent must travel to the link and return to its own root for every internal link. The result is very costly in terms of complexity but unavoidable given that we know our solution is optimal (Miklik, 2010).

There are other approaches that we could take. For instance, we could colour the trees with the id of the root. Each time a new node is added to the tree, it is marked with the id of the root of the tree. When an exploring agent finds an already visited node, the agent only marks the node for verification if its colour is different from the agent's. Unfortunately, this solution suffers from the same problem as ours. When two trees are merged, all the nodes in one of the trees must be recoloured. Playing the same adversarial game, the adversary can delay the recolouring so that every internal link appears to be external when the agent explores it. Even if the recolouring has finished by the time the verifying agent arrives, it makes no difference because the verifying agent has already made the trip. Our solution has the benefit of waiting until the last possible moment, when the verifying agent is going to leave, to see whether v is now in the map.

4.3 Correctness and complexity

We start by stating the theorem that we prove in this section.

Theorem 4.3. *Algorithm ExploreDG after at most $O(n_S \cdot m)$ moves correctly and within finite time solves the* DGE *problem by constructing a rooted spanning tree of G_S, marking all safe edges as such, and marking all ports in G_S leading to a black hole or to a black edge as dangerous.*

We prove a series of lemmas that show the work of merging, verification, and exploration is all done in finite time.

Lemma 4.4. *Any merger terminates in finite time.*

Proof. A merger starts when a verifying agent picks up a root marker. Let a_1 be the verifying agent from tree T_1 with root r_1 (see Figure 4.3) that is verifying link $[v_1, u_0]$ between T_1 and T_0. Let agent a_1 pick up the root marker of r_0 in order to merge it with the root marker of its own tree. By construction, a_1 can only pick up r_0's root marker only if T_0 does not have a verifying agent or $\mathrm{id}(r_0) > \mathrm{id}(r_1)$, where $\mathrm{id}(r)$ is a function that returns the id of the root marker associated with r. Agent a_1 returns along the path from r_0 to u_0, reversing the parent pointers along the way, then from u_0 to v_1 and along the path to r_1.

If the root marker at r_1 has not been picked up in a_1's absence, the agent arrives at r_1, merges the root markers and the merger is completed proving the lemma.

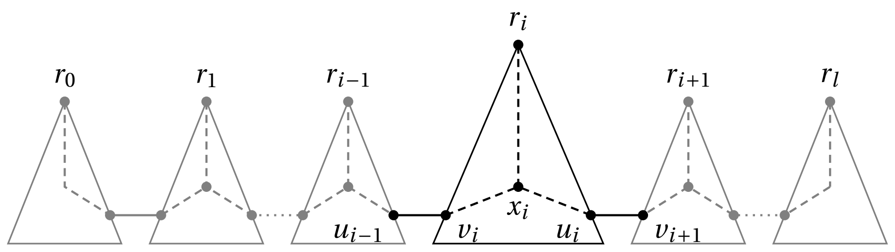

Figure 4.3: Chain of $l+1$ trees used for Lemmas 4.4 and 4.6.

Consider instead the case where the root marker of r_1 has been picked up. By construction, because a_1 is the verifying agent for T_1, the root marker can only be picked up by the verifying agent a_2 of a tree T_2, where $\mathrm{id}(r_1) > \mathrm{id}(r_2)$. Since the agent a_2 is reversing the parent pointers from r_1 to u_1 for its merger, agent a_1 encounters the reversed parent pointers at a *redirection point* x_1 on the path between u_1 and r_1. When a_1 reaches the redirection point, we say that it has completed the first step of the migration. It then continues with the next step.

We say that in step i where $i \geq 1$, a_1 moves from the redirection point x_i in tree T_i towards the root marker of tree T_{i+1}. Step i ends when either a_1 reaches r_{i+1} and the root marker for T_{i+1} has not moved, or a_1 reaches the redirection point x_{i+1}, which only exists if the root marker for T_{i+1} has been moved.

Since the agent's movement is safe (Property 4.2), each step of the migration is completed in finite time. Consider the steps of the migration and the corresponding roots $r_1, r_2, \ldots, r_i, \ldots$. Since there are a finite number of root markers and $\mathrm{id}(r_{i-1}) > \mathrm{id}(r_i)$, where $1 \leq i \leq l$, the number of steps is finite, completing the proof. □

Lemma 4.5. *Any verification terminates in finite time.*

Proof. Verification starts when a verifying agent chooses an unverified link to verify and ends when it reaches the root of the tree on the other side of the link. Let agent

a be the verifying agent from tree T with root r. Let a be verifying link $[u,v]$ where $u \in T$, $v \in T'$, and T' is a tree with root r'. We say that link $[u,v]$ is internal if $T \subseteq T'$ and $r \in T'$ and external otherwise. The verification is completed trivially if $[u,v]$ is internal and v appears in r's root marker's map. Otherwise, agent a takes the tree path from r to u, traverses from u to v, and takes the tree path from v towards r'. We know from Property 4.2 that the verifying agent makes the traversal safely. If the root marker of r' is not being merged, agent a reaches it and the verification is completed in finite time, proving the lemma. If the root marker of r' is being merged, by the algorithm, a follows the root marker, and, by Lemma 4.4, we know that the root marker eventually stops moving. As a result, agent a reaches the root with the root marker and the verification is completed in finite time proving the lemma. □

There are three results from verification. The link can be internal, in which case we know by Lemma 4.5 that it is marked internal within finite time and the verifying agent looks for another edge to verify. The link can be external and lead to a merger, in which case we know by Lemma 4.4 that the merger is completed within finite time. Finally, the link can be external and lead to the verifying agent becoming an exploring agent in the external tree. We now show that the verifying agent's tree, in that case, must be merged within finite time.

Lemma 4.6. *The tree of a verifying agent that becomes an exploring agent in another tree is merged within finite time.*

Proof. By contradiction, assume that agent a_1 from tree T_1 with root r_1 verifying link $[u_1, v_2]$ becomes an exploring agent in some tree T_2 with root r_2 and T_1 is never merged. Agent a_1 can only become an exploring agent in T_2 if $\mathrm{id}(r_1) > \mathrm{id}(r_2)$ and some agent a_2 is T_2's verifying agent. If $\text{id}(r_1) < \text{id}(r_2)$ or T_2 did not have a verifying agent, agent a_1 would merge T_2 with T_1, proving the lemma.

By construction, a verifying agent verifying a link $[u, v]$ ensures that the link $[v, u]$ is marked for verification before it becomes an exploring agent in the new tree. We refer to this link as the *return link*. By Property 4.2, we know verification and mergers are safe. Hence, agent a_2 cannot die before it verifies the return link $[v_2, u_1]$ that agent a_1 ensured was marked for verification. Since T_1 is never merged, the only other possibility is that agent a_2 must itself have become an exploring agent.

Let agent a_2 verify some link $[u_2, v_3]$ between T_2 and some tree T_3 with root r_3, where $\text{id}(r_2) > \text{id}(r_3)$, and let agent a_2 become an exploring agent in T_3. If tree T_3's verifying agent agent a_3 verified the return link $[v_3, u_2]$ added by agent a_2 then it would merge T_2 with T_3. By construction, the merger of two root markers leaves possible external links marked for verification untouched. Only known internal links between the two trees being merged are marked as such. Therefore, the return link $[v_2, u_1]$ must still be marked for verification after the merger of T_2 and T_3.

Since T_1 is never merged, agent a_3 must have become an exploring agent in some other tree before it could verify $[v_2, u_1]$ and, as a consequence, merge T_1. Generalizing this argument, agent a_i, $i \geq 3$, must have become an exploring agent in some other tree T_{i+1}, where $\text{id}(i) > \text{id}(i+1)$, before it could verify $[v_2, u_1]$. As a consequence, since T_1 is never merged, all verifying agents must have become exploring before they could merge their tree with T_1. However, the verifying agent of the lowest id tree with a verifying agent cannot become an exploring agent in some other tree, a contradiction. □

Corollary 4.7. *A verification ends in either the link being marked internal or a merge* *operation where the edge is marked internal.*

Lemma 4.8. *Any exploration terminates in finite time*

Proof. Exploration starts when an exploring agent chooses an unexplored link to explore. Let agent a be an exploring agent from tree T with root r. Let $[u, v]$ be the link to be explored, where $u \in T$. Agent a takes the tree path from r to u and performs the first step of its cautious walk by traversing $[u, v]$ to v. If $[u, v]$ is a black link or v is a black hole, the agent is eliminated and the exploration is completed in finite time proving the lemma. If not, a returns from v by taking the second step of its cautious walk and takes the tree path from u to r. If the root marker of r is not being merged, agent a reaches it and the exploration is completed in finite time proving the lemma. If the root marker of r is being merged, we know from Lemma 4.4 that the merger finishes within finite time, so agent a reaches the root marker and the exploration is completed in finite time proving the lemma. □

An agent is performing work if it is merging a root marker, verifying a link, or exploring a link. There is work to be done in the system if there is at least one root marker that has links to be verified or explored.

Lemma 4.9. *At any time before termination, at least one agent is performing work.*

Proof. By contradiction, assume at time t that there is still work to be done but no agent is performing work. We first show that the available work must be exploration. By Property 4.1, we know that at least one agent must be alive. By construction, a tree only has a verifying agent if there is verification work to be done. By Property 4.2, we know that verification work is safe. By Lemmas 4.4, 4.5, and 4.6, we know that verifications and mergers take a finite time to complete and all trees whose verifying agents become exploring agents are eventually merged, so if there were still verification work to be done, a verifying agent would be doing it. All live agents must therefore be waiting for work. They can only wait for work if all available exploration work is being done. Since we assume that no work is being done, all the agents doing exploration work must have been eliminated by black holes or black links. There are two cases: multiple trees and a single tree. If there is more than one tree left, the elimination of all non-waiting agents means that all the links leading out of those trees lead to black holes or are black links. If the safe portion of the network is connected, at least one link leading out of each of the trees must be safe. Since all the non-waiting agents have been eliminated, the safe portion of the network must be disconnected, a contradiction. If there is a single tree, the elimination of all non-waiting agents implies the tree contains all safe nodes and that the remaining agents should have terminated, a contradiction. □

Lemma 4.10. *Within finite time, all safe nodes are explored and all surviving agents terminate.*

Proof. By Lemma 4.9, we know that at any time t before termination there is work being done. By Lemmas 4.4, 4.5, 4.6, and 4.8, we know that each piece of work finishes within finite time. As a result, within finite time, there is a single node with a map of the network that includes all safe nodes and all safe edges are marked as such. □

Lemma 4.11. *After at most $O(n_S \cdot m)$ moves, all agents that are still alive terminate.*

Proof. We look at the cost for merging, verification, and exploration separately.

There are at most k trees and at most $k-1$ possible merging agents. When a merging agent picks up a root marker, even if it follows the longest path in the chain shown in Figure 4.3, it can move at most $2(n-1)$ moves because all the links it passes over are tree links. Each of the at most $k-1$ merging agents could take up to $O(n)$ moves to complete a merger, giving an overall complexity of $O(kn)$ moves for merging.

There are at most k verifying agents. When a verifying agent crosses an external link, it either becomes an exploring agent or performs a merger, causing another agent to become an exploring agent. As a result, at most k verifying agents can cross external links at a cost of at most $2(n-1)$ moves, giving a complexity for external link verification of $O(kn)$ moves. When a verifying agent traverses an internal link, it takes up to $2(n-1)$ moves to do so. An agent traverses an internal link if only one end of the link appears in the root marker's map. Tree links do not require traversal to be verified. For all other links, up to $O(m)$ of them, an adversary can ensure that the information for one side of the link is always delayed, giving a complexity for internal link verification of $O(nm)$ moves. Because of internal links, the overall complexity of verification is $O(nm)$ moves.

Every accessible link, up to m or $O(m)$, in the graph is explored. It requires at most $n-1$ moves to reach a link, 2 moves for cautious walk, and at most $n-1$ moves to return to the root. An agent that is eliminated travels at most $n-1$ or $O(n)$ moves. An agent that is not eliminated travels at most $2n$ or $O(n)$ moves. Therefore, the overall complexity of exploration is $O(nm)$.

Adding all these costs together, we get a complexity of $O(kn)$ for merging plus $O(nm)$ for verification plus $O(nm)$ for exploration or $O(nm)$ moves overall. Note, however, that only nodes in the safe portion of the network, G_S are connected by tree links or n_S of them, so we can reduce the complexity to $O(n_S \cdot m)$ moves overall proving the lemma. □

Miklik (2010) proves that the complexity of our DGE solution, as presented in (Flocchini et al., 2009a), is optimal (Theorem 2.5.1, Corollary 2.5.2).

4.4 Conclusion

In this chapter, we presented a solution to the DGE problem in the network model that is both agent and move optimal. The DGE problem (Definition 2.7) is to construct a map of a network that contains both black holes (Definition 2.5) and black links (Definition 2.6). We presented a solution to the problem using a team of agents scattered in the network. We proved that the solution is correct and works with an optimal number of agents $k = f + 1$, where $f = |F_B| + 2|E_B \setminus (E_I \cup F_B)|$, F_B is the set of frontier links connecting safe nodes to black holes, and $E_B \setminus (E_I \cup F_B)$ is the set of accessible black links that are not also frontier links. We then proved that our solution has a complexity of $O(n_S \cdot m)$ or $O(nm)$ moves, where n_S is the number of safe nodes, n is the number of nodes, and m is the number of links in the network. Our solution is proven optimal by Miklik (2010).

CHAPTER 5

Dangerous Graph Exploration by Scattered Agents with Link Deletions in the Network Model

In this chapter, we look at the DGE-LD problem in the network model. The DGE-LD problem (Definition 2.8) is to visit every accessible link in the network, marking locally the frontier links leading to black holes (Definition 2.5) and the accessible black links (Definition 2.6), while an adversary is allowed to delete links under certain conditions. A solution correctly solves the problem if at least one agent survives and all surviving agents terminate within finite time. We present a solution that solves the DGE-LD problem in the network model with an optimal number of agents.

We review the network model in Section 5.1. The model is presented in full detail in Section 2.2. We provide a solution to the DGE-LD problem in the general setting of arbitrary networks of unknown topology with an arbitrary number of black holes and black links and with asynchronous scattered agents. During the execution, we play an adversarial game in which the adversary is allowed to delete

Problem	**Result**	**Reference**
DGE by scattered agents	$O(nm)$	Chapter 4, Flocchini et al. (2009a)
DGE-LD by scattered agents	$O(k^2 \cdot n_S + n_S \cdot m + k \cdot n_S \cdot D)$ $O(nm^2)$, if $D, f = O(m)$ $O(nm)$, if $D, f = O(1)$	Chapter 5
Note that $k \geq f+1$.		

Table 5.1: Results related to DGE-LD solution.

any link so long as no agent is traversing it and its removal does not disconnect the safe portion of the graph, G_S. The algorithm is presented in Section 5.2 and its correctness and complexity are presented in Section 5.3. We summarize the chapter in Section 5.4.

We present a protocol that solves the DGE-LD problem. The proposed protocol does so at a worst case cost of at most $O(k^2 \cdot n_S + n_S \cdot m + k \cdot n_S \cdot D)$ agent moves, where k is the number of agents, n_S is the number of safe nodes, m is the number of edges, and D is the number of link deletions by the adversary. The number of deletions can range from $0 \leq D \leq m - n_S - 1$ and the number of faults f and agents $k = f+1$ can have the same range. As a result, our DGE-LD solution has the same complexity as our DGE solution, $O(nm)$, when there are a small constant number of deletions and faults and a worst case complexity of $O(nm^2)$ when there are $O(m)$ deletions and/or faults. These results are summarized in Table 5.1.

5.1 Model

We summarize the network model here. For a full description of the model, see Section 2.2, and for a summary of the notation used, see Table 2.1.

Let $G = (V, E)$ be a simple connected undirected graph with $n = |V|$ vertices or nodes connected by $m = |E|$ edges or links. Let $E(u)$ be the edges incident to node $u \in V$ and $d(u) = |E(u)|$ be the degree of u. Working in G is a set A of $k = |A|$ agents that start scattered in the network at varying times, follow the same algorithm, and move asynchronously. Each node in the network is equipped with a whiteboard that can be accessed in fair mutual exclusion by agents resident on the node. In the network are a set of black holes V_B and a set of black links E_B. Let F_B be the set of frontier links, which are links that connect safe nodes to black holes. Let E_I be the set of inaccessible links, which are links that connect two black holes.

We make the following additional assumptions. The safe portion of the network, $G_S = (V_S, E_S)$, is connected. The adversary can delete without leaving a discernible trace any link in the network as long as no agent is traversing it and G_S remains connected. There are $k \geq f + 1$ agents, where $f = |F_B| + 2|E_B \setminus (E_I \cup F_B)|$. The agents know the number of safe nodes, $n_S = |V_S|$, which is needed for termination.

5.2 Algorithm

We present an algorithm, *ExploreDG-LD*, that solves the DGE-LD problem (Definition 2.8). We start by describing the basic work activities of exploring, verifying, and merging. We then describe how an agent deals with deletions during its work. The result of *ExploreDG-LD* is the visiting of all safe nodes and accessible links in the network by some agent, the survival of at least one agent, and the marking locally of all frontier links, F_B, and accessible black links, $E_B \setminus E_I$.

5.2.1 Overview

In general, algorithm *ExploreDG-LD* works as follows. The agents build trees out from their homebases in the *exploration* process. The *cautious walk* technique is used during exploration to ensure that only one agent is eliminated per frontier link and at most two agents are eliminated per black link. The *verification* process is used to detect when two trees are touching (a non-tree link is incident to a node in each tree). When two trees are found to be adjacent, they are merged in the *merging* process. An agent terminates the algorithm when the current tree contains n_S nodes and there is no verification or exploration work left.

Link deletions obviously complicate the process of building the trees and connecting them together. Some deletions—such as the deletion of an unexplored link or an inaccessible link between two black holes—have no effect on the execution of the algorithm. On the other hand, the deletion of a tree link can have a significant effect. The trees that the agents build out from their homebases provide safe paths through the safe portion of the network, G_S. By eliminating a tree link, the adversary can cut an agent or agents off from access to the root marker of the tree in which they are working. Because the deletions are undetectable, we use maps at the nodes and carried by the agents to detect such deletions.

5.2.2 Operations without deletions

In the absence of deletions, algorithm *ExploreDG-LD* implements almost exactly algorithm *ExploreDG* presented in Chapter 4. We discuss the differences between the two algorithms in the absence of deletions at the end of this section. We describe the algorithm here from the point of view of an agent and with the help of

Figures 5.1 to 5.3, which show the movement of the agent from the time it starts on a single work task until it finishes that task. The square numbers in each of the diagrams refer to the cases where no deletion is encountered. The circled letters in subsequent diagrams refer to cases where a deletion is encountered. In this section, we focus on the non-deletion cases.

When an agent *a* first wakes up, it enters the *initialization* phase. It accesses the whiteboard of its homebase to see if the node has a root marker. If there is no root marker then agent *a* creates one. The *root marker* contains a map of the tree rooted at the node, as well as all the information needed to find verification and exploration work in the tree. In fact, every visited node will have a root marker for the subtree of which it is the root, even if that subtree is only the node itself. We say that a root marker is *active* if its node has no parent; otherwise, we say that a root marker is *passive*. Only active root markers are used to coordinate work and, as we will see, a passive root marker only becomes active because of the deletion of the link to its parent in the tree.

After initialization, the agent enters the *main loop* of the algorithm and continues until the termination conditions are met. Each round, the agent looks for work in the active root marker. If the current node does not have an active root marker then the agent “grabs” the active root marker by following the parent pointers at each node until the agent finds it. The agent first looks to see if there is any verification work in the root marker. If there is no verification work then it looks for exploration work. Finally, if there no exploration work then it waits until work arrives, following the active root marker if it moves because of a merger. We describe the work of the agent starting first with exploration, then verification, and then merging.

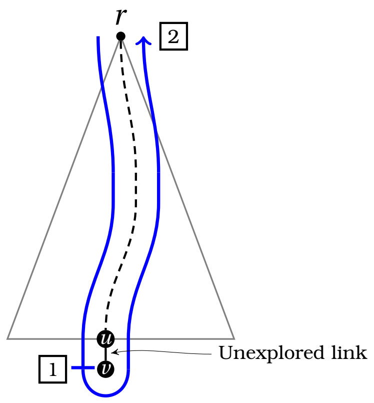

Figure 5.1: Exploration of unexplored link $[u, v]$ with no deletions. Cases include encountering a black hole or black link ($\boxed{1}$), and successful return to its own root ($\boxed{2}$) (possible movement of r due to mergers is not shown).

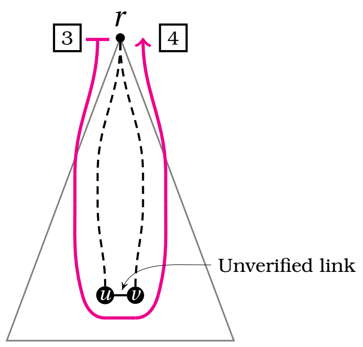

Figure 5.2: Verification of internal link $[u, v]$ with no deletions. Cases include the successful verification of the link ($\boxed{3}$, $\boxed{4}$).

Exploration is the work of exploring every accessible link in the network using cautious walk. Agent a in tree T with root r, as shown in Figure 5.1, chooses a link $[u, v]$ for exploration and takes the tree path from r to u. The agent updates all the passive root markers along the path noting that $[u, v]$ is being explored. It then uses the *cautious walk* technique to test if $[u, v]$ is safe. It marks as *dangerous* the port on u leading to $[u, v]$ and then traverses $[u, v]$ to v. If $[u, v]$ is a black link or v is a black hole then the agent is eliminated as shown in case $\boxed{1}$ in Figure 5.1. The port on u remains marked as dangerous and no other agent can enter it to be eliminated.

If the agent is not eliminated then it checks to see if v has been previously visited. If v has not been previously visited then a marks it as visited on its whiteboard, creates a parent pointer pointing towards u, and creates a root marker for the subtree rooted at v. It completes the cautious walk by returning to u and marking the port to $[u, v]$ as *explored*. It returns from u to r (or wherever the active root marker is) by grabbing the root marker, adding v to all the passive root markers along the way, marking $[u, v]$ for verification, and marking v's other links for exploration. If v has been previously visited then a completes the cautious walk and returns to r marking $[u, v]$ for verification in all the root markers along the way.

A safe return to r is shown as case $\boxed{2}$ in Figure 5.1. Note that because of merging, which we describe below, the active root marker may have moved. In this case, the agent continues following the parent pointers up the tree, adding the information about v to each root marker it passes, until it reaches the active root marker. Its exploration is then done.

Verification is the work of determining whether every safely explored link is internal or external to the tree in which the agent is working. If an external link is found then the agent may try to merge the two trees connected by the link.

We look first at the verification of internal links as shown in Figure 5.2. Agent a working in tree T with root r chooses link $[u, v]$ for verification. By construction, $[u, v]$ can only be marked for verification if $u \in T$ and $[u, v]$ has already been explored. The agent needs to determine if $v \in T$ or $v \notin T$. If v is in the map of T's active root marker at r then the agent knows that $[u, v]$ is internal and it does not even have to leave r to complete the verification, which is shown as case $\boxed{3}$ in Figure 5.2. Case $\boxed{3}$ always occurs for tree links that have been marked for verification.

If v is not in the map, it could be because $v \notin T$ or the agent from T that explored v has not yet returned to the active root marker (see Section 4.2.7 for a discussion of the internal link verification problem). For an internal link, it must be the latter case. Agent a traverses to the root of v's tree T' with root r' to determine if $T \subseteq T'$ (since r may have been merged with r' during the agent's traversal). The agent traverses from r to u, across $[u, v]$, and then from v to r'. Since we can assume that the links are FIFO and we are looking at internal link verification, we know that the agent that explored v must reach the active root marker before a reaches it and a marks the link as internal on its return, which is shown as case $\boxed{4}$ in Figure 5.2. Agent a's verification of $[u, v]$ is finished.

Unlike during exploration, agent a does not mark $[u, v]$ as being verified on all the passive root markers on its traversal from r to u and v to r', where $T \subseteq T'$. Instead, the agent checks to see if v is in the map of the subtree rooted at each passive root marker it passes on the path from v to r'. If it is then a marks the link as internal to that subtree. Otherwise, a does nothing. Note that all the root markers from r to u already had $[u, v]$ marked for verification by the agent that explored $[u, v]$ and the same is true for the link $[v, u]$ on the path from r' to v. As a result, if a deletion were to create a new tree below the point where both u and v are in the same subtree, the link would already be marked for verification again in the new active root marker.

We now look at the verification of external links as show in Figure 5.3. Agent a working in tree T with root r chooses link $[u, v]$ for verification. In the external case, node v belongs to some tree T' with root r', where $T \nsubseteq T'$. The agent traverses from r to u, across $[u, v]$, and from v to r' using parent pointers to find the active root marker in T', which is shown as case $\boxed{5}$ in Figure 5.3. Along the way it marks $[v, u]$ for verification in every root marker, if it is not already there. The agent must now decide whether to merge T' with T. If $\mathrm{id}(r') > \mathrm{id}(r)$, where id() is a function that returns the id of a node's root marker, it picks up the active root marker to

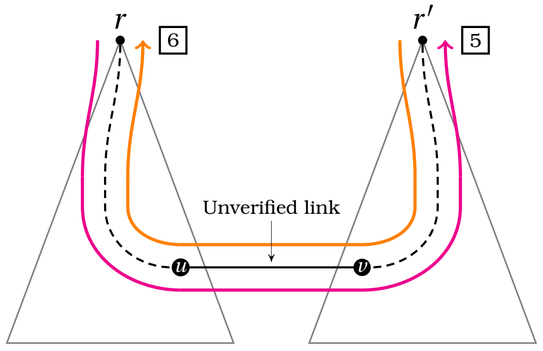

Figure 5.3: Verification of external link $[u, v]$ and merging across link $[v, u]$ with no deletions. Cases include successfully verifying the link is external ($\boxed{5}$), and successfully merging the two trees ($\boxed{6}$) (possible movement of r or r' due to mergers is not shown).

perform a merger. If $\mathrm{id}(r') < \mathrm{id}(r)$, it adds $[v, u]$ to the links in need of verification, if it is not already there, and begins working in tree T'. In either case, agent a's verification of $[u, v]$ is finished.

Merging is the work of adding one tree to another. Agent a has picked up the active root marker in tree T' with root r' and it traverses from r' to v. Along the way, it reverses the parent pointer to point towards the tree's new root and adjusts the maps of the passive root markers along the way to reflect the branch of the subtree lost by the reversal. It then adds $[v, u]$ as a tree link and traverses from u to the active root marker, on r or elsewhere due to mergers, adding T' and its work information to the root markers along the way, including the active root marker. The merger is then finished as shown in case $\boxed{6}$ in Figure 5.3.

Difference with algorithm *ExploreDG*

There are two significant differences between algorithm *ExploreDG-LD* that we present in this chapter and algorithm *ExploreDG* that we presented in that last chapter.

The first difference is the number and location of the root markers. In algorithm *ExploreDG*, only the roots of trees have root markers. In algorithm *ExploreDG-LD*, all nodes have root markers that contain information about the subtree of which they are the root. When a deletion happens, the passive root markers allow the agents to continue their work without having to generate the information for a new root marker, a process that itself could be interrupted by a deletion.

The second difference is the lack of a single verifying agent for each tree. In algorithm *ExploreDG*, the single verifying agent was used to simplify the complexity of the algorithm. Unfortunately, it turns out to be impossible to adapt the single verifying agent approach to the DGE-LD problem. Let agent a be the verifying agent for tree T and let it verify edge $[u,v]$ where $u \in T$ and $v \in T'$, where $T \neq T'$. Let T be the lowest id tree in the network. If an adversary deletes $[u,v]$ after a traverses it during its verification, the agent cannot cross it when it returns to merge T''s root marker with T's. Agent a is forced to start working in tree T' with its new root v. Let there be only exploration work available and let agent a encounter a black hole or black link as the first link it tries to explore. Using the single verifying agent per tree approach of algorithm *ExploreDG*, it is now impossible for T to be merged, since it has the lowest id and a verifying agent and that verifying agent has been eliminated. We can actually use this scenario to chain $k-1$ trees together so that none of them can be merged. Consequently, we must allow all agents to be verifying agents despite the cost in complexity.

Since it is impossible to adapt the single verifying agent approach, we must prove that having multiple verifying agents is an effective solution. To do that, we re-prove Lemma 4.6 with the extra verifying agents.

Lemma 5.1 (*ExploreDG-LD* version of Lemma 4.6)**.** *In the absence of deletions, the* *tree of a verifying agent that becomes an exploring agent is merged within finite time.*

Proof. When a verifying agent verifies an external link and finds the tree on the other side has a lower id than its own tree, it starts working for that tree. Let a_2 be the verifying agent from tree T_2 with root r_2 that is verifying edge $[v_2, u_1]$ between T_2 and T_1 and has arrived on root r_1. Let $\mathrm{id}(r_1) < \mathrm{id}(r_2)$. Since T_1 has a lower id root marker, a_2 marks the edge it was verifying, but in the opposite direction, $[u_1, v_2]$, for verification in T_1's root marker, and then starts working there.

Without loss of generality, assume that all other agents are currently working on exploration and there are no other links to be verified. Agent a_2 immediately chooses to verify $[u_1, v_2]$ and returns to r_2. Since T_2 has a higher id root marker, agent a_2 picks it up and returns to r_1 to merge T_2's root marker with T_1's. Hence the lemma follows. $\square$

It is possible that other agents are already trying to merge the two trees, but since mergers are only performed by agents from lower id trees, we still have a finite number of mergers that can take place and no cycle of mergers can emerge.

Consequently, we can make the following conclusion.

Lemma 5.2. *In the absence of deletions, algorithm ExploreDG-LD correctly implements algorithm ExploreDG.*

Proof. By construction, algorithm *ExploreDG-LD* implements algorithm *ExploreDG* with two differences: algorithm *ExploreDG-LD* requires that there be root markers at every node and requires that any agent can become a verifying agent. The extra root markers only affect the computations the agents need to do at each node and increase the overall storage needed by the algorithm across all nodes. The extra verifying agents increase the complexity but, by Lemma 5.1, do not change the correctness because verification is inherently safe by Property 4.2. □

5.2.3 Operations with deletions

There are certain types of deletion that an agent never encounters or which have very little effect on the agent. For instance, the deletion of an inaccessible link could never be detected nor could the deletion of a link that no agent had ever known about because the link or a node incident to it had never been explored. The deletion of a known non-tree link is worth noting in the agent's map if it passes by it, but it does not have an effect on the actions taken by the agent. As a result, we are only concerned with the deletion of a tree link or of a link being explored or verified. When we discover such a deletion, we work around it while taking steps to repair the damage. The actions taken often depend more on what was deleted than on the work being performed by the agents, so many of the deletion handling cases overlap. We look at the actions taken by the agent in each case and note any task specific differences.

As a consequence of how we have defined work—the exploration or verification of a link or the merging of one root marker with another—it is only possible for an agent to encounter two deletions during a single piece of work. It can encounter a single deletion either on its way away or back towards the root of its own tree. It can only encounter two deletions if it encounters one on the way away from the root of its own tree and another on the way back. If an agent is unable to return to its own root, it simply starts working wherever it is. The cases presented below take into account both possibilities, one deletion or two, where necessary.

The cases cover the following work. Let r, r', and r'' be the roots of trees T,

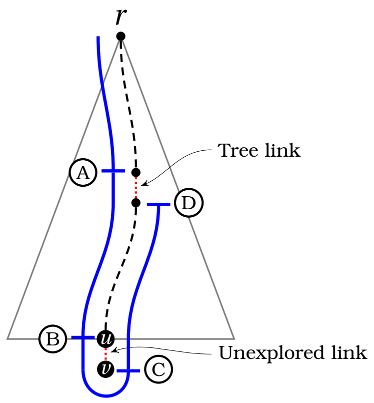

Figure 5.4: Exploration of unexplored link $[u, v]$ with one deletion. Cases include the deletion of a tree link (Ⓐ, Ⓓ) and the deletion of the unexplored link, (Ⓑ, Ⓒ).

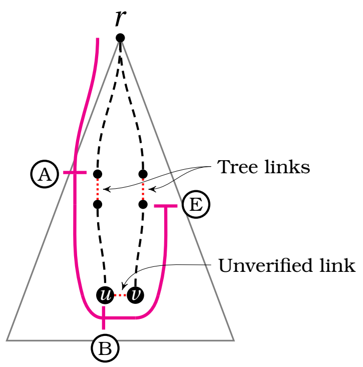

Figure 5.5: Verification of internal link $[u, v]$ with one deletion. Cases include the deletion of a tree link (Ⓐ, Ⓓ, Ⓔ), and the deletion of the unverified link (Ⓑ).

T', and T'', respectively. For exploration, an agent a is exploring link $[u, v]$, where $u \in T$. For internal verification, an agent a is verifying link $[u, v]$, where $u \in T$, $v \in T'$, and $T \subseteq T'$. For external verification, an agent a is verifying link $[u, v]$, where $u \in T$, $v \in T'$, and $T \neq T'$. For merging, an agent a is merging across link $[v, u]$, where $v \in T'$ or $v \in T''$ depending on the number of deletions, $u \in T$, $T'' \subset T'$, and $T \neq T'$. Let $[x, y]$ be a tree link on the path from the root to the link being worked on, where x is closest to the root and y is closest to the link.

Case Ⓐ in Figures 5.4 to 5.6 applies to an exploring or verifying agent that finds a tree link $[x, y]$ deleted on the path from r to u. The agent returns from x to the active root marker deleting the subtree rooted in x from the map of every root marker along the way. The agent then looks for new work.

Case Ⓑ in Figures 5.4 to 5.6 applies to an exploring or verifying agent that finds that the link $[u, v]$ it is meant to explore or verify has been deleted. The agent returns from u to the active root marker marking the deletion on the map of every

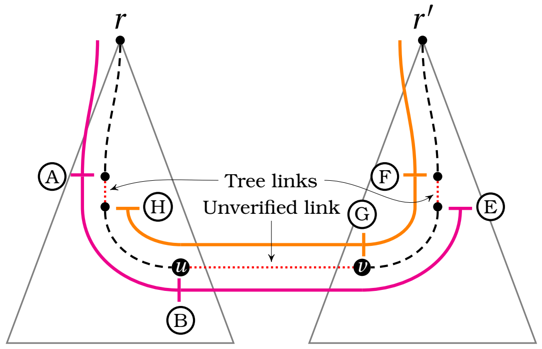

Figure 5.6: Verification of external link $[u, v]$ and merging across link $[v, u]$ with one deletion. Cases for verifying (Ⓐ, Ⓑ, Ⓔ) and merging (Ⓕ, Ⓖ, Ⓗ) include the deletion of a tree link (Ⓐ, Ⓔ, Ⓕ, Ⓗ) and deletion of the unverified link, (Ⓑ, Ⓖ).

root marker along the way. The agent then looks for new work.

Case Ⓒ in Figure 5.4 applies to an exploring agent that finds that $[u, v]$ has been deleted after it has traversed it during the first step of its cautious walk. If v is a new node, the agent creates a root marker and begins working in the new tree rooted at v. If v has already been visited, the agent traverses from v to the active root marker in v's tree. The agent then looks for new work.

Case Ⓓ in Figures 5.4 and 5.7 applies to an exploring agent that finds a tree link $[y, x]$ deleted on the path to r. The agent starts working for the new active root marker at y.

Case Ⓔ in Figures 5.5 and 5.6 applies to a verifying agent that finds a tree link $[y, x]$ deleted on the path from v to the root of v's tree. If $\text{id}(y) > \text{id}(r)$ then the agent picks up y's root marker and merges it. If $\text{id}(y) < \text{id}(r)$ then the agent adds $[v, u]$ to the links to be verified, if it is not already there, and starts working for the new active root marker at y.

Case Ⓕ in Figures 5.6 and 5.7 applies to a merging agent that finds a tree link $[x, y]$ deleted on the path to v. The agent deletes the subtree rooted in y from the

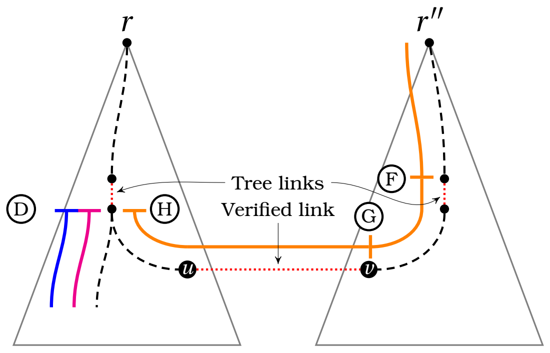

Figure 5.7: Agent encounters second deletion during exploration, verification, or merging. Cases include, for returning exploring and verifying agents, the deletion of a tree link (Ⓓ), and, for merging agents, the deletion of tree links (Ⓕ, Ⓗ) and the deletion of the verified link (Ⓖ).

now active root marker on x and starts working there.

Case Ⓖ in Figures 5.6 and 5.7 applies to a merging agent that finds the link $[v, u]$ deleted. The agent starts working for the active root marker at v.

Case Ⓗ in Figures 5.6 and 5.7 applies to a merging agent that finds a tree link $[y, x]$ deleted on the path from u to r. The agent starts working for the new active root marker at y.

5.3 Correctness and complexity

We start by stating the theorem proved in this section.

Theorem 5.3. *Algorithm ExploreDG-LD after at most $O(k^2 \cdot n_S + n_S \cdot m + k \cdot n_S \cdot D)$ moves correctly and within finite time solves the* DGE-LD *problem by constructing a rooted spanning tree of G_S, marking all safe edges as such, and marking all ports in G_S leading to a black hole or to a black edge as dangerous.*

To prove the theorem, we prove the correctness of algorithm *ExploreDG-LD* by showing that, while deletions can slow the algorithm, they cannot stop the algorithm from making progress.

We start by noting that because of our use of cautious walk, a team size of $k = f + 1$, where f is the number of faults that could eliminate an agent, and the assumption that deletions do not eliminate agents, Property 4.1 also applies to algorithm *ExploreDG-LD*: at any time, there is always one agent alive.

We next look at work that is aborted as the result of deletion. We say that a work task—exploration, verification, or merging—is *aborted* if the agent is unable to reach the link being explored, verified, or merged over, respectively.

Lemma 5.4. *Within finite time, an agent completes aborted work.*

Proof. The proof is by construction. In the cases of exploration and verification, the agent reports its inability to reach the link it has chosen to work on to the active root marker of its current tree and its work is complete. In the case of merger, the agent drops the root marker on the current node and its work is complete. □

Note that in the case of exploration and verification, the active root marker may have moved farther away due to a merger or closer due to a deletion.

We now show that there is always at least one edge leading out of any tree or subtree explored by the agents that cannot be deleted by the adversary.

Let G_E be the explored portion of the graph. Let $G_{SD} = (V_{SD}, E_{SD})$ be the safe portion of the graph G_S after the adversary has performed all its deletions. Let $T = (V_T, E_T)$ be any tree or portion thereof, where $V_T \neq \emptyset$, built by the agents during the algorithm. We define a tree *border* edge as any safe edge connecting two safe nodes where one end is in the tree and one is not in the tree. Let the set of tree border edges be $E_{TF} = \{e \in E_S : e = [u, v] \wedge u \in V_T \wedge v \notin V_T\}$. For any such tree or subtree thereof that covers less than all the nodes, we show that there is a tree border edge that cannot be deleted.

Lemma 5.5. *For any tree or subtree $T \subseteq G_E$, where $V_T \subset V_S$, there is a safe edge $e \in E_{TF}$ such that $e \in E_{SD}$.*

Proof. By contradiction, assume that no such edge $e \in E_{TF} \cap E_{SD}$ exists. Since we have assumed that no edge is in E_{TF} and E_{SD}, the adversary deletes the all edges in E_{TF}. Since $V_T \subset V_S$, T cannot span the entire safe portion of the graph. As a result, the deletions of all the edges in E_{TF} disconnects T from the rest of the graph, contradicting our assumption that deletions do not disconnect the safe portion of the graph. □

We now focus on work in the trees created by the agents. Each tree has an active root marker. We show that, with one exception, all tree border edges are marked for work in the tree's active root marker, being worked on, or become internal within finite time. The one exception comes as the result of the deletion of a tree link. For the links that become internal, we assume that there is an agent available to do the work that leads to this outcome. We show later that such an agent eventually becomes available.

When the adversary deletes a tree link, it creates a subtree in the tree where the deletion takes place, even if that subtree is only a single node. We call a subtree created in this way a *deletion subtree* and the tree to which the deleted edge currently belongs the *original tree*.

Let T_O be the original tree and T_S be the deletion subtree created by the deletion of tree edge e_D. Let E_{OS} be the tree border edges, if any, that connect T_O to T_S and E_{SO} be the same edges in the opposite direction. Let $E_{OS} = E_{SO} = \emptyset$, if the tree is not an original tree or a deletion subtree, respectively.

Lemma 5.6. *For any tree $T \subseteq G_E$ with an active root marker, where $V_T \subset V_S$, all edges $e \in E_{TF} \setminus E_{OS}$ are marked for work in T's active root marker, being worked on, or are marked as internal within finite time.*

Proof. By contradiction, assume that there is an edge $e \in E_{TF} \setminus E_{OS}$ that is not marked for work, being worked on, or marked as internal within finite time. By definition, we know that $e = [u, v]$, where $u \in T$, $v \notin T$, and e, u, and v are safe.

Since u is part of T, all its links should have been marked for exploration. Since e is safe and is not marked for work or being worked on, the exploration must have completed successfully. If v was a new node then the exploring agent added v to T, contradicting the assumption that $v \notin T$. So, v must be a node that had been previously visited and added to some other tree.

By construction, every explored safe link is immediately marked for verification. Since e is not marked for work or being worked on, the verification must have completed successfully. Since $v \notin T$ and belongs to some other tree, we know it is an external link. Since the verification was successfully completed, by Corollary 4.7, we know that, in the absence of deletions, every external link that is verified eventually becomes internal because of a merger. As a result, e is marked internal within finite time, contradicting our assumption that it is not. The merger must have been prevented by a deletion. By construction, e is marked for verification in the opposite direction, $[v, u]$, in every passive root marker from v to the root of v's tree (similarly, $[u, v]$ is marked from u to the root of u's tree). To prevent a merger, the deletion must have been of a tree link along the path from v to the root of v's original tree, which we call T_0. The deletion creates a deletion subtree T_1 that contains v, where $[v, u]$ is marked for verification in T_1's newly active root marker.

Assume that an agent in T_1 decides to verify $[v, u]$. In the absence of deletions, e becomes internal within finite time, contradicting our assumption that it does not. Therefore, the merger must have been prevented by a deletion. In general, assume that the ith deletion of a tree link on the path from v to T_{i-1}, where $i > 2$, creates a deletion subtree T_i that contains v. Assume that an agent in T_i decides to verify $[v, u]$. In the absence of deletions, e becomes internal, contradicting our assumption. However, there are a finite number of tree links on the path from v to the root of T_0 that can be deleted to stop $[v, u]$ from being verified; therefore, e becomes internal within finite time, contradicting our assumption that it does not. □

What about the links in E_{OS}? The reason that they are not marked for work is because they are marked internal to T_O even though they are now external due to the deletion of e_D. By contrast, the links of E_{SO}, which are the same links as E_{OS} but in the opposite direction, are automatically marked for verification in T_S's active root marker. By construction, when an agent verifies an internal link, it only marks the link as internal in the passive root markers on its path if both ends are in the subtree rooted in the passive root marker; otherwise, the link remains marked for verification. The set E_{SO} are exactly those links in T_S that are internal to T_O before the deletion but were marked for verification in T_S's passive root marker when it became active.

We now prove that the links in E_{OS} are guaranteed to be marked for work in T_O's active root marker if there is previously reported work in T_S, there is an agent working in T_O, and there are no other connections to T_S; otherwise, there is no guarantee they are marked for work.

Lemma 5.7. *Let $e_W \in T_S \setminus E_{SO}$ be marked for work in T_S and let T_S have no agents working for it. Let an agent a in T_O choose to work on e_W. Within finite time, the links in E_{OS} are marked for work in the active root marker of T_O.*

Proof. By construction, the links in E_{OS} are marked for work when agent a decides to work on link $e_W \in T_S$. Since there are no agents in T_S, the agent reporting the work at e_W must have crossed over e_D before the deletion and, therefore, the work must have been reported all the way up to the active root marker of T_O. Agent a discovers the deletion at e_D, which must be on the path from the root of T_O to e_W and returns to the root of T_O to remove T_S from T_O map in the active root marker and mark the links in E_{OS} for verification. □

This result suggests that there is two circumstances when a deletion subtree is never detected: the subtree contains no agents and no work except for the links in E_{SO} or the subtree contains agents exploring frontier or black links and no other work except for the links in E_{SO}. We say such a deletion subtree is *empty*.

Lemma 5.8. *The links E_{OS} leading to an empty deletion subtree are never marked for work.*

Proof. The links in E_{OS} are only marked for work if an agent a tries to do work in T_S and discovers the deletion of link e_D. Since the work on E_{SO} only appears in the active root marker because of the deletion and there is no other work available in the tree, a has no work to do in T_S and the links in E_{OS} are never marked for verification. □

These *empty* trees do not affect the correctness of the algorithm.

Lemma 5.9. *The failure to detect a deletion subtree with no other work than E_{SO} does not affect the correctness of the algorithm.*

Proof. The lemma follows from the fact that the lack of work means that all the edges must have been explored and all the safe edges must have been verified. In other words, every edge has been visited at least once and the problem can been considered solve for T_S. □

The same is true of the deletion of non-tree links, although they can and do affect the complexity of the algorithm.

Lemma 5.10. *The deletion of a non-tree link does not stop the agent from completing its current work.*

Proof. By construction, an agent performs a single work task by moving along safe tree links to and from the single non-tree link that it has chosen to explore, verify, or merge across. If such a link is deleted, the agent simply returns to the active root marker of the current tree and its work is complete. The deletion of any other non-tree link has no impact on the agent doing work. □

We now need to deal with the assumption in Lemma 5.7 that there must be an agent in T_O that chooses to work on $e_W \in T_S$ in order to guarantee that the links in E_{OS} are marked for work and in Lemma 5.6 that there is an agent available to verify $[v, u]$ in the deletion subtree T_i, where $i \geq 1$. We show that before termination there must always be an agent working in the network and because that agent must eventually work on edges that cannot be deleted, every tree is eventually worked on.

Lemma 5.11. *At any time before termination, at least one agent is performing work.*

Proof. By contradiction, assume at time t that there is still work to be done outside of empty subtrees but no agent is performing work. By Property 4.1, we know that at least one agent must be alive. By Lemmas 4.4, 4.5, and 4.8, we know that in the absence of deletions all work is completed in finite time. By Lemma 5.4, we know that in the presence of deletions all aborted work is completed in finite time. Since verification and mergers are safe, by Property 4.2, we know that all verifications and mergers, including the aborted ones and the ones that make links internal in Lemma 5.6, must have been completed. The work that is still left to be done must be exploration. Furthermore, all live agents must be waiting for work. They can only be waiting if all available exploration work is being done. Since we assume that no work is being done, all the agents doing exploration work must have been eliminated by black holes or black links. However, by Lemma 5.5, we know that there is a safe border edge out of every tree with fewer nodes than n_S that cannot be deleted. By Lemmas 5.6 and 5.7, we know that the non-deletable safe border edge must be marked for work. There are two cases: multiple trees and a single tree. If there is more than one tree left, the elimination of all non-waiting agents means that all the links leading out of those trees have been deleted or are frontier or black links, which contradicts our assumption that the safe portion of the network remains connected despite deletions and dangerous elements. If there is a single tree, the elimination of all non-waiting agents implies the tree contains all safe nodes and that the remaining agents should have terminated, a contradiction. □

Corollary 5.12. *An agent is eventually available to do the work in T_O described in Lemma 5.7 and in deletion subtree T_i, where $i \geq 1$, described in Lemma 5.6.*

We can now prove the correctness of the algorithm.

Lemma 5.13. *Within finite time, all accessible links have been visited and all surviving agents terminate.*

Proof. By Lemma 4.9, we know that at any time t before termination there is work being done. By Lemmas 4.4, 4.5, 4.8, and 5.4, we know that each piece of work finishes within finite time. By Lemma 5.11, we know that there is always an agent working despite the deletions. As a result, within finite time, all accessible links have been visited and all frontier and black links have been marked locally as dangerous. □

Lemma 5.14. *After at most $O(k^2 \cdot n_S + n_S \cdot m + k \cdot n_S \cdot D)$ moves, all agents that are still alive terminate.*

Proof. We look at the cost for exploration, verification, and merging. We look first at the cost of each when there are no deletions and then at the cost for each per deletion. Let D be the number of deletions.

For exploration without deletions, every accessible link, up to m or $O(m)$, in the graph is explored. It requires at most $n-1$ moves to reach a link, 2 moves for cautious walk, and at most $n-1$ moves to return to the root. An agent that is eliminated travels at most $n-1$ or $O(n)$ moves. An agent that is not eliminated travels at most $2n$ or $O(n)$ moves. Therefore, the overall complexity of exploration without deletions is $O(nm)$.

For verification without deletions, we need look at the verification of internal and external links. When a verifying agent traverses an internal link, it takes up to $2n-1$ moves to do so. An agent traverses an internal link if only one end of the link appears in the root marker's map. Tree links do not require traversal to be verified. For all other links, up to $O(m)$ of them, the adversary can ensure that the information for one side of the link is always delayed, giving a complexity for internal link verification without deletions of $O(nm)$ moves.

When a verifying agent traverses an external link, it takes up to $2n-1$ moves to reach the root on the other side. For each of the possible $k-1$ mergers, there can be up to $2k$ links verified with k links verified from the higher id tree to the lower id tree and up to k links verified from lower id tree to the higher id tree, one of which ends in a merger. Therefore, the total cost for the external link verification without deletions is at most $2(2n-1)k(k-1)$ or $O(k^2n)$ moves.

For merging without deletions, there are at most k trees and at most $k-1$ possible merging agents. When a merging agent picks up a root marker, even if it follows the longest path, it can move at most $2n-1$ moves because all the links it passes over, except the verified link, are tree links. Each of the at most $k-1$ merging agents could take up to $O(n)$ moves to complete a merger, giving an overall complexity of $O(kn)$ moves for merging.

The cost of algorithm *ExploreDG-LD* when no links are deleted is $O(nm) + O(nm) + O(k^2n) + O(kn) = O(k^2n + nm)$ moves.

For exploration with deletions, the case when an agent successfully traverses the link before encountering the deletion is already handled by the non-deletion case. If the link is black or leads to a black hole, its deletion after eliminating the agent has no affect on the complexity. If the link and node are safe then any deletion encountered by the agent after it completes its exploration can only shorten the path it takes from the link it was exploring to the active root marker. We do need to account for the deletion of tree link on the path to the link being explored. The effort taken by the exploring agent is wasted since some agent must still explore that link. An agent that fails to traverse the link being explored can travel up to $n-2$ moves before it finds the deletion and up to $n-2$ moves to report the deletion or $2(n-2) = O(n)$. For D deletions, the additional cost for exploration is $O(nD)$ moves.

For verification with deletions, the case when an agent successfully traverses the link before encountering the deletion is already handled by the non-deletions case for essentially the same reasons. We need only account for the wasted effort of a verification agent that tries to get to a link and fails because it encounters a link deletion. The agent travels up to $n-2$ moves before it finds the deletion and up to $n-2$ moves to report it or $2(n-2) = O(n)$ moves total. For D deletions, the additional cost for internal link verification is $O(nD)$ moves.

For merging with deletions, the case when an agent successfully traverses back over the merger link, the link that was originally verified by the merging agent, is already handled by the non-deletions case because the merger is done with some active root marker on the other side. We need only worry about a deletion that stops a merger from ever taking place by keeping the root marker from leaving its tree. The merging agent can move at most $n-1 = O(n)$ moves before encountering the deletion. For D deletions, the additional cost for merging is $O(nD)$ moves.

Finally, we must account for the merging of all the subtrees created by deletions. The deletion of a tree link severs a tree into two subtrees that must eventually be merged. Just like the non-deletions merging case, the cost of finding the link to do the merger between two trees created by a deletion is $O(kn)$ moves. For D deletions, the additional cost for external link verification is $O(knD)$ moves.

The additional cost for algorithm *ExploreDG-LD* when there are D deletions is $O(nD) + O(nD) + O(nD) + O(knD) = O(knD)$ moves.

The total cost for algorithm *ExploreDG-LD* is therefore $O(k^2n+nm+knD)$ moves. Note, however, that only nodes in the safe portion of the network, G_S are connected by tree links, or n_S of them, so we can reduce the complexity to $O(k^2 \cdot n_S + n_S \cdot m +$

$k \cdot n_S \cdot D$) moves overall, proving the lemma. □

We can now look at the worst case cost of algorithm *ExploreDG-LD* and how it compares to algorithm *ExploreDG*. The number of deletions can range from $0 \leq D \leq m - n_S - 1$, since everything but the tree links connecting the safe nodes can be deleted. The number of faults can range from $0 \leq f \leq m - n_S - 1$, since all the links that can be deleted can also be black links or lead to black holes. The worst case for deletions is $D = O(m)$ and for faults is $f = O(m)$. The worst case for an optimal number of agents is $k = f + 1 = O(m)$. Taken together, we get a worst case complexity for algorithm *ExploreDG-LD* of $O(m^2 n + nm + mnm) = O(nm^2)$ moves. In comparison to algorithm *ExploreDG*, if there is a small constant number of deletions, $D = O(1)$, we still get the same worst case complexity for algorithm *ExploreDG-LD* of $O(m^2 n + nm + mn0) = O(nm^2)$ moves. If there is only a small number of faults and deletions, $f = D = O(1)$, the complexity is $O(1n + nm + 1n1) = O(nm)$, which is the same complexity as algorithm *ExploreDG*.

5.4 Conclusion

In this chapter, we presented a solution to the DGE-LD problem in the network model that is at least agent optimal. The DGE-LD problem (Definition 2.8) is to visit every accessible link in a network and mark locally the locations of black holes (Definition 2.5) and black links (Definition 2.6) while an adversary is allowed to delete links in the network under certain conditions. We presented a solution to the problem using a team of agents scattered in the network. We proved that the solution is correct and works with an optimal number of agents $k = f + 1$, where $f = |F_B| + 2|E_B \setminus (E_I \cup F_B)|$, F_B is the set of frontier links connecting safe nodes to black holes, and $E_B \setminus (E_I \cup F_B)$ is the set of accessible black links that are not also frontier links. We then proved that our solution has a complexity of $O(k^2 \cdot n_S + n_S \cdot m + k \cdot n_S \cdot D)$ moves, where k is the number of agents, n_S is the number of safe nodes, m is the number of links, and D is the number of deletions. When the number of faults or deletions is maximized, $f = O(m)$ or $D = O(m)$, our algorithm requires $O(nm^2)$ moves, where n is the number of nodes, which is its worst case complexity. When the number of faults and deletions is minimized, $f = D = O(1)$, our algorithm runs in $O(nm)$ moves, which is optimal.

CHAPTER 6

Basic Limitations and Lower Bound Complexity of Black Hole Search in the Subway Model

In this chapter, we look at the assumptions needed to solve the EXP and BHS problems in the subway model and we prove lower bounds on the complexity of any BHS solution. We present basic limitations on the assumptions needed to provide a solution to the both EXP and BHS problems in the subway model. Although the assumptions appear to make the subway model stronger than necessary, we show that each assumption is necessary for a solution to be viable. We then prove lower bounds on the complexity of any BHS solution in the subway model.

We review the subway model in Section 6.1. The model is presented in full detail in Section 2.3. We discuss basic limitations (impossibility results) on solutions to the EXP and BHS problems in the subway model in Section 6.2. These limitations require additional assumptions be made in order to make the problem solvable in the subway model. We prove a lower bound on any BHS solution of $\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves in Section 6.3, where γ is the number of stops

Problem	**Result**	**Reference**
EXP in a PV graph with homogenous routes	$O(n_C \cdot l_R)$	Flocchini et al. (2009b)
EXP in a PV graph with heterogenous routes	$O(n_C \cdot l_R^2)$	Flocchini et al. (2009b)
BHS Lower bound	$\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chapter 6, Flocchini et al. (2010b, 2012)
BHS by co-located agents with site whiteboards	$O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chapter 7, Flocchini et al. (2010b, 2012)
BHS by scattered agents with carrier whiteboards	$O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chapter 8
Note that $k \geq \gamma + 1$ for our subway model solutions.		

Table 6.1: Results related to lower bounds on EXP in the subway model.

that all carriers make on black hole sites, n_C is the number of carriers, and l_R is the length of the longest carrier route. Both our solutions use $k \geq \gamma + 1$ agents. We summarize the chapter in Section 6.4.

Related results include exploration in PV graphs and our own solutions to BHS in the subway model. Flocchini, Mans, and Santoro (2009b) give solutions for the EXP problem in PV graphs where the agents jump from carrier to carrier to explore a synchronous bus-like network. Their solutions works in $O(n_C \cdot l_R)$ moves when the carriers all have the same route length (the homogenous case) and $O(n_C \cdot l_R^2)$ moves when the carriers may have different route lengths (the heterogenous case). Our own solutions in Chapters 7 and 8 are optimal in that they meet the lower bound proven in Section 6.3. These results are summarized in Table 6.1.

6.1 Model

We summarize the subway model here. For a full description of the model see Section 2.3 and for a summary of the notation used see Table 2.2.

A set C of n_C carriers move asynchronously among a set S of n_S sites. Each carrier $c \in C$ follows a route $R(c)$ between the sites in its domain $S(c) = \{s_0, s_1, \ldots, s_{n_S(c)-1}\} \subseteq S$, where $n_S(c) = |S(c)|$. The route $R(c) = \langle r_0, r_1, \ldots, r_{l(c)-1} \rangle$ is a cyclic sequence of stops that the carrier visits in order. A route has a length $l(c) = |R(c)|$ and is called simple if its size of its domain is equal to the length of its route, $n_S(c) = l(c)$. A transfer site is any site that is in the domain of two or more carriers.

Each carrier's route defines a carrier graph, which is an edge-labelled directed multigraph $\vec{G}(c) = (S(c), \vec{E}(c), \lambda(c))$. The entire network defines a subway graph, which is an edge-labelled directed multigraph $\vec{G} = (R, \vec{E}, \lambda)$. Each subway graph has associated with it a transfer graph, which is an edge-labelled undirected multigraph $H(\vec{G}) = (C, E_T)$, where the nodes are the carriers and an edge connects two carriers if they share a transfer site.

Working in the network is a team A of k mobile agents that start at unpredictable times. An agent moves around the network by boarding any carrier stopping at its current site and disembarking at some other site the carrier stops at on its route. Agents perform computations asynchronously and communicate with one another using whiteboards located either on the sites or the carriers.

In the network are $n_B < n_S$ black hole sites that eliminate agents disembarking on them, but do not affect carriers. We define the number of stops that a carrier c makes at black hole sites as its faulty load $\gamma(c)$ and the total of all faulty loads in the systems as $\gamma(\vec{G})$ or simply γ where no ambiguity arises.

In addition, we make the following assumptions that we prove in Section 6.2 are necessary for the Exp and Bhs problems to be solvable in the subway model.

- Each carrier is labelled with a distinct id.
- The agents know the number of carriers n_C.
- Each transfer site is labelled with the number of carriers stopping there. Without loss of generality, since the agent knows the number of carriers and can wait for all the carriers to pass by, we assume that each transfer site is also labelled with the ids of the carriers. This additional assumption is not significant in terms of complexity.
- We assume that the standard assumptions for Bhs apply:
  - An agent's homebase is a safe site.
  - The transfer graph is connected when black holes are removed.
  - There are more agents than faulty stops ($k > \gamma$).
  - The agents know the number of faulty stops γ.

We make additional assumptions depending on whether the whiteboards are the sites or the carriers. For site whiteboards, we make the following assumptions proven in Section 7.1. Each carrier is labelled with the length of its route. Once disembarked, an agent can board a carrier at the same point in its route. For carrier whiteboards, we make the following assumption in Section 8.1. There is a function $r_{curr}(c)$ that returns a distinct id for the current stop when the agent is on carrier c. Without loss of generality, since the agent can record every stop during the carrier's traversal of the route, we assume that there is a second function $R(c)$

that returns the set of all stops on the route. This additional assumption is not significant in terms of complexity. The same function can be used to determine the length of the carrier's route.

6.2 Basic Limitations

There are some basic limitations for the BHS problem to be solvable in subway graphs; these in turn dictate some assumptions, which we listed at the end of the previous section. It may appear from the number of assumptions that the model is stronger than necessary. We prove here that each of these assumptions is necessary.

The BHS problem is a restricted version of the EXP problem. The subway EXP problem (Definition 2.11) is for an agent or agents to visit every stop on every carrier's route. A solution correctly solves the problem if at least one agent visits every stop on every carrier's route and all agents enter a terminal state. The subway BHS problem (Definition 2.12) is for the agents to explore and produce a map of the subway graph in the presence of black holes.

We start by looking at the EXP problem. We prove a number of impossibility results that show that the problem is deterministically unsolvable without a particular assumption. Essentially, we show that without the assumption it is possible to create a subway graph that fits the model but for which any solution to the EXP problem can be made to terminate before all the stops in the subway graph have been visited by an agent. We start by showing that carriers must have distinct ids that are visible to the agents.

Lemma 6.1. *If the carriers do not have distinct ids visible to the agents, the* EXP *problem is deterministically unsolvable. This result holds regardless of the number $k \geq 1$ of agents and even if the agents know n_C, n_S, and the length of the routes.*

Proof. By contradiction, let $\mathscr{P}$ be a deterministic protocol that always allows a team of $k > 0$ agents starting co-located on site s to explore all the sites of all subway graphs in which there are no black holes but the carriers do not have distinct identities visible to the agents. Consider now the subway graph $\vec{G}$, corresponding to the routes $R(c_1)$ and $R(c_2)$ of only two carriers c_1 and c_2, in which the only transfer site is the homebase s and $n_S(c_1) = n_S(c_2) > 1$. Since the two carriers have no visible identifiers, an agent at s cannot distinguish whether an arriving carrier is c_1 or c_2. An adversary can clearly choose the speed of the two carriers in such a way that, whenever an agent a at s decides to board the next carrier, the carrier arriving there is c_1. In other words, none of the agents will ever board c_2 and visit the sites reachable only through that route, contradicting the correctness of $\mathscr{P}$. □

Next notice that some metric information, either the number of carriers n_C or the number of sites n_S, must be be available to the agents for exploration, and thus black hole search, to be possible.

Lemma 6.2. *If the agents have no knowledge of n_C nor of n_S, the* Exp *problem is deterministically unsolvable. This result holds regardless of the number $k \geq 1$ of agents and even if the carriers have distinct visible ids.*

Proof. By contradiction, let $\mathscr{P}$ be a deterministic protocol that always allows a team of $k > 0$ agents starting co-located on site s to explore all the sites of all subway graphs in which there are no black holes without knowledge of n_C nor of n_S. Consider now an execution of $\mathscr{P}$ in the subway graph $\vec{G}$ corresponding to the route of a single carrier c. Since the protocol is correct, within finite time T all sites will have been visited and all agents enter a terminal state. Consider now the subway graph $\vec{G}'$ corresponding to the routes $R(c_1)$ and $R(c_2)$ of two carriers c_1 and c_2, where $R(c_1) = R(c)$, $n_S(c_2) > 1$, and the only transfer site is the homebase s. Consider now in $\vec{G}'$ precisely the same execution of $\mathscr{P}$ as in $\vec{G}$, in which the adversary delays the arrival of c_2 at s until time $T' > T$. Notice that by time T none of the other agents will discover the existence of c_2. Since neither n_C nor n_S are known, at time T the agents will notice no difference with the previous setting and will thus enter a terminal state without visiting the sites on the route of c_2, contradicting the correctness of the protocol. □

In other words, at least one of n_C and n_S must be known to the agents for the problem to be solvable.

Transfer sites play a crucial role in the connectivity of the system, with and without black holes. Knowing that a site is a transfer site is necessary but not sufficient; in fact an agent disembarking there must also know the number of distinct carriers stopping there.

Lemma 6.3. *If the agents have no knowledge of the number of carriers stopping at a site, the* Exp *problem is deterministically unsolvable. This result holds regardless of the number $1 \leq k < n_S - 1$ of agents, even if the carriers have distinct visible ids, and the agents know n_C, n_S and the length of the routes.*

Proof. By contradiction, let $\mathscr{P}$ be a deterministic protocol that always allows a team of $1 \leq k < n_S - 1$ agents to explore all the sites of all subway graphs, in which there are no black holes, when the agents can detect whether a site is a transfer site but without knowledge of the number of carriers stopping there. Consider now the subway graph $\vec{G}$ corresponding to the route of three carriers c_1, c_2, and c_3.

The routes of c_1, c_2 coincide except for the direction; i.e., $R(c_1) = \langle r_0, r_1, \ldots, r_{l-1}\rangle$ and $R(c_2) = \langle r_0, r_{l-1}, \ldots, r_1\rangle$ where $l = |R(c_1)| = |R(c_2)| = n_S - 1 > 1$. The routes of c_1, c_2 meet with the route of c_3 at a single transfer site x, where $l(c_3) = 2$. Notice that all the sites $r_0, r_1, \ldots, r_{l-1}$ are transfer sites between two carriers,except for x which is a transfer site between all three carriers. The agents can detect whether a site is a transfer site, but not the number of carriers stopping there. The adversary allows the agents to board both c_1 and c_2 and visit all the l sites on their route. Now, to complete the exploration, at least one agent must board c_3; to do so, an agent must wait at x until c_3 arrives; because of asynchrony, this might take a finite but unpredictable amount of time. Since the agents do not know which stop is x and since $k < n_S - 1 = l(c)$, they cannot wait at all the sites on the route of c_1 (or c_2) simultaneously. Hence the adversary can make the agents wait forever, contradicting the termination of every execution of $\mathscr{P}$. □

In other words, the agents must be able to determine the number of carriers stopping at a site for the problem to be solvable.

The next set of limitations are directly related to the nature of black hole search and are a direct extension to the subway graph model of the limitations existing for standard network models.

Lemma 6.4. *For the* BHS *problem to be deterministically solvable*

(i) an agent's homebase must be a safe site;

(ii) the transfer graph must stay connected once the black holes are removed;

(iii) $k > \gamma$;

(iv) γ must be known to the agents.

This result holds even if the carriers have distinct visible ids and the agents know n_C and n_S, the length of the routes, and the number of carriers stopping at each site.

Proof. Conditions (i) and (ii) trivially follow from the fact that, since solving the BHS problem requires visiting all carrier stops, it is clearly necessary that the safe stops are reachable from all homebases. Condition (iii) expresses the obvious fact that to solve BHS, it is necessary to have more agents than the number of stops at black holes. The necessity of condition (iv) follows from the requirement of knowledge of the stops leading to black holes when an agent enters a terminal state. Because of asynchrony, slow computation by an agent exploring a safe stop is indistinguishable from an agent having been eliminated by a black hole stop; hence if γ is not known to the agents, a surviving agent cannot decide whether to wait or enter a terminal state. □

We will refer to this set of conditions as *standard* for the BHS problem, and assume that they hold. Note that a corollary to condition (ii) is that if there is more than one carrier, i.e. $n_C > 1$, then each carrier must have at least one safe transfer site in its domain.

Summarizing, in light of the above impossibility results, we make the following necessary assumptions for BHS in the subway model. Each carrier is labelled with a distinct id (necessary by Lemma 6.1). The agents know the number of carriers n_C (one of the two values is necessary by Lemma 6.2). Each transfer site is labelled with the number of carriers stopping there (necessary by Lemma 6.3). Without loss of generality, since the agent knows the number of carriers and can wait for all the carriers to pass by, we assume that each transfer site is also labelled with the ids of the carriers. Finally, we assume that the standard assumptions for BHS apply (necessary by Lemma 6.4).

6.3 Lower bounds

We now establish some *lower bounds* on the worst case complexity of any protocol solving the BHS problem using the minimal number of agents.

Theorem 6.5. *For any α, β, γ, where $\alpha, \beta > 2$ and $1 < \gamma < 2\alpha\beta$, there exists a simple subway graph $\vec{G}$ with α carriers with maximum route length β and faulty load γ in which every protocol $\mathscr{P}$ solving the* BHS *problem requires $\Omega(\alpha^2 \cdot \beta \cdot \gamma)$ carrier moves in the worst case.*

Proof. Consider a subway graph $\vec{G}$ whose transfer graph is a line graph; all α routes are simple and have the same length β; there exists a unique transfer stop between neighbouring carriers in the line graph; no transfer site is a black hole, and the number of black holes is γ. The agents have all this information, but do not know the order of the carriers in the line (the agents do not know in what order the carriers will arrive).

Let $\mathscr{P}$ be a subway mapping protocol that always correctly solves the problem within finite time with the minimal number of agents $k = \gamma + 1$. Consider an adversary $\mathscr{A}$ playing against the protocol $\mathscr{P}$. The powers of the adversary are the following: 1) it can choose which stops are transfers and which are black holes; 2) it can "block" a site being explored by an agent (i.e., delay the agent exploring the stop) for an arbitrary (but finite) amount of time; 3) it can choose the order of the carriers in the line graph. The order of the carrier will be revealed to the agents incrementally, with each revelation consistent with all previous ones; at the end the entire order must be known to the surviving agents.

Let the agents start at the homebase on carrier c_1. Let $q = \left\lceil \frac{k-2}{\beta-2} \right\rceil$. Assume that the system is in the following configuration, which we shall call Flip(i), for some

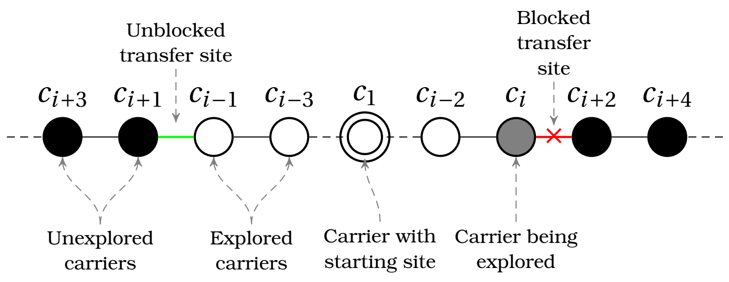

Figure 6.1: Transfer graph used in Theorem 6.5.

$i \geq 1$: (1) carrier c_1 is connected to c_2, and carrier c_j ($j < i$) is connected to c_{j+2}; (2) all stops of carriers $c_1, c_2, \ldots, c_i$ have been explored, except the transfer stop r_{i+1}, leading from carrier c_{i-1} to carrier c_{i+1}, and the stop r_{i+2} on carrier c_{i+1}, which are currently being explored and are blocked by the adversary; and (3) all agents, except the ones blocked at stops r_{i+1} and r_{i+2}, are on carrier c_i. See Figure 6.1.

If the system is in configuration Flip(i), with $i < \alpha - q$, the adversary operates as follows.

1. The adversary unblocks r_{i+1}, the transfer site leading to carrier c_{i+1}. At this point, all $k-1$ unblocked agents (including the $k-2$ currently on c_i) must move to r_{i+1} to explore c_{i+1} without waiting for the agent blocked at r_{i+2} to come back. To see that all must go within finite time, assume by contradiction that only $1 \leq k' \leq k-2$ agents go to explore c_{i+1} within finite time, while the others never go to r_{i+1}. In this case, the adversary first reveals the order of the carriers in the line graph by assigning carrier c_j to be connected to c_{j+1} for $\alpha > j > i$. Then the adversary chooses to be black holes: r_{i+2}, the first k' non-transfer stops visited by the k' agents, and other $k-k'-2$ non-transfer stops arbitrarily chosen in those carriers. Notice this can be done because, since $q = \left\lceil \frac{k-2}{\beta-2} \right\rceil$, the number of non-transfer stops among these carriers is $q(l-2)+1 \geq k-1$. Thus all k' agents will enter a black hole. Since none of the

other agents will ever go to c_{i+1}, the mapping will never be completed. Hence, within finite time all $k-1$ non blocked agents must go to r_{i+1}, with a total cost of $O(k \cdot i \cdot \beta)$ carrier moves.

2. The adversary blocks each stop of c_{i+1} being explored, until $k-1$ stops are being explored. At that point, it unblocks all those stops except one, r_{i+3}. Furthermore, it makes r_{i+2} the transfer stop leading to carrier c_{i+2}.

Notice that after these operations, the system is precisely in configuration Flip$(i+1)$. Further observe now that, from the initial configuration, when all agents are at the homebase and the protocol starts, the adversary can create configuration Flip(0) by simply blocking the first two stops of c_1 being explored, and making one of them the transfer to c_2.

In other words, within finite time, the adversary can create configuration Flip(0); it can then transform configuration Flip(i) into Flip$(i+1)$, until configuration Flip$(\alpha - q - 1)$ is reached.

At this point the adversary reveals the entire graph as follows: it unblocks $r_{\alpha-q+1}$, the transfer site leading to carrier $c_{\alpha-q+1}$; it assigns carrier c_j to be connected to c_{j+1} for $\alpha > j > \alpha - q$; finally it chooses $k-1$ non-transfer stops of these carriers to be black holes; notice that they can be chosen because, since $q = \left\lceil \frac{k-2}{\beta-2} \right\rceil$, the number of non-transfer stops among these carriers is $q(l-2)+1 \geq k-1$.

The transformation from Flip(i) into Flip$(i+1)$ costs the solution protocol $\mathscr{P}$ at least $\Omega(k \cdot i \cdot \beta)$ carrier moves, and this is done for $1 \leq i \leq \alpha - q$; since $\alpha(l-2) \geq (k-2)$ it follows that $\alpha - q = \alpha - \left\lceil \frac{k-2}{\beta-2} \right\rceil \geq \alpha - \frac{k-2}{\beta-2} \geq \frac{\alpha}{2}$; hence, $\sum_{1 \leq i \leq \alpha - q} i = \Omega(\alpha^2)$. In other words, the adversary can force any solution protocol to use $\Omega(\alpha^2 \cdot \beta \cdot \gamma)$ carrier moves. □

Theorem 6.6. *For any α, β, γ, where $\alpha, \beta > 2$ and $1 < \gamma < \beta - 1$, there exists a simple subway graph $\vec{G}$ with α carriers with maximum route length β and faulty load γ in which every protocol $\mathscr{P}$ solving the* Bhs *problem requires $\Omega(\alpha \cdot \beta^2)$ carrier moves in the worst case. This result holds even if the transfer graph $H(\vec{G})$ is known to the agents.*

Proof. Consider a subway graph $\vec{G}$ whose transfer graph is a line graph, where c_i is connected to c_{i+1}, $1 \le i < \alpha$; all α routes are simple and have the same length β; there exists a unique transfer stop between neighbouring carriers in the transfer graph; no transfer site is a black hole and the number of black holes is γ. The agents have all this information, including the order of the carriers in the line. Let $\mathscr{P}$ be a subway mapping protocol that always correctly solves the problem within finite time. Correctness of $\mathscr{P}$ implies that all stops are explored. If the stop is not a black hole, after exploring it, the agent must take a carrier; the adversary can delay this operation ensuring that $\beta - 1$ carrier moves are elapsed from the time the agent starts waiting to the time the carrier arrives. Thus the adversary can ensure that the execution of protocol $\mathscr{P}$ costs at least $m\ (\beta - 1)$ carrier moves, where m is the number of safe stops in the subway graph. Since the total number of stops is $\alpha(\beta-2)+2$ and $\gamma \le \beta - 2$, it follows that $m\ (\beta-1) = (\alpha(\beta-2)+2-\gamma)(\beta-1) \ge (\alpha(\beta-2)+2-(\beta-2))(\beta-1) = ((\alpha-1)(\beta-2)+2)(\beta-1)$, and the theorem holds. □

Since α is the number of carriers, n_C, β is the length of the longest route, l_R, and γ is the faulty load, we get a combined lower bound of $\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves.

6.4 Conclusion

In this chapter, we showed that a number of basic limitations apply to all solutions to the Exp and Bhs problems in the subway model. The subway Exp problem (Definition 2.11) is for an agent or agents to visit every stop on every carrier's route. The Bhs problem (Definition 2.12) is to construct a map of a subway graph that shows where carriers stop at black holes (Definition 2.9). We then proved a lower bound on any possible solution to the Bhs problem of $\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, where γ is the faulty load or number of black hole stops, n_C is the number of carriers, and l_R is the length of the longest route.

CHAPTER 7

Black Hole Search by Co-located Agents in the Subway Model with Site Whiteboards

In this chapter, we look at the BHS problem in the subway model with co-located agents and site whiteboards. The BHS problem (Definition 2.12) is to construct a map of a subway graph that shows where carriers stop at black holes (Definition 2.9). We present a solution to the BHS problem in arbitrary subway graphs of unknown topology with an arbitrary number of black holes and with asynchronous co-located agents and asynchronous carriers. A preliminary version of these results was presented at FUN 2010 (Flocchini et al., 2010b) and a more detailed version appeared in *Theory of Computing Systems* (Flocchini et al., 2012).

We review the subway model in Section 7.1. The model is presented in full detail in Section 2.3. We provide a solution to the BHS problem in the co-located agent, site whiteboard version of the subway model. The algorithm is presented in Section 7.2 and its correctness and complexity are presented in Section 7.3. We summarize the chapter in Section 7.4.

Problem	**Result**	**Reference**
BHS Lower bound	$\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chapter 6, Flocchini et al. (2010b, 2012)
BHS by co-located agents with site whiteboards	$O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chapter 7, Flocchini et al. (2010b, 2012)
BHS by scattered agents with carrier whiteboards	$O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chapter 8
Note that $k \geq \gamma + 1$ for our subway model solutions.		

Table 7.1: Results related to BHS by co-located agents in the subway model with site whiteboards.

The solutions in this chapter and in Chapter 8 are optimal in that they meet the lower bound proven in Section 6.3 as summarized in Table 7.1.

7.1 Model

We summarize the subway model here. For a full description of the model see Section 2.3 and for a summary of the notation used see Table 2.2. In this chapter, we assume that the agents start co-located on the same site in the network and communicate with each other using whiteboards located on the sites.

A set C of n_C carriers move asynchronously among a set S of n_S sites. Each carrier $c \in C$ follows a route $R(c)$ between the sites in its domain $S(c) = \{s_0, s_1, \dots, s_{n_S(c)-1}\} \subseteq S$, where $n_S(c) = |S(c)|$. The route $R(c) = \langle r_0, r_1, \dots, r_{l(c)-1}\rangle$ is cyclic sequence of stops that carrier visits in order. A route has a length $l(c) = |R(c)|$ and is called simple if the size of its domain is equal to the length of its route, $n_S(c) = l(c)$. A transfer site is any site that is in the domain of two or more carriers.

Each carrier's route defines a carrier graph, which is an edge-labelled directed multigraph $\vec{G}(c) = (S(c), \vec{E}(c), \lambda(c))$. The entire network defines a subway graph, which is an edge-labelled directed multigraph $\vec{G} = (R, \vec{E}, \lambda)$. Each subway graph has associated with it a transfer graph, which is an edge-labelled undirected multigraph $H(\vec{G}) = (C, E_T)$, where the nodes are the carriers and an edge connects two carriers if they share a transfer site.

Working in the network is a team A of k mobile agents that start co-located at unpredictable times. An agent moves around the network by boarding any carrier stopping at its current site and disembarking at some other site the carrier stops at on its route. Agents perform computations asynchronously and communicate with one another using whiteboards located on the sites.

In the network are $n_B < n_S$ black hole sites that eliminate agents disembarking on them, but do not affect carriers. We define the number of stops that a carrier c makes at black hole sites as its faulty load, $\gamma(c)$ and the total of all faulty loads in the systems as $\gamma(\vec{G})$ or simply γ where no ambiguity arises.

In addition, we make the following assumptions that we proved in Section 6.2 are necessary for the EXP and BHS problems to be solvable in the subway model. Each carrier is labelled with a distinct id. The agents know the number of carriers n_C. Each transfer site is labelled with the number of carriers stopping there. Without loss of generality, since the agent knows the number of carriers and can wait for all the carriers to pass by, we assume that each transfer site is also labelled with the ids of the carriers. Finally, we assume that the standard assumptions for BHS apply: an agent's homebase is a safe site, the transfer graph is connected when black holes are removed, there are more agents than faulty stops ($k > \gamma$), and the agents know the number of faulty stops γ.

These assumptions are not sufficient in the site whiteboard version of the subway model. We need at least two other assumptions to allow EXP and BHS to be solved by an optimally-sized team of agents. First, we prove that the agents must have knowledge of the length of each route.

Lemma 7.1. *Let the standard conditions for the* BHS *problem hold. If the agents have no knowledge of the length of each route in the subway graph, the* BHS *problem is deterministically unsolvable by a set of $k = \gamma + 1$ agents. This result holds even if the carriers have distinct visible ids and the agents know n_C and n_S.*

Proof. By contradiction, let $\mathscr{P}$ be a deterministic protocol that always allows a team of $k = \gamma + 1$ agents to solve the BHS problem in all subway graphs under the standard conditions without knowledge by the agents the length of each route in the subway graph. Consider the subway graph $\vec{G}$ consisting of a single route defined on-line by an adversary as follows. The carrier route is initially a sequence of distinct sites starting from s: $R = \langle s, s_1, s_2 \ldots \rangle$; the adversary will execute $\mathscr{P}$ until each agent descends at a stop (they must); let $x_1, ..., x_\gamma, x_{\gamma+1}$ be these stops, with x_1 the closest to s. The adversary sets $x_1, ..., x_{\gamma-1}$ to be black hole stops. Clearly only the agents a stopping at $x_\gamma = x$ and b stopping at $x_{\gamma+1} = y$ survive (for the moment) while all others are destroyed. When ready to move, agent a makes a decision on where to go based on algorithm $\mathscr{P}$; since the length of the route is not known, for any algorithm, this decision can be only of the form: wait for the w_a-th carrier passage, board the carrier, descend at the m_a-th stop. Similarly for b. The adversary adds to the route the stops $(s, x)^{w_a}, z_1, z_2, \ldots, z_{(m_a-1)}, z, (s, y)^{w_b}, z_1, z_2, \ldots, z_{(m_b-1)}, z$, where z_i are additional sites, and α^f denotes f consecutive occurrences of the subsequence α. In other words, the adversary finalizes the route as follows:

$$\begin{aligned} R = \langle & s, \ldots, x, \ldots, y, \\ & (s, x)^{w_a}, z_1, z_2, \ldots, z_{(m_a-1)}, z, \end{aligned}$$

$$(s,y)^{w_b}, z_1, z_2, \ldots, z_{(m_b-1)}, z\rangle$$

The adversary then makes a and b ready to move before the carrier reaches x for the second time; in this way both agents stop at z which is chosen by the adversary as a black hole stop. Hence all agents are destroyed, contradicting the correctness of $\mathscr{P}$. □

Even if the length of each route is known to the agents, this condition alone is not sufficient for black hole search with an optimal number of agents. In fact, in the site whiteboard version of the subway model, once disembarked, an agent must be able to board the same carrier at the same point in its route; otherwise, the problem is unsolvable.

Lemma 7.2. *Let the standard conditions for the* BHS *problem hold. Let inter-agent communications be via whiteboards located on the sites. Unless each agent, once disembarked, is able to board the same carrier at the same point in its route, the* BHS *problem is deterministically unsolvable by a set of $k = \gamma + 1$ agents. This result holds even if the carriers have distinct visible ids and the agents know n_C, n_S and the length of each route.*

Proof. By contradiction, let $\mathscr{P}$ be a deterministic protocol that always allows a team of $k = \gamma + 1$ agents to solve the BHS problem in all subway graphs under the standard conditions even if the agents have no means, once disembarked from a carrier, to board the same carrier at the same point in its route. Consider a subway graph $\vec{G}$ consisting of a single route $\langle v_0, v_1, \ldots, v_{(l(c)-1)}\rangle$ where there is only one black hole stop; the team thus consists of two agents, a and b both starting from the homebase $v_0 = s$. The location of the black hole stop and the nature of the other sites in the route is decided on-line by an adversary as follows. The adversary executes $\mathscr{P}$ until each agent descends at a stop (they must); let a descend at v_i and b at v_j, where without loss of generality $i < j$. If ready to move, agent a will make a decision on where to go based on algorithm $\mathscr{P}$; since it is unable to board the carrier at the same point in its route, this decision will be of the form: wait for w_a-th passage, board the carrier, descend at the m_a-th stop. Similarly for b.

If $m_a \neq j$ then the adversary defines that, in the route, $v_{(j-m_a)} = x = v_i$, where the operations on the indices are modulo $l(c)$; that is $v_{(j-m_a)}$ is the same site where a is currently stopped. This means that by activating a again when the carrier reaches $v_{(j-m_a)}$, a will stop at v_j, the site where b is currently stopped. By choosing v_j as the black hole site, the adversary makes both agents disappear, contradicting the correctness of $\mathscr{P}$. The same argument can be used if $m_a = j$ but $m_b \neq i$ (just exchange a and b, and i and j).

If both $m_a = j$ and $m_b = i$, let $p \in \{1, \ldots, n_S - 1\} \setminus \{i, j\}$ be such that $p - i \neq j$ and $p - j \neq i$ where the operations are modulo $l(c)$; such an index p always exists for $l(c) > 9$. Notice that by definition of p, $v_{(p-j)} \neq s \neq v_{(p-i)}$ and, since $m_b = i < j = m_a$, then $v_{(p-j)} \neq v_{(p-i)}$. The adversary then defines that, in the route, $v_{(p-j)} = x = v_i$ and $v_{(p-i)} = y = v_j$; in other words, $v_{(p-j)}$ and $v_{(p-i)}$ are the same sites where a and b are currently stopped, respectively. This means that, by activating a again when the carrier reaches $v_{(p-j)}$ and activating b when the carrier reaches $v_{(p-i)}$, both a and b will stop at v_p; notice that by definition v_p is neither s, nor x nor y. By choosing v_p as the black hole site, the adversary makes both agents disappear, contradicting the correctness of $\mathscr{P}$, and proving the lemma. □

As a result, we assume that each carrier is labelled with the length of its route (necessary by Lemma 7.1), and, once disembarked, an agent can board a carrier at the same point in its route (necessary by Lemma 7.2).

7.2 Algorithm

In this section, we present the proposed algorithm *SubwayExploreSWB*; as we will show later, our algorithm works correctly with any number of agents $k \geq \gamma + 1$ and is cost optimal. We first describe how the algorithm works in general, followed by a more detailed description.

7.2.1 Overview

Algorithm *SubwayExploreSWB* works as follows. The agents start at unpredictable times from the same site s, called the *homebase*, and collectively search all the carriers,[1] by visiting all their stops, looking for black holes. Each agent performs a series of search tasks; each task, called *work*, involves visiting a previously unexplored stop on a carrier's route and returning, if possible, to report what was found there, in our adaptation of the cautious walk technique to the subway model.

Every carrier is searched starting from a *work site*; the work sites are organized into a logical *work tree* that is rooted in the homebase. The first agent to access the homebase's whiteboard initializes the homebase as a work site (Section 7.2.3). It and the agents awaking after it then begin to work by visiting the stops of the carriers stopping at the homebase (Section 7.2.4). If an exploring agent finds a previously unexplored transfer site, the agent "competes" to add the transfer site to the work tree. If the agent succeeds, the transfer site becomes a work site for

[1]We use the terminology of searching a *carrier* to mean the searching of the route travelled by that carrier.

some or all of the other carriers stopping at it and the work site from which it was discovered becomes the work site's parent in the work tree (Section 7.2.5).

When the carrier that the agent is exploring has no more unexplored stops, the agent tries to find work by walking the work tree looking for another carrier with work to be done. The agent looks for work in the subtree rooted in its current work site and if there is no work available it moves to the work site's parent and tries again (Section 7.2.6). An agent terminates if it is at the homebase, there is no work, and there are n_C carriers in the work tree.

7.2.2 Example subway graph

Throughout the rest of this chapter, we use the subway graph presented in Table 7.2 as an example to help explain how our proposed solution works. Figure 7.1 shows a view of the subway graph $\vec{G}$. The routes of carriers c_3 and c_5 are simple while the other carriers' routes are not. Figure 7.2 shows the transfer graph associated with the example. Note that the transfer graph remains connected when the black holes are removed, including transfer site 13. Figure 7.3 shows the carrier graphs of the routes $R(c)$ for each carrier c. Even though there are only 3 black holes (nodes 6, 12, 13), the routes show that there are $\gamma = 8$ black hole stops in the network, meaning that our solution uses $k = \gamma + 1 = 9$ agents to map this subway, which is optimal.

7.2.3 Initialization

When an agent awakes for the first time on the homebase, it tries to initialize the homebase as a work site. Only the first agent accessing the whiteboard succeeds

Carrier	**Route**	**Length**	**Domain**	**Size**
c	$R(c) = \langle r_0, \ldots, r_{l(c)-1} \rangle$	$l(c) = \lvert R(c) \rvert$	$S(c) = \{s_0, \ldots, s_{n_S(c)-1}\}$	$n_S(c) = \lvert S(c) \rvert$
c_1	$\langle 1,3,3,2,4,4,2,6,1,2,5,6 \rangle$	12	$\{1,2,3,4,5,6\}$	6
c_2	$\langle 1,9,10,9 \rangle$	4	$\{1,9,10\}$	3
c_3	$\langle 5,7,8,15,13 \rangle$	5	$\{5,7,8,13,15\}$	5
c_4	$\langle 9,11,13,14,12,10,12,14,13,11 \rangle$	10	$\{9,10,11,12,13,14\}$	6
c_5	$\langle 14,13,15,16 \rangle$	4	$\{13,14,15,16\}$	4
Homebase: $s = 1$; Black holes: $6,12,13$, $n_B = 3$; Transfer sites: $1,5,9,10,13,14,15$				

Table 7.2: Example graph in subway model.

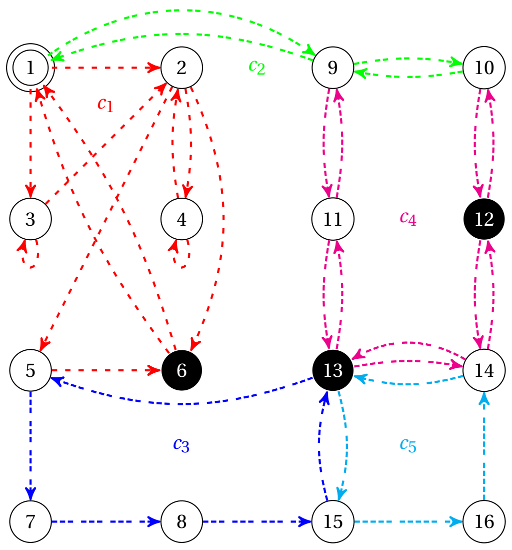

Figure 7.1: Subway graph of example from Table 7.2. Nodes in black are black holes; doubly-circled node is the homebase.

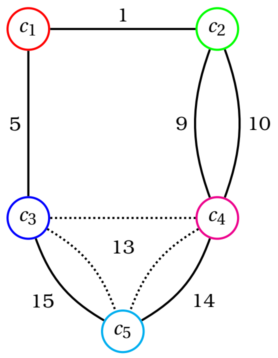

Figure 7.2: Transfer graph $H(\vec{G})$ of example from Table 7.2. Edge labels are corresponding transfer site ids. Transfer site 13 is a black hole.

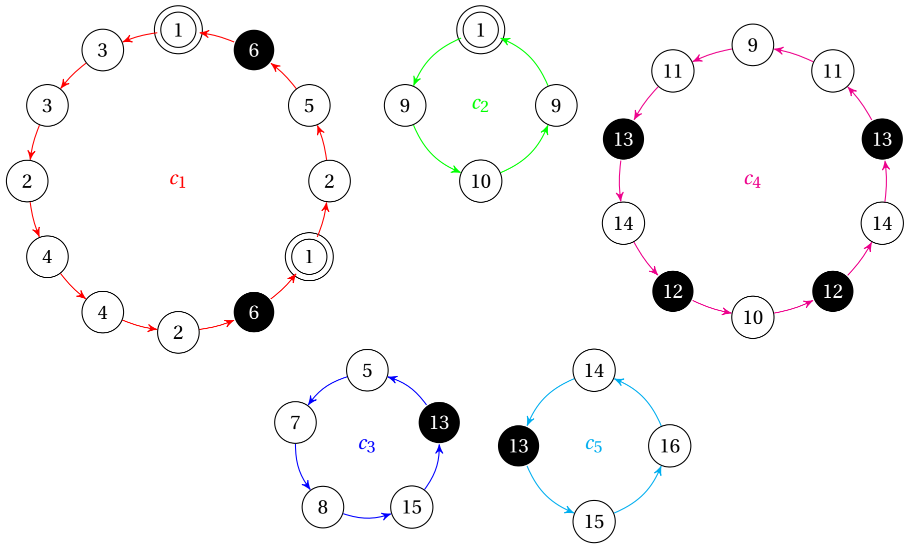

Figure 7.3: Carrier graphs for carriers in example in Table 7.2.

and executes the Initialize Work Site procedure. All other agents proceed directly to trying to find work.

The Initialize Work Site procedure is used to set up each work site in the work tree. The procedure takes as input the parent of the work site and the carriers to be worked on or serviced from the work site. For the homebase, the parent is *null* and the carriers to be accessed are all those stopping at s. The procedure initializes the work site's whiteboard with the information needed to find work, do work, and compete to add work. More precisely, when a work site ws is initialized, its parent is set to the work site from which it was discovered (*null* in the homebase's case) and its children are initially *null*. The carriers it will service are added to $C_{subtree}$, the set of carriers in the work tree at and below this work site. The same carriers are also added to C_{work}, the set of carriers in the subtree with unexplored stops, and C_{local}, the set of carriers serviced specifically by this work site. For each carrier c added to C_{local}, the agent setting up the whiteboard creates three sets U_c, D_c, and E_c. The set E_c of *explored stops* is initialized with the work site at $r_0 = ws$ (r_0 is always the work site servicing the carrier). The set U_c of *unexplored stops* is initialized with the rest of the stops on the carrier's route $\{r_1, r_2, \ldots, r_{l(c)-1}\}$, which is possible because each carrier is labelled with its length as well as its id. The set D_c of *stops being explored* (and therefore potentially dangerous sites) is initially empty. The pseudocode for initialization is in Algorithm 7.1.

7.2.4 Do work

We now discuss how the agents do their exploration of unexplored stops. To limit the number of agents eliminated by black holes, we use a technique similar to the cautious walk technique used by black hole search algorithms in static networks.

Algorithm 7.1 *SubwayExploreSWB*

```
    Agent $a$ awakes on starting site $s$.
    ▷ Initialize work information on whiteboard
 1: if whiteboard is blank then
 2:     INITIALIZE WORK SITE($null$, carriers stopping at $s$)
 3: end if
 4: FIND WORK

    Agent $a$ is initializing the whiteboard of work site $ws$ with information needed to find
    work, do work, and compete to add work.
 5: procedure INITIALIZE WORK SITE(parent $p$, carriers $C$)
 6:     for $c \in C$ do
            ▷ Work site information (Do work)
 7:         $U_c \leftarrow \{r_1, r_2, ...r_{l(c)-1}\}$          ▷ Set of $c$'s unexplored stops
 8:         $D_c \leftarrow \emptyset$                     ▷ Set of $c$'s stops being explored
 9:         $E_c \leftarrow \{r_0\}$       ▷ Set of $c$'s explored stops where $r_0$ is the work site
            ▷ Work competition information (Compete to add work)
10:         $C_{subtree} \leftarrow C_{subtree} \cup \{c\}$     ▷ Set of carriers in subtree rooted in current node
            ▷ Work tree information (Find work)
11:         $C_{local} \leftarrow C_{local} \cup \{c\}$     ▷ Add $c$ to set of carriers worked on from this work site
12:         $C_{work} \leftarrow C_{work} \cup \{c\}$   ▷ Add $c$ to set of carriers in subtree with unexplored stops
13:     end for
        ▷ Work tree information (Find work)
14:     $parent \leftarrow p$
15:     $children \leftarrow \emptyset$
16: end procedure
```

Consider an agent a on the work site *ws* of a carrier c that still has unexplored stops, i.e. $U_c \neq \emptyset$. The agent does the following. It chooses an unexplored stop $r \in U_c$ for exploration, removes r from U_c, and adds it to the set D_c of stops being explored. It then takes c to r and disembarks.

If r is a black hole, the agent is eliminated.

If r is not a black hole, the agent marks the site as *visited* on its whiteboard and returns to *ws* using the same carrier c and disembarks. The agent can make the trip back to *ws* because, in addition to being able to get back on the carrier at exactly the same point in its route, it knows the index of r and the length of c's

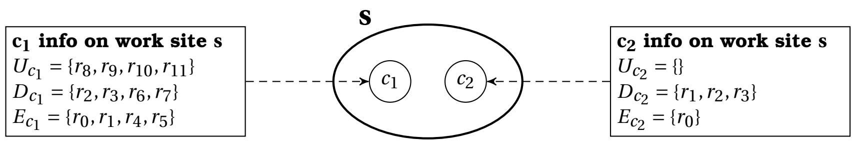
s
c_1 info on work site s
$U_{c_1} = \{r_8, r_9, r_{10}, r_{11}\}$
$D_{c_1} = \{r_2, r_3, r_6, r_7\}$
$E_{c_1} = \{r_0, r_1, r_4, r_5\}$
c_1
c_2
c_2 info on work site s
$U_{c_2} = \{\}$
$D_{c_2} = \{r_1, r_2, r_3\}$
$E_{c_2} = \{r_0\}$

Figure 7.4: Work site information from example in Table 7.2. Agent is doing work in example subway graph at some instant.

route, $l(c)$, and can therefore calculate the number of stops between r and ws. At ws, it removes r from D_c and adds it to the set E_c of explored stops. At this point, the agent also adds the site id and any other information of interest.

If r is a transfer site and a is the first to visit it (its whiteboard is not marked *visited*), then, before returning to ws, the agent proceeds as follows. Recall that we assume that the ids of the carriers are recorded on the transfer sites. The agent initializes two sets in its own memory: the set of *new carriers* initially containing all the carriers stopping at r; and the set of *existing carriers*, initially empty. Upon returning to ws and updating the local information, the agent then executes the next procedure that we discuss: competing to add work. The Do Work procedure is in Algorithm 7.2.

Figure 7.4 shows an example of the work site information at some instant used by agents to do work from the root in our example subway graph. At this point in the execution, carrier c_1 has 4 unexplored stops, 4 stops being explored that could potentially be black holes, and 4 stops that have been explored and are known not to be black holes. Carrier c_2 has no unexplored stops, 3 stops being explored, and one stop that has been explored, which happens to be the homebase in this case.

Algorithm 7.2 Do work

```
    Agent $a$ is working on carrier $c$ from work site $ws$.
17: procedure DO WORK(carrier $c$)
18:     while $U_c \neq \emptyset$ do
19:         choose a stop $r$ from $U_c$
20:         $U_c \leftarrow U_c \setminus \{r\}$          ▷ Remove $r$ from the set of unexplored stops
21:         $D_c \leftarrow D_c \cup \{r\}$          ▷ Add $r$ to the set of stops being explored
22:         take $c$ to $r$ and disembark
            ▷ If not eliminated by black hole
23:         if whiteboard is blank then
24:             mark node as *visited*
25:             if $r$ is a transfer site then
26:                 $a.newC \leftarrow \emptyset$          ▷ Initialize agent's set of new carriers
27:                 $a.existingC \leftarrow \emptyset$          ▷ Initialize agent's set of existing carriers
28:                 for each carrier $c$ stopping at $r$ do
29:                     record $c$ on whiteboard
30:                     $a.newC \leftarrow a.newC \cup \{c\}$          ▷ Add carrier to agent's set of new carriers
31:                 end for
32:             end if
33:         end if
34:         take $c$ to $ws$ and disembark
35:         $D_c \leftarrow D_c \setminus \{r\}$          ▷ Remove $r$ from the set of stops being explored
36:         $E_c \leftarrow E_c \cup \{r\}$          ▷ Add $r$ to the set of explored stops
37:         if $r$ was a transfer site then
38:             COMPETE TO ADD WORK
39:         end if
40:     end while
41: end procedure
```

7.2.5 Compete to add work

When an agent a discovers that a stop r is an unvisited transfer site, that stop is a potential new work site for the other carriers stopping at it. There is a problem, however: other agents may have independently discovered some or all of those carriers stopping at r. To ensure that to each carrier there is only one associated work site in the work tree, in our algorithm agent a must compete with all those other agents to add r as the new work site in the tree for these carriers. We use $C_{subtree}$ on the work sites in the work tree to decide the competition (if any).

Let us describe the actions that agent a performs; let a have just finished exploring r on carrier c_{ws} from work site ws and found that r is a new transfer site. The agent has a set of *new carriers* that initially contains all the carriers stopping at r, a set of *existing carriers* that is initially empty, and is currently on its work site ws. The agent walks up the work tree from ws to s checking the set of new carriers against $C_{subtree}$ on each work site. If a new carrier is not in $C_{subtree}$, the agent adds it. If a new carrier is in $C_{subtree}$, the agent moves it to the set of existing carriers. The agent continues until it reaches s or its set of new carriers is empty. The agent then walks down the work tree to ws. It adds each carrier in its set of new carriers to C_{work} on each work site on the way down to ws. For each carrier in its set of existing carriers, it removes the carrier from $C_{subtree}$ if it was the agent that added it. When it reaches ws, it removes the existing carriers and if there are no new carriers, it continues its work on c_{ws}. If there are new carriers, the agent adds r as a child of ws and goes to r. At r, the agent initializes it as a work site using the INITIALIZE WORK SITE procedure with ws as its parent and the set of new carriers as its carriers. The agent then returns to ws and continues its work on c_{ws}. The

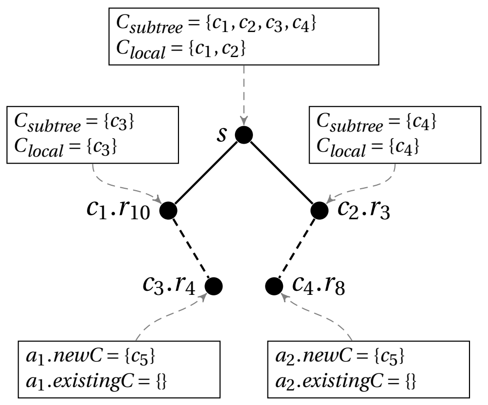

Figure 7.5: Work competition information for example from Table 7.2. Agent is competing to add work at some instant after Figure 7.4.

pseudocode for the Compete to Add Work procedure is in Algorithm 7.3.

Figure 7.5 shows the work competition information at some instant used by two agents to compete to add a new work site servicing carrier c_5 to the work tree for our example subway graph. Agents a_1 and a_2 have discovered carrier c_5 independently from carriers c_3 and c_4 respectively. The agents will walk up the tree towards s adding c_5 to $C_{subtree}$ on the way. Only one of the agents will be able to add c_5 to s's $C_{subtree}$ and that agent's site will become the new work site for carrier c_5

The Compete to Add Work procedure ensures the acyclic structure of the work tree and that all new work is reported to the root:

Lemma 7.3. *If a new carrier is discovered, within finite time it is added to the work tree and the new work is reported to the root.*

Proof. By construction, all the steps taken by an agent after finding a new carrier are safe. They involve either moving on the carrier, which is immune to black holes, or switching carriers at a work site in the work tree, which must be safe. Hence each move will be completed in finite time. Let c be a newly discovered carrier.

Algorithm 7.3 Compete to add work

Agent a has found a new transfer site r while exploring carrier c_{ws} from work site ws and is competing to add it to the work tree with ws as r's parent.

```
42: procedure Compete to Add Work
        ▷ Walk up tree
43:     repeat
44:         take the appropriate carrier to $parent$ and disembark
45:         for $c \in a.newC$ do
46:             if $c \in C_{subtree}$ then
47:                 $a.newC \leftarrow a.newC \setminus \{c\}$        ▷ Remove from agent's set of new carriers
48:                 $a.existingC \leftarrow a.existingC \cup \{c\}$        ▷ Add to agent's set of existing carriers
49:             else
50:                 $C_{subtree} \leftarrow C_{subtree} \cup \{c\}$
51:             end if
52:         end for
53:     until (on $s$) $\vee$ $(a.newC = \emptyset)$
        ▷ Walk down tree
54:     while not on $ws$ do
55:         for $c \in a.newC$ do
56:             $C_{work} \leftarrow C_{work} \cup \{c\}$        ▷ Add new carriers with work in subtree
57:         end for
58:         for $c \in a.existingC$ do
59:             if $a$ added $c$ to $C_{subtree}$ then
60:                 $C_{subtree} \leftarrow C_{subtree} \setminus \{c\}$        ▷ Remove carrier from subtree set
61:             end if
62:         end for
63:         take appropriate carrier to $child$ in direction of $ws$ and disembark
64:     end while
        ▷ Remove any existing carriers on $ws$
65:     for $c \in a.existingC$ do
66:         if $a$ added $c$ to $C_{subtree}$ then $C_{subtree} \leftarrow C_{subtree} \setminus \{c\}$ ▷ Remove carrier from subtree
67:     end for
        ▷ Add any new carriers to the tree with $r$ as their work site
68:     if $a.newC \neq \emptyset$ then
69:         $children \leftarrow children \cup \{r\}$
70:         for $c \in a.newC$ do
71:             $C_{work} \leftarrow C_{work} \cup \{c\}$
72:         end for
73:         take carrier $c_{ws}$ to $r$ and disembark
74:         Initialize Work Site($ws$, $a.newC$)
75:         take carrier $c_{ws}$ to $ws$ and disembark
76:     end if
77:     Do Work($c_{ws}$)        ▷ Keep working on original carrier
78: end procedure
```

If it is discovered by only a single agent a, working from work site ws, then the agent a clearly reaches s, adding c to $C_{subtree}$ on ws up to s. Consider now the case when two or more agents $a_1, a_2, \ldots a_i$ from work sites $ws_1, ws_2, \ldots ws_i$ independently discover carrier c; let w be the closest common ancestor in the work tree of the work sites $ws_1, ws_2, \ldots ws_i$. Each such agent competes to add the new information to the work tree. However, mutual exclusion access to the whiteboards ensures that only one, say a_j, will be able to proceed from w to s, and ws_j will become the only work site for carrier c. □

7.2.6 Find work

Now that we have seen work being done and new work added to the tree, it is easy to discuss how an agent a finds work. When a work site is initialized, its parent is set to the work site from which it was discovered (*null* in the homebase's case) and its children are initially *null*. As mentioned before, each work site has a set C_{work} that contains the carriers in its subtree with unexplored stops.

An agent a looking for work at a work site checks the local set C_{work}. If it is not empty a chooses a carrier $c \in C_{work}$ and walks down the tree until it reaches the work site ws servicing c, or it finds that c is no longer in C_{work}. Assume that agent a reaches ws without finding c missing from C_{work}. Then a works on c until it is either eliminated by a black hole or U_c is empty. If the agent survives and is the first agent to discover that U_c is empty, it walks up the tree from ws to s removing c from C_{work} along the way. So, it is possible for an agent descending to do work on c to find out before it reaches ws that the work on c is finished. In that case, the agent starts over trying to find work.

If agent a looking for work finds that C_{work} at the current work site is empty,

Algorithm 7.4 Find work

Agent a is looking for work in the work tree. The agent knows n_C, the number of carriers, which is needed for termination. Let ws be the current work site.

```
79:  procedure FIND WORK
         ▷ Main loop
80:      while (not on $s$) $\vee$ ($C_{work} \neq \emptyset$) $\vee$ ($|C_{subtree}| < n_C$) do      ▷ Termination conditions
             ▷ Choose carrier to work on and go there
81:          if $C_{work} \neq \emptyset$ then
82:              choose carrier $c$ from $C_{work}$
83:              while ($c \notin C_{local}$) $\wedge$ ($c \in C_{work}$) do      ▷ While not $c$'s work site and $c$ has work
84:                  take appropriate carrier to *child* in direction of $c$ and disembark
85:              end while
86:              if $c \in C_{local}$ then      ▷ On the work site servicing $c$
87:                  DO WORK($c$)
                     ▷ Carrier $c$ has no more work so remove it from the tree
88:                  if $c \in C_{work}$ then      ▷ The first agent to find no work left on $c$
89:                      while not on $s$ do
90:                          $C_{work} \leftarrow C_{work} \setminus \{c\}$      ▷ Remove $c$ from all $C_{work}$ on way to $s$
91:                          take appropriate carrier to *parent* and disembark
92:                      end while
93:                      $C_{work} \leftarrow C_{work} \setminus \{c\}$      ▷ Remove $c$ from $C_{work}$ on $s$
94:                  end if
95:              end if
             ▷ No work in subtree
96:          else
97:              take appropriate carrier to *parent* and disembark
98:          end if
99:      end while
100: end procedure
```

it moves to the work site's parent and tries again. If it reaches the root without finding work but the other termination conditions are not met (there are fewer than n_C carriers in the work tree), the agent waits (loops) until new work arrives or the termination conditions are finally met. The pseudocode for the FIND WORK procedure is in Algorithm 7.4.

Figure 7.6 shows the work tree information at some instant used to find work in our example subway graph. Note the agent from carrier c_4 won the competition

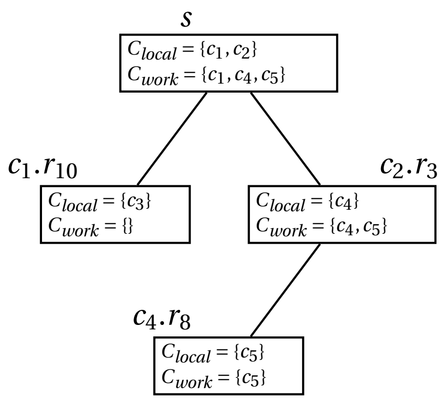

Figure 7.6: Work tree information for example from Table 7.2. Agent is finding work at some instant after Figure 7.5.

described in the last section to add carrier c_5 to the tree. Assume that an agent has just finished exploring carrier c_3 from stop r_{10} on carrier c_1's route and the agent finds no more unexplored stops. There are no other carriers in this part of the work tree, so the agent moves to the parent in the tree, s. It finds that three carriers, c_1, c_4, and c_5, still have work. It randomly chooses a carrier from C_{work} and safely traverses the work tree to that carrier's work site, where it starts to do work.

The FIND WORK procedure ensures the following property:

Lemma 7.4. *Within finite time, an agent looking for work either finds it or waits at the root.*

Proof. By Lemma 7.3, all work gets reported to the root within finite time. By construction, if an agent does not find work on the current work site, it moves to the work site's parent. Both checking for work and moving to the parent take finite time. Therefore, within finite time, an agent looking for work either finds it or reaches the root and waits there until work arrives or the termination conditions are satisfied. □

7.3 Correctness and complexity

Now that we have seen how algorithm *SubwayExploreSWB* works, we can show that it works correctly.

To do so, we need to establish some additional properties of the Algorithm:

Lemma 7.5. *Let $r_i \in R(c)$ be a black hole. At most one agent is eliminated by stopping at r_i when riding c.*

Proof. By contradiction, assume that two agents have chosen to explore a black hole stop r_i on the same carrier c. By Lemma 7.3, each carrier is worked on from a single work site, so both agents must have chosen r_i from U_c on the work site. However, since U_c is kept on the whiteboard which is accessed in mutual exclusion, it is impossible for both agents to choose the same stop for exploration. Hence, the lemma follows. □

Lemma 7.6. *There is at least one agent alive at all times before termination.*

Proof. By Lemma 7.5, we know that only one agent can die per black hole stop. Since we have $\gamma(\vec{G})+1$ agents working, where $\gamma(\vec{G})$ is the number of black hole stops in $\vec{G}$, there is always at least one more agent than can be eliminated by the black holes in the network. □

Lemma 7.7. *An agent that undertakes work completes it within finite time.*

Proof. An agent works by visiting an unexplored stop on a carrier's route. The carrier arrives within finite time and takes finite time to deliver the agent to the stop. If the agent is eliminated then its work is completed and other agents are protected from being eliminated by the same stop on the same carrier. If the agent survives then within finite time it takes the carrier back to the work site to report on the work. Hence, the lemma follows. □

Lemma 7.8. *If there is work available, some agent eventually does it.*

Proof. By contradiction, assume that there is work available but no agent does it. By construction, an agent is either trying to do work, competing to add work, or find work and by Lemmas 7.3, 7.4, and 7.7, the agent completes each activity in finite time. Furthermore, by Lemma 7.4, if there is work available an agent finds it. Therefore, the only reason that work would be available but no agent does it is if all the agents have been eliminated by a black hole. But, we know from Lemma 7.6 that there is always at least one agent alive, a contradiction. Hence, the lemma follows. □

Lemma 7.9. *All carriers are eventually added to the tree.*

Proof. By Lemma 7.8, available work is eventually done. As a result, every carrier is eventually discovered. When an agent doing work finds a new carrier, by Lemma 7.7, it is added to the tree in finite time. Therefore, all carriers are added within finite time and the lemma follows. □

We can now state the correctness of our algorithm:

Theorem 7.10. *Algorithm SubwayExploreSWB correctly and in finite time solves the* BHS *problem with* $k \geq \gamma(\vec{G}) + 1$ *agents in any subway graph* $\vec{G}$.

We now analyze the costs of our algorithm, establish lower bounds on the complexity of the problem and prove that they are tight, showing the optimality of our protocol.

Theorem 7.11. *The algorithm solves black hole search in a connected dangerous asynchronous subway graph in $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves in the worst case.*

Proof. First let us examine the DO WORK procedure. Each time an agent chooses a node for exploration, it waits up to $l(c)-1$ carrier moves for the carrier to return to the work site. The agent then takes at most $l(c)$ carrier moves to reach the stop and return to the work site. Once the stop has been explored, the agent waits at most $l(c)$ carrier moves for the carrier to return. Therefore, the cost for exploring one stop is at most $3l(c)-1$ carrier moves. Since $l(c) \leq l_R$, the cost for exploring all the stops on a route is $O(l_R^2)$ carrier moves and the cost for searching all n_C carriers is $O(n_C \cdot l_R^2)$.

Next let us examine the cost of the COMPETE TO ADD WORK procedure. Consider when an agent finds a transfer site where a previously unknown carrier c causing the resulting work to have to be added. Note that several agents, possibly all,might discover c independently and must compete to add the new work to the tree. Each of these agents tries to move towards the root, moving from its current work site to its parent, each move costing up to $l(c)-1 \leq l_R-1$ carrier moves. However, for each work site, at most one agent wanting to add the work of c is allowed to proceed to its parent work site. Hence the total number of moves up the tree incurred to add the work of c is at most one for each edge in the tree, i.e. n_C-1, for a total of at most $O(n_C \cdot l_R)$ carrier moves.

Finally, let us consider the cost of FIND WORK procedure. When finding work, an agent a first looks for information of where work can be found, moving up the work tree if needed; once the information is found, agent a moves down the tree following the specified direction. Notice that it is possible that, when looking for information, agent a finds none, ending up at the root of the work tree; if there are still some routes to be explored (a condition that a can verify), eventually some other agent will add a new carrier to be explored (i.e., new work to be performed) to the list at the root and a will follow the indication. All this movement will elicit at most $2n_C - 1$ moves up and down the tree, each costing at most l_R carrier moves. An agent looks for an indication only when it has no work; i.e., when the stops of the local carrier it was working for have all been explored; this means that this process can occur at most n_C times. In other words, the total number of carrier moves caused by finding work will be at most $O(k \cdot n_C^2 \cdot l_R)$.

Summarizing, the total number of carrier moves required by the algorithm in the worst case is at most $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$. □

We know from Chapter 6 that the lower bound on solutions to the BHS problem in the subway model is $\Omega(\gamma(\vec{G}) \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves. Since our algorithm requires only $k = \gamma(\vec{G}) + 1$ agents and uses $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, the optimality of the algorithm now follows.

Theorem 7.12. *Algorithm SubwayExploreSWB is agent-optimal and move-optimal.*

7.4 Conclusion

In this chapter, we presented a solution to the BHS problem in the subway model with site whiteboards that is both agent and move optimal. The BHS problem (Definition 2.12) is to construct a map of a subway graph that shows where carriers stop at black holes (Definition 2.9). We presented a solution to the problem using a team of agents that start co-located on the same homebase. We proved that the solution is correct and works with an optimal number of agents $k = \gamma + 1$, where γ is the faulty load or number of black hole stops. We then proved that our solution has a complexity of $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, where n_C is the number of carriers, and l_R is the length of the longest route. The complexity of our solution matches the lower bound for solutions to the BHS problem proven in Chapter 6 and is therefore optimal.

CHAPTER 8

Black Hole Search by Scattered Agents in the Subway Model with Carrier Whiteboards

In this chapter, we look at the BHS problem in the subway model with scattered agents and carrier whiteboards. The BHS problem (Definition 2.12) is to construct a map of a subway graph that shows where carriers stop at black holes (Definition 2.9). We present a solution to the BHS problem in arbitrary subway graphs of unknown topology with an arbitrary number of black holes and with asynchronous scattered agents and asynchronous carriers.

We review the subway model in Section 8.1. The model is presented in full detail in Section 2.3. We provide a solution to the BHS problem in the scattered agent, carrier whiteboard version of the subway model. The algorithm is presented in Section 8.2 and its correctness and complexity are presented in Section 8.3. We summarize the chapter in Section 8.4.

The solutions in this chapter and in Chapter 7 are optimal in that they meet the lower bound proven in Section 6.3 as summarized in Table 8.1.

Problem	**Result**	**Reference**
BHS Lower bound	$\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chapter 6, Flocchini et al. (2010b, 2012)
BHS by co-located agents with site whiteboards	$O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chapter 7, Flocchini et al. (2010b, 2012)
BHS by scattered agents with carrier whiteboards	$O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$	Chapter 8
Note that $k \geq \gamma + 1$ for our subway model solutions.		

Table 8.1: Results related to BHS by scattered agents in the subway model with carrier whiteboards.

8.1 Model

We summarize the subway model here. For a full description of the model, see Section 2.3, and for a summary of the notation used, see Table 2.2. In this chapter, we assume that the agents start scattered in the network and communicate with each other using whiteboards located on the carriers.

A set C of n_C carriers move asynchronously among a set S of n_S sites. Each carrier $c \in C$ follows a route $R(c)$ between the sites in its domain $S(c) = \{s_0, s_1, \ldots, s_{n_S(c)-1}\} \subseteq S$, where $n_S(c) = |S(c)|$. The route $R(c) = \langle r_0, r_1, \ldots, r_{l(c)-1}\rangle$ is cyclic sequence of stops that carrier visits in order. A route has a length $l(c) = |R(c)|$ and is called simple if its size of its domain is equal to the length of its route, $n_S(c) = l(c)$. A transfer site is any site that is in the domain of two or more carriers.

Each carrier's route defines a carrier graph, which is an edge-labelled directed multigraph $\vec{G}(c) = (S(c), \vec{E}(c), \lambda(c))$. The entire network defines a subway graph, which is an edge-labelled directed multigraph $\vec{G} = (R, \vec{E}, \lambda)$. Each subway graph has associated with it a transfer graph, which is an edge-labelled undirected multigraph $H(\vec{G}) = (C, E_T)$, where the nodes are the carriers and an edge connects two carriers if they share a transfer site.

Working in the network is a team A of k mobile agents that start scattered at unpredictable times. An agent moves around the network by boarding any carrier stopping at its current site and disembarking at some other site the carrier stops at on its route. Agents perform computations asynchronously and communicate with one another using whiteboards located on the carriers.

In the network are $n_B < n_S$ black hole sites that eliminate agents disembarking on them, but do not affect carriers. We define the number of stops that a carrier c makes at black hole sites as its faulty load, $\gamma(c)$ and the total of all faulty loads in the systems as $\gamma(\vec{G})$ or simply γ where no ambiguity arises.

In addition, we make the following assumptions that we proved in Section 6.2 are necessary for the EXP and BHS problems to be solvable in the subway model. Each carrier is labelled with a distinct id. The agents know the number of carriers n_C. Each transfer site is labelled with the number of carriers stopping there. Without loss of generality, since the agent knows the number of carriers and can wait for all the carriers to pass by, we assume that each transfer site is also labelled with the ids of the carriers. Finally, we assume that the standard assumptions for BHS apply: an agent's homebase is a safe site, the transfer graph is connected when the black holes are removed, there are more agents than faulty stops ($k > \gamma$), and the agents know the number of faulty stops γ.

In the carrier whiteboard version of the subway model used in this chapter, the whiteboard moves with the carrier, so being able to board the carrier at the same point in its route is not sufficient to solve EXP or BHS. Instead, once boarded, an agent must be able to distinguish the current stop from all other stops on the carrier's route. We conjecture that this condition is necessary to solve the BHS problem in the carrier whiteboard version of the subway model.

As a result, we assume that there is a function $r_{curr}(c)$ that returns a distinct id for the current stop when the agent is on carrier c. Without loss of generality, since the agent can record every stop during the carrier's traversal of the route, we assume that there is a second function $R(c)$ that returns the set of all stops on the route. This additional assumption is not significant in terms of complexity. The same function can be used to determine the length of the carrier's route.

8.2 Algorithm

In this section we present the proposed algorithm *SubwayExploreCWB*; as we show later, our algorithm works correctly with any number of agents $k \geq \gamma + 1$ and is cost optimal. We first describe how the algorithm works in general, followed by a more detailed description.

8.2.1 Overview

Algorithm *SubwayExploreCWB* works as follows. We describe it first from the perspective of a single agent and then discuss how the agents coordinate their activities.

An agent starts on a safe site in the subway graph. It takes the first carrier stopping there, which becomes its *home carrier*. Each carrier's whiteboard is used to keep track of the exploration of that carrier's stops. The agent works by choosing a stop, visiting it, and, if possible, returning to report what was found there, in our adaptation of the cautious walk technique to the subway model. All the information about the carriers, their stops, and the transfer sites that connect them are kept in a map carried by the agent. To find work, the agent computes a spanning tree of the transfer graph of the subgraph of the network covered by its map. The tree is rooted in the agent's home carrier and spans the carriers, which are the nodes of the transfer graph, that are known to the agent. The agent does a traversal of this tree and works on any carrier where there are unexplored stops. The agent terminates when the number of nodes (carriers) in the transfer graph of its map equals the number of carriers, n_C, and it has completed a traversal without doing any work.

Since some agents are eliminated by black holes, the agents must coordinate their efforts both locally on carriers being searched and globally for all carriers. Locally, coordinating the work of searching an individual carrier is accomplished by keeping sets of unexplored, dangerous (being explored), and explored stops on the carrier. Globally, the agents coordinate their finding of new work by merging their maps with the map located on each carrier's whiteboard every time they visit a carrier. Any new carriers added to their map are included in the tree computed for the next traversal looking for work.

Finally, it is possible for an agent not to find any work and for the termination conditions not to be met. This situation arises when the transfer sites out of the subgraph described by the agent's map are all being explored (marked dangerous). In this case, the agent waits on its home carrier for new carriers to be added to the home carrier's map. When an exploring agent finds a transfer site, it boards each of the carriers stopping there and takes a copy of its map. When it returns to the carrier it is working on, it checks its map against the carrier's map. If there are new carriers in its map, the agent computes a new tree on the map and traverses that tree merging its map with the carriers' maps in the tree. If an agent is waiting on one of those carriers, it stops waiting when it sees the updated map.

The algorithm terminates with each carrier having a map of the entire subway graph when the last agent finishes it last traversal.

8.2.2 Initialization

An agent a waking for the first time on its homebase s initializes in its memory a map, $a.M$, which initially contains information about s. The agent then takes the first carrier c_s stopping at s, which becomes its *home carrier*. The agent checks to see if c_s's whiteboard is blank and, if so, initializes it using the INITIALIZE WORK INFO procedure. The agent then enters the FIND WORK procedure.

The INITIALIZE WORK INFO procedure is used to setup a carrier c's whiteboard with all the information needed to both coordinate the searching of c's stops and the finding of work more generally. Let the function $R(c)$ return the set of stops visited by the carrier c and the function $r_{curr}(c)$ return c's current stop from $R(c)$. To coordinate the exploration work, the agent creates three sets on c's whiteboard: the set of unexplored stops, U, which initially all of c's stops, $R(c)$, minus the current stop, $r_{curr}(c)$; the set of dangerous stops (or stops being explored), D, which initially is set to $\emptyset$; and the set of explored stops, E, which initially contains only the current stop $r_{curr}(c)$. To coordinate the finding of work, the agent initializes the map M on c's whiteboard with a copy of its own map.

The pseudocode for the start of algorithm *SubwayExploreCWB* is shown in Algorithm 8.1 and the pseudocode for procedure INITIALIZE WORK INFO is shown in Algorithm 8.2.

Algorithm 8.1 *SubwayExploreCWB*

```
    Agent $a$ awakes on a safe site $s$.

 1: $a.M \leftarrow$ info about $s$                          ▷ Agent's map of network
 2: board first carrier $c_s$ arriving at $s$                 ▷ Board home carrier
 3: if whiteboard is blank then
 4:     INITIALIZE WORK INFO ($c_s$)
 5: end if
 6: FIND WORK
```

Algorithm 8.2 Initialize work information

```
    Agent $a$ is initializing the whiteboard of carrier $c$ with information needed to do work
    on carrier.
    Functions used:
    ▷ $R(c)$ returns the set of stops for current carrier $c$
    ▷ $r_{curr}(c)$ returns the current stop on current carrier $c$

 7: procedure INITIALIZE WORK INFO (carrier $c$)
 8:     $U \leftarrow R(c) \setminus r_{curr}(c)$             ▷ Set of $c$'s unexplored stops
 9:     $D \leftarrow \emptyset$                              ▷ Set of $c$'s stops being explored
10:     $E \leftarrow r_{curr}(c)$                            ▷ Set of $c$'s explored stops
11:     $M \leftarrow a.M$                                    ▷ Carrier's map of network
12: end procedure
```

8.2.3 Find work

An agent a follows the FIND WORK procedure until the termination conditions are met. The termination conditions are that the agent has traversed the subway graph without doing any work or finding any new carriers and its map contains n_C nodes (carriers).

Each round, the agent does the following. It sets the work flag, *a.worked*, to *false*. It synchronizes its map with its home carrier's map by merging the two maps together and saving the result to the carrier's map and then taking a copy. We use the operator $\oplus$ to denote the merging of two maps. The agent then computes a spanning tree $a.T$ on the transfer graph of the subgraph of the subway graph contained in its map. We use the tree to allow the agent to efficiently look for work and to check if any new carriers are found during the traversal. The map related functions include the following: $H(M)$, which returns the transfer graph of a map M; $T(H)$, which returns the spanning tree of a transfer graph H; and $C(M)$ or $C(T)$, which return the set of carrier ids for a map M or tree T, respectively. The calculation of the tree is therefore expressed as $A.T = T(H(a.M))$, with c_s being the root of the resulting tree.

The agent does a preorder traversal of tree $a.T$ starting at the tree's root c_s. On each carrier c in the traversal, a first collects information about c for its map. The agent then checks to see if the set of unexplored stops is empty $U \neq \emptyset$. If it is not empty, the agent sets its work flag, *a.worked*, to *true* and then follows the Do Work procedure on c. The agent synchronizes its map with the carrier's and then continues to the next carrier in its traversal of $a.T$.

It is possible that a traversed the entire tree and returned to its home carrier, c_s, without doing any work, $\neg$*a.worked*, or adding any new carriers to its map, $C(a.M) = C(a.T)$. In that case, the agent waits on c_s until new carriers are added to c_s's map, $|C(a.M)| < |C(M)|$.

The pseudocode for Find Work is shown in Algorithm 8.3.

8.2.4 Do work

An agent a follows the Do Work procedure while working on carrier c. The agent continues until the set of unexplored stops is empty, $U = \emptyset$, or it is eliminated by a black hole.

Each round, a starts by synchronizing its map with the carrier's map. It then chooses a stop r from the set of unexplored stops, U. The agent moves r from U to

Algorithm 8.3 Find work

Agent a is looking for work starting from its home carrier c_s. The agent knows n_C, the number of carriers, which is needed for termination. The operator $\oplus$ denotes the merger of two maps.
Functions used:
▷ $H(M)$ returns the transfer graph of a map M
▷ $T(H)$ returns a spanning tree of a transfer graph H
▷ $C(M)$ or $C(T)$ returns the set of carrier ids in a map M or tree T

```
13: procedure FIND WORK
      ▷ Main loop
14:   repeat
15:     $a.worked \leftarrow false$                                  ▷ Work flag
16:     $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$;              ▷ Synchronize maps $M$ and $a.M$
17:     $a.T \leftarrow T(H(a.M))$     ▷ Compute spanning tree of transfer graph of agent's map
18:     while depth-first traversal of $a.T$ do                    ▷ Preorder traversal
          ▷ On each carrier $c$ in the traversal
19:       $a.M \leftarrow a.M \oplus$ info about carrier $c$
20:       if $U \neq \emptyset$ then
21:         $a.worked \leftarrow true$
22:         DO WORK ($c$)
23:       end if
24:       $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$;            ▷ Synchronize maps $M$ and $a.M$
25:       take transfer link to next carrier in traversal of $a.T$
26:     end while
        ▷ Back on home carrier $c_s$
27:     if $\neg a.worked \wedge C(a.M) = C(a.T) \wedge |C(a.M)| < n_C$ then    ▷ No new work
28:       wait until $|C(a.M)| < |C(M)|$       ▷ Awake periodically to check $c_s$'s map $M$
29:     end if
30:   until $\neg a.worked \wedge C(a.M) = C(a.T) \wedge |C(a.M)| = n_C$
31: end procedure
```

the set of dangerous stops, D, which are the stops currently being explored. The agent then disembarks when r is the current stop, $r_{curr}(c) = r$. If it is not eliminated by a black hole, a collects information on r in its map.

If r is a transfer site, a collects information on the carriers stopping there. For each carrier $c_i \notin C(a.M)$ stopping at r, the agent boards the carrier. If its whiteboard is blank, it initializes it using the INITIALIZE WORK INFO procedure. If the whiteboard is not blank, the agent moves the stop it arrived from, r, from the set of unexplored stops, U, to the set of explored stops, E. The agent finishes by synchronizing its map before disembarking at r to wait for the next carrier.

The agent boards c from r. It moves r from D to the set of explored stops, E and then merges its map with c's and takes a copy. If no new carriers were discovered, as would be the case if r were not a transfer site, a continues to the next round of work on c.

If agent a discovered new carriers during its exploration of r, $|C(a.M)| > |C(M)|$, then it adds the new carriers to the maps of the carriers in its map. It synchronizes its map with c's map, computes a new tree, $a.T_{notify}$, based on its map and does a traversal of the new tree synchronizing its map on each carrier before returning to c. The agent then continues to the next round of work on c.

The pseudocode for DO WORK is shown in Algorithm 8.4.

8.3 Correctness and complexity

We prove a series of lemmas that leads to the theorem that proves the correctness of our algorithm. We start by proving that the subway-model-with-carrier-whiteboards version of the cautious walk technique (Section 2.1.2) works and only

Algorithm 8.4 Do work

Agent a is working on carrier c.

```
32: procedure Do Work (carrier $c$)
33:     while $U \neq \emptyset$ do
34:         $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$;                ▷ Synchronize maps $M$ and $a.M$
35:         choose a stop $r \in U$
36:         $U \leftarrow U \setminus \{r\}$                ▷ Remove $r$ from the set of unexplored stops
37:         $D \leftarrow D \cup \{r\}$                ▷ Add $r$ to the set of stops being explored
38:         disembark when $r_{curr}(c) = r$
            ▷ If not eliminated by black hole
39:         $a.M \leftarrow a.M \oplus$ info about $r$
            ▷ If $r$ is transfer site check the carriers passing by for information
40:         for each carrier $c_i \notin C(a.M)$ stopping at $r$ do                ▷ If $r$ is a transfer site
41:             board $c_i$
42:             if whiteboard is blank then
43:                 Initialize Work Info ($c_i$)
44:             else
45:                 $U \leftarrow U \setminus \{r\}$                ▷ Remove $r$ from set of unexplored stops
46:                 $E \leftarrow E \cup \{r\}$                ▷ Add $r$ to set of explored stops.
47:             end if
48:             $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$;                ▷ Synchronize maps $M$ and $a.M$
49:             disembark when $r_{curr}(c_i) = r$
50:         end for
51:         board $c$
52:         $D \leftarrow D \setminus \{r\}$                ▷ Remove $r$ from the set of stops being explored
53:         $E \leftarrow E \cup \{r\}$                ▷ Add $r$ to the set of explored stops
            ▷ If previously unknown carriers found distribute new map
54:         if $|C(a.M)| > |C(M)|$ then
55:             $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$;                ▷ Synchronize maps $M$ and $a.M$
56:             $a.T_{notify} = T(H(a.M))$                ▷ Compute spanning tree for notification
57:             while depth-first traversal of $a.T_{notify}$ do                ▷ Preorder traversal
                    ▷ On each carrier in the traversal
58:                 $a.M \leftarrow a.M \oplus$ info about carrier
59:                 $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$;                ▷ Synchronize maps $M$ and $a.M$
60:                 take transfer link to next carrier in traversal of $a.T_{notify}$
61:             end while
                ▷ Back on carrier $c$
62:         end if
63:     end while
64: end procedure
```

one agent is eliminated per black hole stop.

Lemma 8.1. *Let $r \in R(c)$ be a black hole. At most one agent is eliminated by stopping at r when riding carrier $c \in C$.*

Proof. By contradiction, assume that two agents have chosen to explore a black hole stop r on the same carrier c. Since the exploration of a carrier c is coordinated from that carrier, both agents must have chosen r from the set of unexplored stops U on carrier c. However, since U is kept on the carrier c's whiteboard, which is accessed in mutual exclusion, it is impossible for both agents to choose the same stop for exploration. Hence, the lemma follows. □

Since only one agent is eliminated per stop, it is now easy to show that one agent must always be "alive" before termination.

Lemma 8.2. *There is at least one agent alive at all times.*

Proof. By Lemma 8.1, we know that only one agent can die per black hole stop. Since we have $\gamma(\vec{G})+1$ agents working, where $\gamma(\vec{G})$ is the number of black hole stops in the subway graph $\vec{G}$, there is always at least one more agent than can be eliminated by the black holes in the network. □

We now prove that an agent doing work completes in finite time.

Lemma 8.3. *Within finite time, an agent that undertakes work completes it.*

Proof. By construction, an agent does work on a carrier c by choosing a stop r on the carrier's route, doing a cautious walk, and, if possible, reporting what it found either locally or within the subgraph described by its map. Note that the movement of carriers and the computations of agents are asynchronous but finite, and that boarding and disembarking are considered to be instantaneous.

The agent, on carrier c, arrives at r within finite time and disembarks. There are four possible cases for r: (i) r is a black hole, (ii) r is a safe non-transfer site, (iii) r is a transfer site but no new carriers are found, and (iv) r is a transfer site and new carriers are found. We describe the agent's actions in each of these situations.

(i) r is a black hole. The agent is eliminated when it disembarks, completing its work since the stop leading to the black hole is marked dangerous and no other agent can be eliminated there.

(ii) r is a safe non-transfer site. The agent must wait a finite time for the carrier to return before boarding again. The agent reports its results within finite time, completing its work.

(iii) r is a transfer site but no new carriers are found. The agent boards each of the carriers stopping there. For each carrier $c_i \notin C(a.M)$, the agent must wait for c_i to arrive, board it, possibly initialize its whiteboard, synchronize the agent's map with the carrier's, wait for c_i to return to r, and disembark. Each of these actions takes a finite amount of time and since there are a finite number of carriers, the whole process is finite. The agent then waits for c to return and boards it. The agent checks its map, which has been synchronized with the maps on all the unknown carriers stopping at r, against the map on c and finds its map contains no new carriers. The agent reports its results in finite time, completing its work.

(iv) r is a transfer site and new carriers are found. The agent follows all the same steps as the previous case, followed by an additional step. On its return to c, the agent checks its map, which has been synchronized with the maps on all the carriers stopping at r, against the map on c and finds its map contains at least one new carrier. It computes and traverses a spanning tree of the transfer graph of the subgraph described by its own map in order to spread the information about the new carrier or carriers. We know the spanning tree is safe because it can only contain transfer sites as links if the transfer sites themselves are safe. We also know that all carrier movements and agent calculations are asynchronous but take a finite amount of time. Therefore, any traversal is safe and takes a finite time to complete. Hence, the agent finishes its traversal within finite time, completing its work.

These cases cover all possibilities and all work is completed within finite time. Hence, the lemma follows. □

If an agent is looking for work, we now prove that within finite time, it finds it, waits, or terminates the algorithm.

Lemma 8.4. *Within finite time, an agent looking for work either finds it, waits on its home carrier, or terminates the algorithm.*

Proof. By construction, an agent finds work by repeatedly computing and traversing a spanning tree of the transfer graph of the subgraph described by its own map. We know the spanning tree is safe because it can only contain transfer sites as links if the transfer sites themselves are safe. We also know that all carrier movements and agent calculations take a finite amount of time. Therefore, any traversal or part thereof is safe and takes a finite time to complete.

There are four possible outcomes for any single traversal to find work by the agent: (i) it finds work, (ii) it waits at its home carrier c_s, (iii) it terminates the algorithm or (iv) it detects a new carrier. Let $a.M_i^B$ be the agent's map at the beginning of the of the ith traversal, $a.M_i^E$ be the agent's map at the end of the ith traversal, and $a.T_i$ be the spanning tree computed by the agent for the ith traversal.

(i) The agent finds work. By the algorithm, this case occurs when the agent finds a carrier with a non-empty set of unexplored stops, $U \neq \emptyset$, during its traversal.

(ii) The agent waits at its home carrier c_s. By the algorithm, this case occurs when the agent has completed a full traversal and returned to its home carrier. By construction, the agent synchronizes its map with the map of every carrier in $a.T$ during the traversal. In this case, the agent finds that the number of carriers in its map after traversal equals the number of carriers in the tree it computed for the traversal, $|C(a.M_i^E)| = |C(a.T_i)|$, so it waits.

(iii) The agent terminates the algorithm. By the algorithm, this case occurs when the agent has completed a full traversal and returned to its home carrier. By construction, the agent synchronizes its map with the map of every carrier in $a.T$ during the traversal. In this case, the agent finds that the number of carriers in its map after traversal equals the number of carriers in the tree it computed for the traversal and that number equals the number of known carriers $|C(a.M_i^E)| = |C(a.T_i)| = n_C$.

(iv) The agent detects a new carrier. By the algorithm, this case occurs when the agent has completed a full traversal and returned to its home carrier. By construction, the agent synchronizes its map with the map of every carrier in $a.T$ during the traversal. Note that since $C(a.M_i^B) = C(a.T_i)$ at the start of every traversal, the detection of a new carrier at the end requires that $|C(a.M_i^E)| > |C(a.T_i)|$. In other words, the agent must detect at least one carrier outside of $a.T$ for this case to apply. If a new carrier is detected, by the algorithm, the agent calculates $a.T_{i+1}$ for the next traversal with the same three possible outcomes. However, since there are only n_C carriers and one carrier must be found each traversal for this case, the agent must find work, wait, or terminate after at most n_C rounds.

These cases cover all possibilities and always lead to finding work, waiting, or termination. Hence, the lemma follows. □

While working and terminating are useful, waiting would seem to be something that agent could do indefinitely. We show that within finite time, a waiting agent learns about a new carrier where it can go to search for more work.

Lemma 8.5. *Within finite time, a waiting agent learns of a new carrier.*

Proof. By Lemma 8.4, we know that a waiting agent found neither work nor new carriers on its last traversal. This situation can only arise if all the links leading out of the subgraph described by the agent's map are either explored or being explored. If the waiting agent's map contained all the carriers, it would have terminated, so there must be carriers that do not appear in the agent's map.

There must be at least one stop being explored that is a safe transfer site that connects to a carrier that is not in the map. By contradiction, assume that no such transfer site exists. All the stops being explored must be black holes, safe non-transfer sites, or transfer sites that do not connect to new carriers. However, that would mean that the transfer graph of the network is disconnected, a contradiction.

By Lemma 8.3, we know that a working agent exploring a transfer site that connects to new carriers finishes its work within finite time. Its work includes synchronizing its map with the map of every carrier in the subgraph it's map describes. Since the working agent's map has been synchronized with the map of the carrier from which it was working, and that carrier's map includes c_s because it was synchronized with the map of the waiting agent on its last traversal, the working agent within finite time must update c_s with a map that contains new carriers. Hence, the lemma follows. □

In fact, if there is any work available in the system, we show that some agent must eventually find it and do it.

Lemma 8.6. *If there is work available, some agent eventually does it.*

Proof. By contradiction, assume that there is an unexplored stop r on carrier c that is never explored. Let C_E be the set of visited carriers, the carriers in some agent's map, and C_U be the set of unvisited carriers, the carriers in no agent's map. We look at the cases where $c \in C_E$ and $c \in C_U$.

The $c \in C_E$ case. In this case, r never being explored means that it is still in the set of unexplored stops, $r \in U$, on c and all the agents have either terminated or been eliminated by black holes. By construction, an agent terminates when it has completed a traversal that includes all the carriers and has not found any work, meaning that $U = \emptyset$ on all the carriers in the traversal. Since $r \in U$ on c, none of the agents could have terminated, so they must all have been eliminated by black holes. However, by Lemma 8.2, we know that there is always one agent alive, a contradiction. Hence, the lemma follows.

The $c \in C_U$ case. In this case, r never being explored means that c was never added to any agent's map. However, by Lemmas 8.3, 8.4, and 8.5, we know that work finishes within finite time, an agent looking for work finds it or waits within finite time, and a waiting agents stops waiting within finite time. We know that the transfer graph of the subway graph, $H(\vec{G})$ is connected. Therefore, there must be a path between $c \in C_U$ and some carrier $c' \in C_E$. But r is never explored and c is never added to any agent's map. If the path exists then all the agents have been eliminated by black holes. However, by Lemma 8.2, we know that there is always one agent alive, a contradiction. Hence, the lemma follows. If the path does not exist then $H(\vec{G})$ is not connected, a contradiction. Hence, the lemma follows. □

As a consequence, we get the following corollary:

Corollary 8.7. *All carriers are eventually added to each carrier's map.*

We can now state the correctness of our algorithm based on the preceding lemmas.

Theorem 8.8. *Algorithm SubwayExploreCWB correctly and in finite time solves the mapping problem in any subway graph $\vec{G}$.*

We now prove an upper bound on the complexity of our algorithm.

Theorem 8.9. *Algorithm SubwayExploreCWB solves black hole search in a connected dangerous asynchronous subway graph in $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves in the worst case.*

Proof. We look at the cost of finding work and doing work separately and then combine the costs together.

Let us start by looking at cost of the FIND WORK procedure. Each round of trying to find work, an agent computes a spanning tree on the transfer graph of the subgraph described by its own map and traverses it. In the transfer graph, the nodes are carriers and the links are transfer sites. Doing a full traversal of the tree means the agent transfers up to $2(n_C - 1)$ times between carriers or $O(n_C)$ transfers. The movement of an agent from one carrier to another across a transfer site costs up to $l(c)-1$ carrier moves for the agent to arrive at the transfer site and up to $l(c)$ carrier moves waiting for the next carrier to arrive for a cost of $2l(c)-1$ carrier moves or $O(l_R)$ carriers moves since $l(c) \leq l_R$. Combining the two costs, we get the cost for a single traversal of the tree as $O(n_C \cdot l_R)$ carrier moves.

The agent repeats the traversal each time it finds work or detects a new carrier. Since it completes all work it finds in a single traversal, it does at most one extra traversal to find work afterwards for a maximum of 2 traversals for each carrier on which it finds work. If no new carriers are found, it waits. If it detects new carriers—carriers it has never seen before—it does another traversal that includes the new carriers. It can detect at most $n_C - 1$ or $O(n_C)$ new carriers, so the total cost for an agent to find work is $2 \cdot O(n_C) \cdot O(n_C \cdot l_R) = O(n_C^2 \cdot l_R)$ carriers moves.

The cost of finding work is the same for all k agents, so the total cost of the Find Work procedure is $O(k \cdot n_C^2 \cdot l_R)$ carrier moves.

Let us look at the cost of the Do Work procedure. An agent does work by choosing a stop, doing a cautious walk, and, if possible, reporting what it found either locally or within the subgraph described by its map.

We look first at the cautious walk. The agent must wait up to $l(c)-1$ carrier moves for the carrier to reach the stop and up to $l(c)$ carrier moves for the carrier to return for a total of $2l(c)-1$ or $O(l_R)$ carriers moves. The cost for doing cautious walk on all $O(l_R)$ stops on a route is therefore $O(l_R) \cdot O(l_R) = O(l_R^2)$ carrier moves. The total cost for searching all n_C carriers is $O(n_C \cdot l_R^2)$ carriers moves.

We look next at the cost of exploring a transfer site. An agent explores a transfer site by boarding each carrier arriving there that is not in its map. The agent can wait up to $l(c)$ carrier moves for a carrier to arrive and $l(c)$ carrier moves for it to return to the stop being explored for a total of $2l(c)$ or $O(l_R)$ carrier moves.

The agent boards a maximum of n_C carriers during the algorithm. The cost for an agent to explore all carriers from transfer sites is $O(n_C \cdot l_R)$ carrier moves. The total cost for all k agents is $O(k \cdot n_C \cdot l_R)$ carriers moves.

We look finally at the cost of notification for finding a new carrier. Each time a new carrier is found, the agent traverses a spanning tree computed on the transfer graph of the subgraph described by its map. We know that each traversal of the tree costs $O(n_C \cdot l_R)$ carrier moves. It is possible for an adversary to force each of the k agents to perform a traversal for each of the $n_C - 1$ carriers other than its home carrier. The cost for notification then is $k \cdot O(n_C \cdot l_R) \cdot O(n_C) = O(k \cdot n_C^2 \cdot l_R)$ carrier moves.

The total cost of the Do Work procedure is $O(n_C \cdot l_R^2)$ carriers moves for cautious walk, plus $O(k \cdot n_C \cdot l_R)$ carriers moves for exploring transfer sites, plus $O(k \cdot n_C^2 \cdot l_R)$ carrier moves for notification or $O(n_C \cdot l_R^2 + k \cdot n_C^2 \cdot l_R)$ carriers moves.

The cost for the whole algorithm is $O(k \cdot n_C^2 \cdot l_R) + O(n_C \cdot l_R^2 + k \cdot n_C^2 \cdot l_R)$ carrier moves. Simplifying, we get a final cost of $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carriers moves. □

We know from Chapter 6 that the lower bound on solutions to the Bhs problem in the subway model is $\Omega(\gamma(\vec{G}) \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves. Since our algorithm requires only $k = \gamma(\vec{G}) + 1$ agents and uses $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, the optimality of the algorithm now follows.

Theorem 8.10. *Algorithm SubwayExploreCWB is agent-optimal and move-optimal.*

8.4 Conclusion

In this chapter, we presented a solution to the Bhs problem in the subway model with carrier whiteboards that is both agent and move optimal. The Bhs problem

(Definition 2.12) is to construct a map of a subway graph that shows where carriers stop at black holes (Definition 2.9). We presented a solution to the problem using a team of agents scattered in the network. We proved that the solution is correct and works with an optimal number of agents $k = \gamma + 1$, where γ is the faulty load or number of black hole stops. We then proved that our solution has a complexity of $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, where n_C is the number of carriers, and l_R is the length of the longest route. The complexity of our solution matches the lower bound for solutions to BHS problem proven in Chapter 6 and is therefore optimal.

CHAPTER 9

Conclusion

In this thesis, we were interested in looking at distributed algorithms for the mobile agent model that work in dynamic networks with dangerous elements. In particular, we were interested in algorithms that mapped or at least indicated the dangerous elements in these networks in order to allow follow-on protocols to work in the network as if it were safe. We looked at black hole search and its related problem of dangerous graph exploration in two models: the network model based on a simple, connected graph, and the subway model based on an urban subway system.

We presented two solutions in the network model to the dangerous graph exploration (DGE) problem—the exploration of a network with black holes and black links. The first solution produced a map of the dangerous elements, worked with agents starting scattered in the network, and had a worst case complexity of $O(nm)$ agent moves, where n is the number of nodes and m is the number of links. The complexity of this solution was proven optimal in (Miklik, 2010). The second solution worked in the same model, but allowed an adversary to delete almost any link in the network during the algorithm's execution, and, as a result, produced a local marking of the dangerous elements rather than a map. It had a worst case complexity of $O(nm^2)$ agent moves.

We introduced the subway model and established its basic properties. We established a lower bound of $\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves on any solution to the black hole search problem, where γ is the number of carrier stops at black hole sites, n_C is the number of carriers, and l_R is the length of the longest route. We presented two solutions in the subway model to the black hole search (BHS) problem—the mapping of a subway network with black holes. Both worked with an optimal number of agents, $k = \gamma + 1$. The first solution worked with agents starting co-located in the network and communicating using whiteboards located on the sites. It had a complexity of $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, which is optimal. The second solution worked with agents starting scattered in the network and communicating using whiteboards located on the carriers. It had a complexity of $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, which is optimal.

Clearly, the problem of computing in dangerous and dynamic environments is much larger than the problems looked at in this thesis. We have barely scratched the surface. One of the limitations hanging over the head of any approach to the problem in an asynchronous network—the type of network that most closely approximates how computer networks such as the Internet work—is the difficulty of computing in the presence of faults. Faults, in this case, cover both the undetectable crash faults that are represented by black holes and black links and the changes in topology that are the very nature of dynamic networks. Fischer, Lynch, and Paterson (1985) found that it is impossible, in the absence of additional assumptions, to solve the consensus problem—all computational entities agreeing on a particular value—in the presence of even one fault. We can see the practical application of this impossibility result in the assumptions we make in our own model. In order for a solution to the dangerous graph exploration with link deletions problem to be solvable, we must put restrictions on which edges can be deleted. If the adversary could disconnect the safe portion of the network or eliminate agents, no solution could be made to work. The same is true of the movements of the carriers in the subway network. The black hole search problem is solvable if the carrier repeats a cyclical route. If a carrier did one loop and stopped, no solution would be possible because there is simply not enough opportunity to move in the network. If a carrier took a different route through the sites every time, no deterministic solution would be possible because of the randomness of the environment. We take advantage of the predictable properties in our models to allow us to find solutions to our problems. So, even though our models are not "dynamic" in a random changing sense, they give us an idea of the cost of computing in dynamic networks.

We can most easily see the cost of limited dynamics in the difference between the two solutions to the DGE problem in the network model. The models used for the two solution differ only in the ability of the adversary to delete links in the second version of the model and problem (the dangerous graph exploration with link deletions (DGE-LD) problem). We know that these deletions must cost something and that it should not be possible for a solution to the DGE-LD problem to match the $O(nm)$ agent move optimal DGE solution. Although we have no proof that our $O(nm^2)$ agent move DGE-LD solution is optimal, we can see that the cost difference between the two solutions is at most a factor of m, the number of edges, which is related to both the number of deletions, faults, and agents. It may be possible to do better than this. We could make some immediate gains by taking into account the number of accessible faults that it is possible to have in the network, in which case, we may find that our solution is indeed optimal.

The subway model is more dynamic than the network model with deletions; nevertheless, it still relies on the cyclical, repetitive routes of the carriers. Both our solutions are optimal in terms of carrier moves and their models differ in the location of the whiteboards used for inter-agent communication. However, the two models also differ in the starting locations of the agents with the site whiteboard solution using co-located agents and the carrier whiteboard solution allowing the agents to start scattered in the network. It is unclear why there is this apparent asymmetry between the two models; why carrier whiteboards appear to be slightly more powerful than site whiteboards. Symmetry is still possible. It may be possible to find a scattered solution to the site whiteboard version of the problem, or, just as likely, to find an impossibility result that proves such a solution is impossible and suggests why the two models differ in power.

Finally, it would be interesting to compare the network model solutions to the subway model solutions. Unfortunately, no basis for comparison currently exists. The complexity of the network model solutions is measured in agent moves, while the complexity of the subway model solutions is measured in carrier moves, a combination of agent moves and agent waits. There is obviously a cost to opportunistic movement in the subway model, but how this cost compares to the simpler network model is unclear. Further work is required on the complexity measure for opportunistic networks such as those described by the subway model.

Although we can't compare our subway model results to our network model results, we can compare the subway model results to the results for PV graphs presented in (Flocchini et al., 2009b). Remember that in PV graphs the agents ride the synchronously moving carriers and switch between carriers when they meet at sites. In the heterogenous case, where the routes may have different lengths, the authors give a solution with $O(n_C \cdot l_R^2)$ carrier move complexity in PV graphs. Despite the subway model requiring the agents to disembark on sites rather than just pass them by as they do in the PV graph model, the complexity is remarkably similar with both our solutions using an optimal $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves. The subway model solutions have similar complexity to the PV graph heterogenous solution when the subway graph with few faults is dominated by a small number of carriers with long routes. The similarity comes from the cost of an agent exploring a PV graph having to wait up to $O(l_R^2)$ carrier moves each time it wants to switch between two carriers with similar but unequal route lengths. The same cost for subway graphs is caused by an agent having to wait for the carrier to complete a full traversal before it can board. The additional $O(k \cdot n_C^2 \cdot l_R)$ term for subway graphs, which becomes significant when the graph is dominated by many carriers with short routes, is caused by forcing all the agents to move back and forth over the whole length of the network in order to advance their exploration; a cost that cannot occur in PV graphs since the carriers and agents move synchronously and cannot be "blocked" by an adversary. Still, the two results are remarkably similar and it would be an interesting exercise to combine the two by looking at synchronous subway graphs where, in addition to being able to board and disembark from carriers, the agents are allowed to transfer from carrier to carrier directly.

9.1 Future work

Where to next? The work of this thesis suggests some open problems that we have already mentioned and which we list below. Beyond these specific problems is the general problem of computing in dangerous and dynamic networks and the questions of how dangerous? and how dynamic?, which we briefly discuss below.

Problems arising directly from this thesis include

- Is our solution to the DGE-LD problem optimal? What is the true cost of link deletions in comparison with the $O(nm)$ agent move lower bound on solutions to the DGE problem?

- Are site whiteboards really less powerful than carrier whiteboards in the subway model?

- Carrier moves is a non-intuitive metric of the cost of opportunistic movement since it includes the "intent" of all agents waiting for a carrier. Is there a better cost measure? Does splitting carrier moves into agent moves and agent waits make for a more intuitive measurement?

- How does PV graph exploration compare with black hole search in the synchronous subway model when the agents are allowed to transfer from carrier to carrier directly?

Beyond these thesis-specific problems is the general problem of computing in dangerous and dynamic networks. As we have seen in the chapter on related work, the dangerous network problem has been well studied and the community is already well on the way to determining how dangerous is too dangerous to allow for a solution. The algorithmic approach to dynamic networks, on the other hand, is a nascent area. Finding deterministic algorithmic approaches for dynamic networks is especially difficult. The line between too dynamic and just dynamic enough has yet to be mapped.

In a sense, the work of this thesis comes at dynamicism from both ends with the network model being mostly static and the subway model being mostly dynamic. Having added link deletions to the network model, we can now think of adding link insertions. To achieve a truly dynamic network, we can also look at adding node insertions and deletions. However, dealing algorithmically with these changes to the model is not a trivial exercise.

The subway model, although fully dynamic, still relies on the periodicity of the carriers and the infinite repetition of their routes. Casteigts, Flocchini, Quattrociocchi, and Santoro (2011) detail a classification of the dynamics of a network. The subway model, based on (Flocchini et al., 2010b), is classified as a Class 8 network, which means that the network has a “periodicity of edges”. Class 8 is a subclass of Class 7, networks with a “time-bounded recurrence of edges”, which is a subclass of Class 6, networks with a “recurrence of edges”, and so on until we reach Class 1 where there is at least one node that is reachable by all other nodes at least once in the lifetime of the network. A carrier that completes one loop and stops, like the one we discussed in the last section, is an example of a Class 1 network. As we also mentioned, not many problems are solvable by mobile agent, or even message passing algorithms, with so little communication. It would be interesting to move our subway model solutions up the ladder of dynamicism laid out in (Flocchini et al., 2010b). In fact, their class structure is a tree and it may be interesting to look at other branches as well.

In the most general sense, performing deterministic computations, such as black hole search, in dynamic networks is entirely dependent on those networks not being too dynamic. There may be properties such as the periodic and infinite repetition of links that allow for predictable behaviour. There is a chance that real world dynamic networks, like mobile ad hoc networks, no matter how random they appear, have stable properties that can be used to support deterministic algorithms. Perhaps the most interesting future work, for which this thesis is but a tiny step forward, is finding those stable properties and mapping the line between too dynamic and just dynamic enough.

Bibliography

Susanne Albers and Monika R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000. doi:10.1137/S009753979732428X.

Christoph Ambühl, Leszek Gąsieniec, Andrzej Pelc, Tomasz Radzik, and Xiaohui Zhang. Tree exploration with logarithmic memory. *ACM Transactions on Algorithms*, 7(2):17:1–17:21, March 2011. ISSN 1549-6325. doi:10.1145/1921659.1921663.

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

Chen Avin, Michal Koucky, and Zvi Lotker. How to Explore a Fast-Changing World (Cover Time of a Simple Random Walk on Evolving Graphs). In *ICALP 2008: Proceedings of the 35th International Colloquium Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2008.

Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152(2):155–172, August 1999. doi:10.1006/inco.1999.2795.

Balasingham Balamohan, Paola Flocchini, Ali Miri, and Nicola Santoro. Time optimal algorithms for black hole search in rings. In *COCOA 2010: Proceedings of the 4th International Conference on Combinatorial Optimization and Applications*, volume 6509 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2010. doi:10.1007/978-3-642-17461-2_5.

Balasingham Balamohan, Paola Flocchini, Ali Miri, and Nicola Santoro. Improving the optimal bounds for black hole search in rings. In *SIROCCO 2011: Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity*, volume 6796 of *Lecture Notes in Computer Science*, pages 198–209. Springer, 2011. doi:10.1007/978-3-642-22212-2_18.

Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *SPAA '02: Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures*, pages 200–209, New York, NY, USA, 2002. ACM. ISBN 1-58113-529-7. doi:10.1145/564870.564906.

Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Can we elect if we cannot compare? In *SPAA '03: Proceedings of the 15th ACM Symposium on Parallel Algorithms and Architectures*, pages 324–332, New York, NY, USA, 2003. ACM. ISBN 1-58113-661-7. doi:10.1145/777412.777469.

Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems*, 40(2):143–162, February 2007. doi:10.1007/s00224-005-1223-5.

Lali Barrière, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M. Thilikos. Searching is not jumping. In *WG 2003: Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 2880 of *Lecture Notes in Computer Science*, pages 34–45. Springer, 2003. doi:10.1007/b93953.

Vic Baston and Shmuel Gal. Rendezvous search when marks are left at the starting points. *Naval Research Logistics*, 48(8):722–731, 2001. doi:10.1002/nav.1044.

Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *STOC '98: Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 269–278, New York, NY, USA, 1998. ACM. ISBN 0-89791-962-9. doi:10.1145/276698.276759.

Michael A. Bender and Donna K. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In *FOCS 1994: Proceedings of the 35th Symposium on Foundations of Computer Science*, pages 75–85, November 1994.

D. Bienstock. Graph searching, path-width, tree-width and related problems. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 5:33–49, 1991.

Daniel Bienstock and Michael A. Langston. Algorithm implications of the graph minor theorem. *Handbooks in Operation Research and Management Science*, 7:481–, 1995. doi:10.1016/S0927-0507(05)80125-2.

Daniel Bienstock and Paul Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12(2):239–245, June 1991. doi:10.1016/0196-6774(91)90003-H.

Lélia Blin, Pierre Fraigniaud, Nicolas Nisse, and Sandrine Vial. Distributed chasing of network intruders. In *SIROCCO 2006: Proceedings of the 13th International Colloquium on Structural Information and Communication Complexity*, pages 70–84, 2006. doi:10.1007/11780823_7.

Manuel Blum and Dexter Kozen. On the power of the compass (or, Why mazes are easier to search than graphs). In *FOCS 1978: Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 132–142. IEEE, October 1978. doi:10.1109/SFCS.1978.30.

R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers*, VI(5): 72–78, 1967.

Harry Buhrman, Matthew Franklin, Juan A. Garay, Jaap-Henk Hoepman, John Tromp, and Paul Vitányi. Mutual search. *Journal of the ACM*, 46(4):517–536, 1999. ISSN 0004-5411. doi:10.1145/320211.320232.

B. Bui Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267, 2003. ISSN 01290541.

J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. MaxProp: Routing for vehicle-based disruption-tolerant networks. In *INFOCOM 2006: Proceeding of the 25th IEEE International Conference on Computer Communications*, pages 1–11, April 2006. doi:10.1109/INFOCOM.2006.228.

Arnaud Casteigts, Serge Chaumette, and Afonso Ferreira. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *SIROCCO 2009: Revised Selected Papers of the 16th International Colloquium on Structural Information and Communication Complexity*, volume 5869 of *Lecture Notes in Computer Science*. Springer, 2009. doi:10.1007/978-3-642-11476-2_11.

Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Deterministic computations in time-varying graphs: Broadcasting under unstructured mobility. In *TCS 2010: Proceedings of the 6th IFIP International Conference on Theoretical Computer Science*, volume 323 of *IFIP*, pages 111–124. Springer, 2010. doi:10.1007/978-3-642-15240-5_9.

Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. In *ADHOC-NOW 2011: Proceedings of the 10th International Conference on Ad Hoc Networks and Wireless*, Lecture Notes in Computer Science, 2011. To appear (Preliminary version: http://arxiv.org/abs/1012.0009).

Jérémie Chalopin, Shantanu Das, Arnaud Labourel, and Euripides Markou. Tight bounds for scattered black hole search in a ring. In *SIROCCO 2011: Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity*, volume 6796 of *Lecture Notes in Computer Science*, pages 186–197. Springer, 2011a. doi:10.1007/978-3-642-22212-2_17.

Jérémie Chalopin, Shantanu Das, Arnaud Labourel, and Euripides Markou. Black hole search with finite automata scattered in a synchronous torus. In *DISC 2011: Proceedings of the 25th International Conference on Distributed Computing*, Lecture Notes in Computer Science. Springer, 2011b. To appear. Accessed as http://arxiv.org/abs/1106.6037.

Jérémie Chalopin, Shantanu Das, and Nicola Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. In *DISC 2007: Proceedings of the 21st International Symposium on Distributed Computing*, volume 4731 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2007. doi:10.1007/978-3-540-75142-7_11.

Jérémie Chalopin, Shantanu Das, and Peter Widmayer. Rendezvous of mobile agents in directed graphs. In *DISC 2010: Proceedings of the 24th International Symposium on Distributed Computing*, volume 6343 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2010. doi:10.1007/978-3-642-15763-9_27.

Jérémie Chalopin, Emmanuel Godard, Yves Métivier, and Rodrigue Ossamy. Mobile agent algorithms versus message passing algorithms. In *OPODIS 2006: Proceedings of the 10th International Conference on the Principles of Distributed Systems*, volume 4305 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2006. doi:10.1007/11945529_14.

Ruay Shiung Chang. Single step graph search problem. *Information Processing Letters*, 40(2):107–111, October 1991. doi:10.1016/0020-0190(91)90018-D.

Colin Cooper, Ralf Klasing, and Tomasz Radzik. Searching for black-hole faults in a network using multiple agents. In *OPODIS 2006: Proceedings of the 10th International Conference on Principles of Distributed Systems*, volume 4305 of *Lecture Notes in Computer Science*, pages 320–332. Springer, November 2006. doi:10.1007/11945529_23.

Colin Cooper, Ralf Klasing, and Tomasz Radzik. Locating and repairing faults in a network with mobile agents. In *SIROCCO 2008: Proceedings of the 15th International Colloquium on Structural Information and Communication Complexity*, volume 5058 of *Lecture Notes on Computer Science*, pages 20–32. Springer, 2008. doi:10.1007/978-3-540-69355-0_4.

Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71(2,3):229–242, 2006. ISSN 0169-2968.

Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Searching for a black hole in synchronous tree networks. *Combin. Probab. Comput.*, 16 (4):595–619, July 2007. doi:10.1017/S0963548306008133.

Shantanu Das. Mobile agent rendezvous in a ring using faulty tokens. In *ICDCN 2008: Proceedings of the 9th International Conference on Distributed Computing and Networking*, volume 4904 of *Lecture Notes in Computer Science*, pages 292–297. Springer, 2008. doi:10.1007/978-3-540-77444-0_29.

Shantanu Das, Paola Flocchini, Shay Kutten, Amiya Nayak, and Nicola Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science*, 385(1–3):34–48, October 2007. doi:10.1016/j.tcs.2007.05.011.

Shantanu Das, Paola Flocchini, Amiya Nayak, and Nicola Santoro. Distributed exploration of an unknown graph. In *SIROCCO 2005: Proceedings of the 12th International Colloquium on Structural Information and Communication Complexity*, volume 3499 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2005. doi:10.1007/11429647_10.

Shantanu Das, Paola Flocchini, Amiya Nayak, and Nicola Santoro. Effective elections for anonymous mobile agents. In *ISAAC 2006: Proceedings of the 17th International Symposium on Algorithms and Computation*, volume 4288 of *Lecture Notes in Computer Science*, pages 732–743. Springer, November 2006. doi:10.1007/11940128_73.

Shantanu Das, Matús Mihalák, Rastislav Srámek, Elias Vicari, and Peter Widmayer. Rendezvous of mobile agents when tokens fail anytime. In *OPODIS 2008: Proceeding of the 12th International Conference on the Principles of Distributed Systems*, volume 5401 of *Lecture Notes in Computer Science*, pages 463–480. Springer, 2008. doi:10.1007/978-3-540-92221-6_29.

Nick D. Dendris, Lefteris M. Kirousis, and Dimitrios M. Thilikos. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science*, 172 (1–2):233–254, February 1997. doi:10.1016/S0304-3975(96)00177-6.

Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. In *FCS 1990: Proceedings of the 31st Annual Symposium on the Foundations of Computer Science*, pages 355–361, 1990. doi:10.1109/FSCS.1990.89554.

Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, November 1999. doi:10.1002/(SICI)1097-0118(199911)32:3<265::AID-JGT6>3.0.CO;2-8.

Anders Dessmark, Pierre Fraigniaud, Dariusz R. Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, September 2006. doi:10.1007/s00453-006-0074-2.

Anders Dessmark and Andrzej Pelc. Optimal graph exploration without good maps. *Theoretical Computer Science*, 326(1–3):343–362, October 2004. doi:10.1016/j.tcs.2004.07.031.

Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, April 2004. doi:10.1016/j.jalgor.2003.10.002.

Stefan Dobrev, Paola Flocchini, Rastislav Královič, P Ružička, Guiseppe Prencipe, and Nicolas Santoro. Black hole search in common interconnection networks. *Networks*, 47(2):61–71, March 2006a. doi:10.1002/net.20095.

Stefan Dobrev, Paola Flocchini, Rastislav Královič, and Nicola Santoro. Exploring an unknown graph to locate a black hole using tokens. In *TCS 2006: Fourth IFIP International Conference on Theoretical Computer Science*, volume 209 of *IFIP International Federation for Information Processing*, pages 131–150. Springer Boston, December 2006b. doi:10.1007/978-0-387-34735-6_14.

Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile search for a black hole in an anonymous ring. In Jennifer L. Welch, editor, *Distributed Algorithms*, volume 2180 of *Lecture Notes in Computer Science*, pages 166–179, Oct 2001. ISBN 3-540-42605-1. doi:10.1007/3-540-45414-4_12.

Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agent protocols. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 153–162, New York, NY, USA, 2002. ACM. ISBN 1-58113-485-1. doi:10.1145/571825.571853.

Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Multiple agents rendezvous in a ring in spite of a black hole. In *OPODIS 2003: Revised Selected Papers from the 7th International Conference on Principles of Distributed Systems*, volume 3144 of *Lecture Notes in Computer Science*, pages 34–46. Springer, July 2003. doi:10.1007/b99477.

Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. *Distributed Computing*, 19(1):1–19, September 2006c. doi:10.1007/s00446-006-0154-y.

Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48(1):67–90, June 2007a. doi:10.1007/s00453-006-1232-z.

Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Improved bounds for optimal black hole search with a network map. In *SIROCCO 2004: Proceedings of the 11th International Colloquium on Structural Information and Communication Complexity*, volume 3104 of *Lecture Notes in Computer Science*, pages 111–122. Springer, September 2004. doi:10.1007/b98251.

Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Cycling through a dangerous network: A simple efficient strategy for black hole search. In *ICDCS 2006: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, pages 1–8, 2006d. doi:10.1109/ICDCS.2006.25.

Stefan Dobrev, Rastislav Královič, Nicola Santoro, and Wei Shi. Black hole search in asynchronous rings using tokens. In *CIAC 2006: Proceedings of the 6th Italian Conference on Algorithms and Complexity*, volume 3998 of *Lecture Notes in Computer Science*, pages 139–150. Springer, June 2006e. doi:10.1007/11758471_16.

Stefan Dobrev, Nicola Santoro, and Wei Shi. Locating a black hole in an un-oriented ring using tokens: The case of scattered agents. In *Euro-Par 2007: Proceedings of the 13th International Euro-Par Conference*, volume 4641 of *Lecture Notes in Computer Science*, pages 608–617. Springer, 2007b. doi:10.1007/978-3-540-74466-5_64.

Stefan Dobrev, Nicola Santoro, and Wei Shi. Using scattered mobile agents to locate a black hole in an un-oriented ring with tokens. *International Journal of Foundations of Computer Science*, 19(6):1355 – 1372, 2008. ISSN 01290541.

Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865, December 1991. ISSN 1042-296X. doi:10.1109/70.105395.

Miroslaw Dynia, Jakub Łopuszański, and Christian Schindelhauer. Why robots need maps. In *SIROCCO 2007: Proceedings of the14th International Colloquium Structural Information and Communication Complexity*, volume 4474 of *Lecture Notes in Computer Science*, pages 41–50. Springer, 2007. doi:10.1007/978-3-540-72951-8_5.

J.A. Ellis, I.H. Sudborough, and J.S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, August 1994. doi:10.1006/inco.1994.1064.

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32:374–382, April 1985. ISSN 0004-5411. doi:10.1145/3149.214121.

Paola Flocchini, Miao Jun Huang, and Flaminia L. Luccio. Decontaminating chordal rings and tori using mobile agents. *International Journal of Foundations of Computer Science*, 18(3):547–563, June 2007a. ISSN 01290541.

Paola Flocchini, Miao Jun Huang, and Flaminia L. Luccio. Decontamination of hypercubes by mobile agents. *Networks*, 52(3):167–178, October 2008a. doi:10.1002/net.20240. to appear.

Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. In *OPODIS 2007: Proceedings of the 11th International Conference on Principles of Distributed Systems*, volume 4878 of *Lecture Notes in Computer Science*, pages 105–118. Springer, April 2007b. doi:10.1007/978-3-540-77096-1_8.

Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. In *SIROCCO 2008: Proceedings of the 15th International Colloquium on Structural Information and Communication Complexity*, volume 5058 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008b. doi:10.1007/978-3-540-69355-0_5.

Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theoretical Computer Science*, 411(14-15):1583–1598, 2010a. doi:10.1016/j.tcs.2010.01.007.

Paola Flocchini, David Ilcinkas, and Nicola Santoro. Ping pong in dangerous graphs: Optimal black hole search with pure tokens. In *DISC 2008: Proceedings of the 22nd International Symposium on Distributed Computing*, volume 5218 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2008c. doi:10.1007/978-3-540-87779-0_16.

Paola Flocchini, David Ilcinkas, and Nicola Santoro. Ping pong in dangerous graphs: Optimal black hole search with pebbles. *Algorithmica*, pages 1–28, 2011. doi:10.1007/s00453-011-9496-3. Online first version.

Paola Flocchini, Matthew Kellett, Peter C. Mason, and Nicola Santoro. Map construction and exploration by mobile agents scattered in a dangerous network. In *IPDPS 2009: Proceedings of the 23rd IEEE International Symposium on Parallel & Distributed Processing*, pages 1–10, May 2009a. doi:10.1109/IPDPS.2009.5161080.

Paola Flocchini, Matthew Kellett, Peter C. Mason, and Nicola Santoro. Mapping an unfriendly subway system. In *FUN 2010: Proceedings of the 5th International Conference on Fun with Algorithms*, volume 6099 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2010b. doi:10.1007/978-3-642-13122-6_20.

Paola Flocchini, Matthew Kellett, Peter C. Mason, and Nicolas Santoro. Searching for black holes in subways. *Theory of Computing Systems*, 50(1):158–184, January 2012. doi:10.1007/s00224-011-9341-8.

Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Flaminia L. Luccio, Nicola Santoro, and Cindy Sawchuk. Mobile agents rendezvous when tokens fail. In *SIROCCO 2004: Proceedings of the 11th International Colloquium on Structural Information and Communication Complexity*, volume 3104 of *Lecture Notes in Computer Science*, pages 161–172. Springer, 2004a. doi:10.1007/b98251.

Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Nicola Santoro, and Cindy Sawchuk. Multiple mobile agent rendezvous in a ring. In *LATIN 2004: Proceedings of the 6th Latin American Symposium on Theoretical Informatics*, volume 2976 of *Lecture Notes in Computer Science*, pages 599–608. Springer, March 2004b. doi:10.1007/b95852.

Paola Flocchini, Flaminia L. Luccio, and Lisa Xiuli Song. Size optimal strategies for capturing an intruder in mesh networks. In *CIC 2005: Proceedings of the 2005 International Conference on Communications in Computing*, pages 200–206. CSREA Press, 2005a.

Paola Flocchini, Bernard Mans, and Nicola Santoro. Sense of direction: Definitions, properties, and classes. *Networks*, 32(3):165–180, October 1998. doi:10.1002/(SICI)1097-0037(199810)32:3<165::AID-NET1>3.0.CO;2-I.

Paola Flocchini, Bernard Mans, and Nicola Santoro. Tree decontamination with temporary immunity. In *ISAAC 2008: Proceedings of the 19th International Symposium on Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 330–341. Springer, 2008d. doi:10.1007/978-3-540-92182-0_31.

Paola Flocchini, Bernard Mans, and Nicola Santoro. Exploration of periodically varying graphs. In *ISAAC 2009: Proceedings of the 20th International Symposium* *on Algorithms and Computation*, volume 5878 of *Lecture Notes in Computer Science*, pages 534–543. Springer, 2009b. doi:10.1007/978-3-642-10631-6_55.

Paola Flocchini, Amiya Nayak, and Arno Schulz. Cleaning an arbitrary regular network with mobile agents. In *ICDCIT 2005: Proceedings of the 2nd International Conference on Distrubted Computing and Internet Technology*, volume 3816 of *Lecture Notes in Computer Science*, pages 132–142. Springer, December 2005b. doi:10.1007/11604655_17.

Paola Flocchini and Nicola Santoro. Distributed security algorithms by mobile agents. In *ICDCN 2006: Proceedings of the 8th International Conference on Distributed Computing and Networking*, volume 4308 of *Lecture Notes in Computer Science*, pages 1–14. Springer, March 2006. doi:10.1007/11947950_1.

Paola Flocchini and Nicola Santoro. *Mobile agents in networking and distributed computing*, chapter Distributed security aglorithms for mobile agents. Agent Technology. Wiley, 2010. To be released June 2010.

Fedor V. Fomin and Petr A. Golovach. Graph searching and interval completion. *SIAM Journal on Discrete Mathematics*, 13(4):454–464, 2000. doi:10.1137/S0895480199350477.

Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, June 2008. doi:10.1016/j.tcs.2008.02.040.

Fedor V. Fomin, Dimitrios M. Thilikos, and Ioan Todinca. Connected graph searching in outerplanar graphs. *Electronic Notes in Discrete Mathematics*, 22:213–216, 2005. ISSN 1571-0653. doi:10.1016/j.endm.2005.06.032. 7th International Colloquium on Graph Theory.

Pierre Fraigniaud, Leszek Gąsieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. In *LATIN 2004: Proceedings of the 6th Latin American Symposium Theoretical Informatics*, volume 2976 of *Lecture Notes in Computer Science*, pages 141–151. Springer, March 2004. doi:10.1007/b95852.

Pierre Fraigniaud, Leszek Gasieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. *Networks*, 48(3):166–177, October 2006a. doi:10.1002/net.20127.

Pierre Fraigniaud and David Ilcinkas. Digraphs exploration with little memory. In *STACS 2004: Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, pages 246–257. Springer, March 2004. doi:10.1007/b96012.

Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2–3):331–344, November 2005. doi:10.1016/j.tcs.2005.07.014.

Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Tree exploration with an oracle. In *MFCS 2006: Proceedings of the 31st International Symposium on the Mathematical Foundations of Computer Science*, volume 4162 of *Lecture Notes in Computer Science*, pages 24–37. Springer, 2006b. doi:10.1007/11821069_2.

Pierre Fraigniaud and Nicolas Nisse. Monotony properties of connected visible graph searching. In *WG 2006: Revised Papers of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 4271 of *Lecture Notes in Computer Science*, pages 229–240. Springer, October 2006a. doi:10.1007/11917496_21.

Pierre Fraigniaud and Nicolas Nisse. Connected treewidth and connected graph searching. In *LATIN 2006: Proceedings of the 7th Latin American Symposium on Theoretical Informatics*, volume 3887 of *Lecture Notes in Computer Science*, pages 479–490. Springer, February 2006b. doi:10.1007/11682462_45.

H. Garcia-Molina. Elections in a distributed computing system. *IEEE Transactions on Computers*, C-31(1):48–59, January 1982. ISSN 0018-9340. doi:10.1109/TC.1982.1675885.

Leszek Gasieniec, Andrzej Pelc, Tomasz Radzik, and Xiaohui Zhang. Tree exploration with logarithmic memory. In *SODA 2007: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 585–594, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5.

Peter Glaus. Locating a black hole without the knowledge of incoming links. In *ALGOSENSORS 2009: Revised Selected Papers from the 5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, volume 5804 of *Lecture Notes in Computer Science*, pages 128–138. Springer, 2009. doi:10.1007/978-3-642-05434-1_13.

M.A. Haddar, A. Hadj Kacem, Y. Metivier, M. Mosbah, and M. Jmaiel. Electing a leader in the local computation model using mobile agents. In *AICCSA 2008: Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications*, pages 473–480, 31 2008-April 4 2008. doi:10.1109/AICCSA.2008.4493575.

S. Hansen and M. Eldredge. Intruder isolation and monitoring. In *Proceedings from the 1st Security Workshop, USENIX*, pages 63–64, 1988.

Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1):69–85, February 2004.

Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, July 1992. doi:10.1016/0020-0190(92)90234-M.

Lefteris M. Kirousis and Christos H. Papadimitriou. Interval graphs and searching. *Discrete Mathematics*, 55(2):181–184, July 1985.

Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47:205–218, 1986. doi:10.1016/0304-3975(86)90146-5.

Ralf Klasing, Euripides Markou, Tomasz Radzik, and Fabiano Sarracco. Approximation bounds for black hole search problems. In *OPODIS*, volume 3974 of *Lecture Notes in Computer Science*, pages 261–274. Springer, January 2005. doi:10.1007/11795490_21.

Ralf Klasing, Euripides Markou, Tomasz Radzik, and Fabiano Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theoretical Computer Science*, 384(2–3):201–221, October 2007. doi:10.1016/j.tcs.2007.04.024.

Ralf Klasing, Euripides Markou, Tomasz Radzik, and Fabiano Sarracco. Approximation bounds for black hole search problems. *Networks*, 52(4):216–226, December 2008. doi:10.1002/net.20233.

Adrian Kosowski, Alfredo Navarra, and Maria Cristina Pinotti. Synchronization helps robots to detect black holes in directed graphs. In *OPODIS 2009: Proceedings of the 13th International Conference on the Principles of Distributed Systems*, volume 5923 of *Lecture Notes in Computer Science*, pages 86–98. Springer, 2009. doi:10.1007/978-3-642-10877-8_9.

Dexter Kozen. Automata and planar graphs. In *FCT 1979: Proceedings of the Conference on the Fundamentals of Computation Theory*, pages 243–254, 1979.

Evangelos Kranakis and Danny Krizanc. An algorithmic theory of mobile agents. In *TGC 2006: Revised Selected Papers from the Second Syposium on Trustworthy Global Computing*, volume 4661 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2006. doi:10.1007/978-3-540-75336-0_6.

Evangelos Kranakis, Danny Krizanc, and Euripides Markou. Mobile agent rendezvous in a synchronous torus. In *LATIN 2006: Proceedings of the 7th Latin American Symposium on Theoretical Informatics*, volume 3887 of *Lecture Notes in Computer Science*, pages 653–664. Springer, 2006a. doi:10.1007/11682462_60.

Evangelos Kranakis, Danny Krizanc, and Pat Morin. Randomized rendez-vous with limited memory. In *LATIN 2008: Proceedings of the 8th Latin American Symposium on Theoretical Informatics*, volume 4957 of *Lecture Notes in Computer Science*, pages 605–616. Springer, 2008. doi:10.1007/978-3-540-78773-0_52.

Evangelos Kranakis, Danny Krizanc, and Sergio Rajsbaum. Mobile agent rendezvous: A survey. In *SIROCCO 2006: Proceedings of the 13th International Colloquium on Structural Information and Communication Complexity*, volume 4056 of *Lecture Notes in Computer Science*, pages 1–9. Springer, June 2006b. doi:10.1007/11780823_1.

Evangelos Kranakis, Danny Krizanc, Nicola Santoro, and Cindy Sawchuk. Mobile agent rendezvous in a ring. In *ICDCS 2003: Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 592–599, May 2003. doi:10.1109/ICDCS.2003.1203510.

Andrea S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, April 1993. ISSN 0004-5411. doi:10.1145/151261.151263.

Thomas Lengauer. Black-white pebbles and graph separation. *Acta Informatica*, 16(4):465–475, December 1981. doi:10.1007/BF00264496.

Cong Liu and Jie Wu. Scalable routing in cyclic mobile networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(9):1325–1338, September 2009. ISSN 1045-9219. doi:10.1109/TPDS.2008.218.

Fabrizio Luccio, Linda Pagli, and Nicola Santoro. Network decontamination with local immunization. In *IPDPS 2006: Proceedings of the 20th International on Parallel and Distributed Processing Symposium*, pages 8 pp.–, April 2006. doi:10.1109/IPDPS.2006.1639553.

Flaminia L. Luccio. Intruder capture in Sierpiński graphs. In *FUN 2007: Proceedings of the 4th International Conference on Fun with Algorithms*, volume 4475 of *Lecture Notes in Computer Science*, pages 249–261. Springer, 2007. doi:10.1007/978-3-540-72914-3_22.

Flaminia L. Luccio. Contiguous search problem in Sierpiński graphs. *Theory of Computing Systems*, 44(2):186–204, February 2009. ISSN 1432-4350. doi:10.1007/s00224-008-9116-z.

Fillia Makedon and Ivan Hal Sudborough. Minimizing width in linear layouts. In *ICALP 1983: Proceedings of the 10th Colloquium on Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 478–490. Springer, April 1983. doi:10.1007/BFb0036931.

N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35(1):18–44, January 1988. ISSN 0004-5411. doi:10.1145/42267.42268.

Stano Miklik. *Exploration in faulty networks*. Ph.d. thesis, Faculty of Mathematics, Physics, and Informaticcs, Comenius University, Bratislava, Slovakia, 2010.

Stewart W. Neufeld. A pursuit-evasion problem on a grid. *Information Processing Letters*, 58(1):5–9, April 1996. doi:10.1016/0020-0190(96)00025-7.

Regina O'Dell and Roger Wattenhofer. Information dissemination in highly dynamic graphs. In *DIALM-POMC '05: Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing*, pages 104–110, New York, NY, USA, 2005. ACM. ISBN 1-59593-092-2. doi:10.1145/1080810.1080828.

Petrișor Panaite and Andrzej Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33(2):281–295, November 1999. doi:10.1006/jagm.1999.1043.

T. Parsons. The search number of a connected graph. In *Proceedings of the 9th Southeastern Conference on Combinatorics, Graph Theory and Computing*, Utilitas Mathematica, pages 549–554, 1978a.

T. D. Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs*, volume 642 of *Lecture Notes in Mathematics*, pages 426–441. Springer, November 1978b. doi:10.1007/BFb0070400.

M.O. Rabin. Maze threading automata. Technical report seminar talk, University of California at Berkeley, 1967.

Nicola Santoro. Sense of direction, topological awareness and communication complexity. *SIGACT News*, 16:50–56, July 1984. ISSN 0163-5700. doi:10.1145/1008959.1008961.

Nicola Santoro. *Design and analysis of distributed algorithms*. Wiley-Interscience, 2007. doi:10.1002/0470072644.

P.D Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, May 1993. doi:10.1006/jctb.1993.1027.

Claude Shannon. Presentation of a maze-solving machine. In *Proceedings of the 8th Conference of the Josiah Macy Jr. Foundation (Cybernetics)*, pages 173–180, 1951.

Wei Shi. Black hole search with tokens in interconnected networks. In *SSS 2009: Proceedings of teh 11th International Symposium on the Stabilization, Safety, and Security of Distributed Systems*, volume 5873 of *Lecture Notes in Computer Science*, pages 670–682. Springer, 2009. doi:10.1007/978-3-642-05118-0_46.

Jonathan D.H. Smith. Minimal trees of given search number. *Discrete Mathematics*, 66(1–2):191–202, August 1987.

Y. Stamatiou and D. Thilikos. Monotonicity and inert fugitive search games. In *Proceedings of the 6th Twente Workshop on Graphs and Combinatorial Optimization*, volume 3 of *Electronic Notes on Discrete Mathematics*, 1999.

Bernharda von Stengel and Ralph Werchner. Complexity of searching an immobile hider in a graph. *Discrete Applied Mathematics*, 78(1–3):235–249, October 1997.

Ichiro Suzuki, Yamashit, Masafumi, Hideki Umemoto, and Tsunehiko Kameda. Bushiness and a tight worst-case upper bound on the search number of a simple polygon. *Information Processing Letters*, 66(1):49–52, April 1998. doi:10.1016/S0020-0190(98)00029-5.

Ichiro Suzuki and Masafumi Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, 1992. doi:10.1137/0221051.

Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, January 1995. doi:10.1016/0304-3975(94)00160-K.

Dimitrios M. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics*, 105(1–3):239–271, October 2000.

Mitsuharu Yamamoto, Koichi Takahashi, Masami Hagiya, Shin ya Nishizaki, and Tetsuo Tamai. Formalization of graph search algorithms and its applications. In *TPHOLs 1998: Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics*, volume 1479 of *Lecture Notes in Computer Science*, pages 479–496. Springer, January 1998. doi:10.1007/BFb0055153.

Boting Yang, Danny Dyer, and Brian Alspach. Sweeping graphs with large clique number. In *ISAAC 2004: Proceedings of the 15th International Symposium on*

Algorithms and Computation, volume 3341 of *Lecture Notes in Computer Science*, pages 908–920. Springer, December 2004. doi:10.1007/b104582.

Xiangdong Yu and Moti Yung. Agent rendezvous: A dynamic symmetry-breaking problem. In *ICALP 1996: Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 610–621. Springer, January 1996. doi:10.1007/3-540-61440-0_163.

Xiaolan Zhang, Jim Kurose, Brian Neil Levine, Don Towsley, and Honggang Zhang. Study of a bus-based disruption-tolerant network: Mobility modeling and impact on routing. In *MobiCom '07: Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, pages 195–206, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-681-3. doi:10.1145/1287853.1287876.

Zhensheng Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges. *IEEE Communications Surveys Tutorials*, 8(1):24–37, 2006. ISSN 1553-877X. doi:10.1109/COMST.2006.323440.

APPENDIX A

Harry Potter without the Marauder's Map

A Plain Language Explanation of this Thesis

For my parents.

I started out writing this appendix as a full plain language version of the thesis. The problem is that it is very hard to explain the concepts in the thesis in plain language. As my significant other has noted, my field uses words that are all recognizably English, but uses them in a way that is completely incomprehensible to non-specialists. As a result, my first try was about as understandable as the thesis itself. In fact, you would be better off just reading the thesis because I take the time to define the most confusing terms, and many of the terms and notation that I do not define can be found on the following Wikipedia pages: Graph (mathematics), Set (mathematics), and List of mathematical symbols. If you do decide to give reading the thesis proper a go, keep those pages handy. In lieu of a full version, I present here a brief explanation of what is presented in the thesis using the world of Harry Potter as an example.

What could Harry Potter or the marauder's map possibly have to do with a thesis in computer science? Harry Potter is the protagonist of the Harry Potter series of books by J.K. Rowling. The marauder's map, which appears first in *Harry Potter and the Prisoner of Azkaban*, is a magical device that allows Harry to move around the increasingly dangerous Hogwarts School of Witchcraft and Wizardry, where he is a student, in relative safety. The map shows the location of everything and everyone at the school with the exception of a few rooms—the Chamber of Secrets and the Room of Requirement—that are crucial to the plots of several of the books. The work of this thesis is to create maps of similarly dangerous environments. However, there is no magic. I briefly explain how Harry could have achieved mostly the same result without the use of the marauder's map. I will show how it could be done both at Hogwarts and in the London Underground.

The main work of the thesis is on algorithms. An *algorithm* is a procedure followed by someone or something to accomplish a task or calculation. A recipe for lasagna is an algorithm used by a cook to solve the *problem* of making lasagna. The recipe works on the assumption that you have the ingredients listed. We refer to these ingredients as the *model* in which the recipe solves the problem. If I give you ingredients that do not fit the model, such as giving you everything except the lasagna noodles, then you cannot successfully solve the problem of making lasagna.

The algorithms in the thesis are all about mapping dangerous environments. Two algorithms map places like Hogwarts and the other two map subway systems like the London Underground. Each solves the problem in a slightly different model, meaning under slightly different assumptions of what the environment looks like and what information is available where. In the thesis, each algorithm includes a proof of its *correctness*, which is a mathematical proof that shows that the algorithm *always* solves the problem for any environment that matches the model. Each algorithm also includes a proof of its *complexity*, which is a measure of how efficient it is at solving the problem or, put another way, how complex the problem is in the model. Three of the four algorithms are optimal, meaning that they are as efficient as it is possible to be. I will explain how all four work from the perspective of a student at Hogwarts.

Mapping Hogwarts

We want to replace the marauder's map, so that Harry can move around Hogwarts in relative safety. The danger is that some authority figure like Snape or some Death Eater like Draco or Lucius Malfoy will capture Harry when he leaves the relative safety of Gryffindor tower. We want to create a map of Hogwarts that shows the locations of all the dangerous places. We are not talking just rooms, hallways, and closets where these malicious characters might be lying in wait, we are also talking doors and staircases that are enchanted to whisk the searcher away. In order to replace the magic of the marauder's map, we need people. We will use Dumbledore's Army, the student club Harry created in *Harry Potter and the Order of the Phoenix* to practice defence against the dark arts. We know that some of the students will be captured. We assume that there are enough students that the malicious characters and enchanted doors and staircases cannot get everyone. We assume that rooms, hallways, and closets that are dangerous always remain dangerous. We assume that each safe room, hallway, and closet has a whiteboard that is used by the students to communicate with one other and coordinate the mapping. Finally, we assume that each student starts off the search in his or her bedroom. These assumptions are part of the network model (Chapter 2) in the thesis.

We describe the first Hogwarts algorithm (Chapter 4) from the perspective of Katie Bell, a member of both Gryffindor house and Dumbledore's Army. Katie starts by setting up the whiteboard in her room. Even though each student starts off in his or her own bedroom, the goal of the algorithm is for all the students to end up using the same whiteboard to coordinate their activities, which will eventually have a map of the entire castle. Katie starts by simply writing down the name of the room—"Katie Bell's bedroom"—and identifying the doors leading out of it. The whiteboard now contains the information need to coordinate the search from her room. She chooses a door and writes down that she is going to explore it. She then goes through the door to visit the room, hallway, or closet on the other side.

Let us say that the door leads to her bathroom and neither the door nor the bathroom are dangerous. She finds the whiteboard in the bathroom is blank, so she writes down which door to take from the bathroom to get to the whiteboard that contains the coordinating information, which is currently the whiteboard in her bedroom. The bathroom might have a door to a closet for instance. Before Katie can go through that door, she has to go back to her bedroom and report that the bathroom and the door she used to get there are both safe. In fact, she has to return to update the whiteboard containing the coordinating information—wherever it is—for every door or staircase she takes to another room, hallway, or closet. When Katie returns to her bedroom, she not only marks that the door and the bathroom are safe, she also adds the door in the bathroom to the list of doors and staircases that need to be explored.

If Katie had been captured going through the door or in bathroom, the information on the whiteboard in her bedroom would keep any other student from entering the bathroom through that door, possibly saving another student from capture. The student will see that someone has gone to explore the door and the room beyond and has not returned. Whether Katie is just being slow in searching or if she has been captured cannot be determined until the entire castle has been searched.

Katie decides to try another door in her room, which leads out into the hallway. When she gets there, she finds that someone has already been there. Like Katie did in the bathroom, the first student to find the hallway has written down the door to take to get to the coordinating information he or she is using for his or her search. The first thing that Katie has to do is go back to her bedroom where her coordinating information is and record that she safely made it through the door to the hallway and that the hallway has x number of doors and staircases leading off of it. She then returns to the hallway and goes through the door indicated on the whiteboard in the direction of the other student's coordinating information. She follows the directions on the whiteboards in each of the rooms, hallways, and closets she finds until she reaches a whiteboard with coordinating information. Assume that the whiteboard says "Neville Longbottom's bedroom". Since Neville's name appears after Katie's in the alphabet, she writes down all the information on the board in her notebook, erases the whiteboard, and writes down which door to take to get back to her bedroom. She then walks back along the same path to her bedroom that she used to get the Neville's bedroom, changing each whiteboard along the way to point towards her bedroom. When she gets to the hallway again, she re-enters her room and, as long as someone has not taken her coordinating information, she adds all the information from Neville's whiteboard to her own.

The process continues for all the students until there is one whiteboard in someone's bedroom where the number of rooms, hallways, and closets listed as safe is equal to the number of known safe rooms, hallways, and closets (a number that all the students have to know from the start in order for the algorithm to work). In which case, the students are done and they have a map showing which doors and staircases to take to safely get from any safe room to any other safe room. Harry can now move around the castle as safely as if he had the marauder's map.

The second Hogwarts algorithm (Chapter 5) works just like the first algorithm. However, the model is changed to allow the staff of Hogwarts to declare certain doors and staircases off limits to students. This situation happens several times in the Harry Potter series. The problem is there might be whiteboards that say that the students should take a now off limits door or staircase to get to the whiteboard that contains the coordinating information. There may be no other known safe route to get to there. To deal with this problem, we first need more information. We need every whiteboard to contain coordinating information. So, for instance, when Katie explores some room, she does not just write down which door or staircase to take to get back to the room with the coordinating information, she also writes down the list of doors and staircases that can be explored from that room on the room's whiteboard and on all the whiteboards on the way back. That way, if any door or staircase along the way back to the room with coordinating information is declared off limits, any student can simply start exploring again using the coordinating information on the whiteboard in that room, hallway, or closet. Similarly, if Katie is blocked from getting to a door or staircase she said she would explore, she erases that part of the map from the whiteboards on her side of the blockage to make sure that no other student has to make a fruitless trip.

Again, the process continues for all the students until there is one whiteboard in someone's bedroom that lists all the safe rooms. The map, in this case, cannot be accurate because doors and staircases could be declared off limits after that part of the map is complete. Nevertheless, every whiteboard will have a list of the safe doors and staircases leading from it.

The first Hogwarts algorithm is as efficient as possible. The number of uses of doors and staircases by the students is in direct proportion to the number of doors and staircases times the number of safe rooms, hallways, and closets. Even though Katie and the other students have to walk all the way back to the room that contains the coordinating information for every door or staircase they explore, there is no way to cut down on how many doors and staircases they have to traverse to map any Hogwarts-like castle. We know the algorithm's complexity is “optimal” because a student in Slovakia proved in his Ph.D. thesis that our algorithm uses the least number of “moves” between rooms possible.

The second algorithm is slightly less efficient than the first because it has to deal with doors and staircases being declared off limits. The number of doors and staircases the students have to traverse is in direct proportion to the number needed to solve the first algorithm times the number of doors and staircases. We have no proof that the algorithm's complexity is optimal because such proofs are exceptionally hard to do, but we suspect it is.

Mapping the London Underground

There is no marauder's map for the London Underground in the world of Harry Potter. Nevertheless, say it is the holidays and Harry wants to spend them at the headquarters of the Order of the Phoenix in London. The Hogwarts Express arrives at King's Cross station and Harry needs a map of the safe stations in the London Underground so that he can move around safely during the holidays. Using the two algorithms in the thesis, we can create a map for his arrival and one for his departure.

The subway model, which is introduced in this thesis (Chapter 2), is a computer network model that is in some ways much more complicated than the average subway system and in other ways much more simple. If you want to understand all the intricacies, it is worthwhile to read the thesis proper. I will just give you an idea of how the algorithms work.

We want to map the system to show the dangerous stations where malicious characters are waiting to capture our searchers. At the same time, the subway trains themselves are safe. We assume that the subway has some unusual properties that make sense in a computer network but not in the London Underground. The students work without a map. Perhaps because they hide on the trains, they cannot tell which station they are in, only that the train has stopped. They do know how many stops each train makes.

The first London Underground algorithm (Chapter 7) assumes that all the students of Dumbledore's Army start from the same station—in this case, King's Cross St. Pancras tube station—and that they communicate using whiteboards in the stations. Again, I describe the algorithm from Katie Bell's perspective. Katie starts by checking the whiteboard to see if it already contains coordinating information. Let us assume that it does and she chooses to visit or explore one of the stops of a train leaving King's Cross St. Pancras. She boards the train, goes to the stop, and disembarks. If she is not captured, she gets back on the same train going in the same direction and rides it all the way around until it gets back to King's Cross St. Pancras. Remember, she does not know at which station the train is stopping. She only knows the number of stops the train makes, so she counts the number of stops until she knows she is back at King's Cross St. Pancras.

Assume that the stop she explored connected to trains on subway lines that do not go through King's Cross St. Pancras. Katie would write down the identification of all the trains stopping there and travel back to King's Cross St. Pancras to see if anyone else had found the same trains. If not, she would add those trains to the coordinating information no the whiteboard at King's Cross St. Pancras, just like she added the door to the closet in her bathroom in the first Hogwarts algorithm, and point the students to the stop she was exploring if they want to search the stops of those trains.

The process continues until all the trains in the system have been added to the coordinating information and all the stops have been explored. Harry can then use the map at King's Cross St. Pancras to move about the London Underground in safety.

The second algorithm (Chapter 8) assumes that all the students of Dumbledore's Army start from the stations closest to their parents' houses in London and that they communicate using whiteboards on the subway trains. Let us say that Katie starts from Shepherd's Bush station. She boards the first train that arrives, which is on the Central line. This train becomes her home train. The premise of this algorithm is much simpler than the first London Underground algorithm. Katie carries a notebook that lists everything she knows about the subway system. She merges the information in her notebook with the information on the whiteboard of every train on every line that she finds. But, she is not the only one synchronizing the map in her notebook; all the students are doing the same thing. So, over time, as Katie sets out from her home train to visit all the trains she knows about to explore their stops, she learns about new trains on other lines that she needs to explore. In fact, every student has to visit every train on every line to see if there are stops that have not been explored. In the end, when every remaining student's map and every train's map contains all the trains on all the lines and every stop has been explored, Harry can use the map on the whiteboard of any train in the system to move about the London Underground in safety.

Instead of moves through doors and staircases, we measure the efficiency or complexity of the London Underground algorithms in the number of times the subway trains travel between stations while a student is waiting for or riding on them. It turns out that both algorithms, although very different in how they work, have the same efficiency and that they cannot be more efficient. The efficiency of the algorithms is a function of the number of students, the number of subway trains, and the maximum number of stops made by a subway train in the system. I know that their efficiency cannot be improved because I show in Chapter 6 that no algorithm could do better.

Conclusion

I guess you are still wondering how this relates to computers. Well, take the students of Dumbledore's Army and make them mobile agents capable of following an algorithm and moving around a computer network. Take Hogwarts and exchange all the rooms, hallways, and closets for computers and all the doors and staircases for communications links. Instead of malicious wizards and enchanted doorways, we have faulty computers that we call "black holes" and faulty networking equipment that we call "black links". For the London Underground, replace every station with a computer. Instead of trains travelling between stations, imagine communications links, like wireless connections or network traffic, that only appear periodically. The dangerous stations are the black holes.

The big picture is not about the algorithm itself but what the algorithm says about the environment in which it is run. Basically, the complexity is a measure of how good a computer network or subway system is at doing a computation, which, in this case, is the creation of a map. It turns out that computing a map of a network that looks like Hogwarts is pretty efficient but doing the same thing in the London Underground is not, even if you cannot do better in either case.