Faulty Node Repair and Dynamically Spawned Black Hole Search in Clouds

by

Mengfei Peng

A Thesis Submitted in Partial Fulfilment

of the Requirements for the Degree of

Master of Science

in

The Faculty of Business and Information Technology

Program

University of Ontario Institute of Technology

March 2015

Abstract

The reality of the constant emergence of new threats justifies the necessity to protect network assets and mitigate the risks associated with attacks. In this context, eliminating faulty network entities in the distributed environment such as cloud of the clouds and smart grids catches the attention of the researchers. Among all threats, black hole is a severe and pervasive one which models a network site that disposes any incoming data without leaving any trace of such distraction. Black hole search is the process that leverages mobile agents to locate black holes in a fully distributed way.

In this paper, we first review the state-of-the-art research in this area by categorizing the research results based on the adopted network models, being either synchronous or asynchronous. Most of the existing works focus on locating a single black hole. As for multiple black holes, the problem becomes even more complex.

For the study of multiple black hole search, we introduce a new attack model that involves not only multiple faulty nodes in the network (a type of black hole), but also a gray virus that can again infect a previously repaired faulty node. Under such a model, the multiple faulty node search problem becomes more complex and realistic. We analyze the proposal model and identify key observations about the multiple faulty node search/location problem. We introduce one-stop and multi-stop gray virus and study the faulty node repair and black hole search problem.

We first propose solutions that use a token model to solve the problem caused by

a multi-stop gray virus in an asynchronous arbitrary network topology. Also under the token model, we then present solutions for the problem caused by this one-stop gray virus in an asynchronous ring network. Apart from the token model, we continue to study the problem caused by the one-stop virus using a whiteboard model, more particularly, with only one whiteboard in the homebase node in an asynchronous ring network

After proposing the new model and our algorithms, we conclude some future work on both single and multiple black holes search. We also highlight some open problems on the one-stop and multi-stop gray virus.

Acknowledgments

I would like to give my sincere thanks to my supervisor Dr. Wei Shi for her time, patience, and infinite support for my research. Without her help, none of the three papers that I have submitted would be possible.

In addition, I would like to thank all colleagues in my research lab for their help.

Furthermore, no words can express gratitude to my parents for their love, support, and encouragement.

Table of Contents

List of Tables

List of Figures

List of Acronyms

Acronyms	Definition
PT	Pure Token Model
FTF	Face-to-face Model
WB	Whiteboard Model
ET	Enhanced Token Model
DIS	Dispersed Agents
CL	Co-located Agents
SD	Sense of Direction
NT	Network Topology
GV	Gray Virus
BH	Black Hole

Chapter 1

Introduction, Motivation, and Contribution

1.1 Introduction

Over the past few decades, as network-based services have become prevalent, so has the need for effective diagnosis of all-too-frequent network anomalies and faults. Among these, a *black hole* is a severe and pervasive problem. A black hole models a computer that is accidentally off-line or a network site in which a resident process (e.g., an unknowingly-installed virus) deletes any visiting agents or incoming data upon their arrival without leaving any observable trace [31]. For example, in a cloud, an accidentally offline or malfunctioning computer node that causes loss of essential data for cloud users or a cloud system would immediately turn this node into a black hole which would compromise the quality of the cloud service. Moreover, any undetectable crash failure of a site in an asynchronous network transforms that site into a black hole. Web search engines use web crawlers, which can traverse the web by following hyper-links and storing downloaded pages, to collect data into a large database that is later indexed for efficient execution of user queries [70]. However, when a web crawler visits a crashed site (a black hole) and is eliminated, not only

the information of the current site but also all the previous collected indexing data will not be available. If the crashed site cannot be located, the web crawlers which execute the same protocol may all die in the black hole and no index data can be updated to the database of the web search engine. Consequently, effective detection of a black hole is a critical issue for the successful deployment of services in computer networks.

A *mobile agent* is an abstract and autonomous software. As such, these agents are versatile and robust in changing environments, and can be programmed to work in co-operative teams. Such team members may have different complementary specialties, or be duplicates of one another [54]. For the black hole search problem, one or a team of identical agents are used to perform the task. These agents have limited computing capabilities and bounded storage. They all obey an identical set of behavioural rules (referred to as the "protocol"), and can move from node to neighbouring node. Also, these agents are anonymous (i.e., do not have distinct identifiers) and autonomous (i.e., each has its own computing and bounded memory capabilities). By virtue of these merits, they are always adopted to locate the black holes in computer networks. There are several advantages of using mobile agents, such as they can reduce the network load, overcome network latency, encapsulate protocols, execute asynchronously and autonomously, and adapt dynamically [64]. To locate the black hole, there are two commonly used methods: a central controller is dedicated to send Ping messages constantly, or forcing each node to send a heartbeat message to the central controller periodically, each of which costs large network traffic. On the other hand, the only traffic caused by black hole search is on mobile agents' traveling during their search.

Specifically, *black hole search* is a task that allows a team of mobile agents to collaborate with each other to locate black holes within finite time, and eventually leaves at least one agent surviving and knowing all the edges leading to the black holes [38]. Many distinct uses of mobile agents to locate a single *black hole* in a

computer network have been studied in different contexts [7, 17, 20, 29, 44, 60].

In practice we usually abstract a network into a graph $G(V, E)$ where nodes in V represent computer hosts and edges in E represent network links. As for the agents, most studies leverage *anonymous agents* (aka identical agents) for black hole search where these mobile agents are indistinguishable from each other (no identification or signature on the agents). The agents also follow the same routing protocol to identify and report any black hole. Moreover, for comparison purposes, some recent results with regard to different types of malicious hosts are also discussed.

Flocchini *et al.* [51] surveyed the black hole search problem and defined the model of asynchronous and synchronous networks in 2006. This survey also introduced the black hole search problem as a special case of exploring and mapping an unknown environment. In their survey [65], Markou *et al.* discussed previous research papers which identifyed hostile nodes. In this survey, the authors focused on synchronous special trees, arbitrary trees, and arbitrary graphs. Meanwhile, they also briefly mentioned co-located agents in asynchronous rings using a whiteboard model. Later, Zarrad *et al.* [73] briefly introduced and classified the black hole search problem in synchronous and asynchronous networks without detailed assumptions.

At the very early stage, a large body of existing literature on unknown graph exploration problems always assumes that the underlying network graph does not contain any other types of malicious entities [3]. As the studies continue, more and more attention is drawn to the security issues on graph exploration which is also called *dangerous graph search* and includes detecting and locating black holes (as one type of malicious hosts), malicious agents, or faulty links [18].

1.2 Motivation

In reality, many computer faults/virus cannot be completely removed by anti-virus software. After a repair, a previously infected node may still be more vulnerable than the normal ones, and can be easily reinfected, for example, with fast spreading worms mentioned in [72], such as W32/CodeRed, Linux/Slapper, W32/Blaster or Solaris/Sadmind, a host can be exploited only if the system has a vulnerability known a priori. Such virus behaviour is commonly referred to as vulnerability dependency. In cloud computing, the term vulnerability refers to the flaws in a system that allow an attack to be successful [56]. This vulnerability security issue has been widely discussed in research work such as [4, 19, 55].

For instance, a hacker injects into a computer host a virus that can delete any incoming data, which may later be removed by an anti-virus agent. However, after repair, an unknown vulnerability remains on the host that enables the hacker's next attack. Cooper *et al.* [21] first introduced a type of a weaker black hole, called a *hole*, which also eliminates any incoming data, however, can be repaired by the first encountering agent. With the vulnerability dependency, the hacker can inject even more powerful virus and turns the target host into a black hole at some point in time. Since the hacker may attack the host for any period, the duration of the host being a black hole can vary from instant to permanent. Under this attack model, we introduce a *Faulty Node Repair and Dynamically Spawned Black Hole Search problem* (*Repair and Search problem* for brevity).

When a black hole is repairable, we call it a faulty node. To repair the fault, there is a cost, for example, part of the content of an agent is a repair kit, so that after repair the agent can no longer continue to explore the network [21].

In this new attack model, as with the holes mentioned above, a *faulty node* can eliminate any incoming data and can be repaired by an encountering agent at the

expense of its life. If one or more agents simultaneously enter a faulty node, one agent will die after repairing the fault, and all other agents die immediately.

After this repair, the repaired node behaves like a normal one, however, it can be again infected by a gray virus. A *gray virus* (GV for brevity) is a piece of malicious software which can infect (due to the vulnerability of a repaired node) a repaired node by residing in it and turning it into a black hole; it has no destructive power on a normal node or link. After infecting a vulnerable node (a faulty node which has been repaired), a GV that can no longer travel to other nodes is called a one-stop gray virus, otherwise, a multi-stop gray virus. As faulty nodes are harmful but can be repaired, part of the mobile agents' task should be repairing all the faulty nodes. The agents then need to locate the black holes that are infected by the GVs.

Comparing to the traditional black hole, the timing of the presence of the GV infected black hole is finite but unpredictable. Contrary to the traditional black hole search in which all agents start in a network with one and only one back hole known a priori, in our proposed new attack model, a repaired faulty node can be infected and turned into a black hole any time while the agents traverse the network to try to repair the faulty nodes. This detail that the black hole can appear at any arbitrary time drastically changes the nature of the black hole search problem in asynchronous networks (i.e., The time that an agent takes for every action (sleep or transit) is finite but unpredictable [51].). The scenarios are significantly more complex than the traditional black hole search, especially with the presence of multiple faulty nodes which eliminate agents and consequently need to be repaired.

In addition, another difference between a faulty node and a black hole is that a faulty node can eliminate only one agent while a black hole can eliminate multiple agents. The co-existance of multiple faulty nodes and a black hole further increases the difficulty of repairing faulty nodes and black hole search. We use the following case study to illustrate why an algorithm for traditional black hole search is not

suitable in our new attack model.

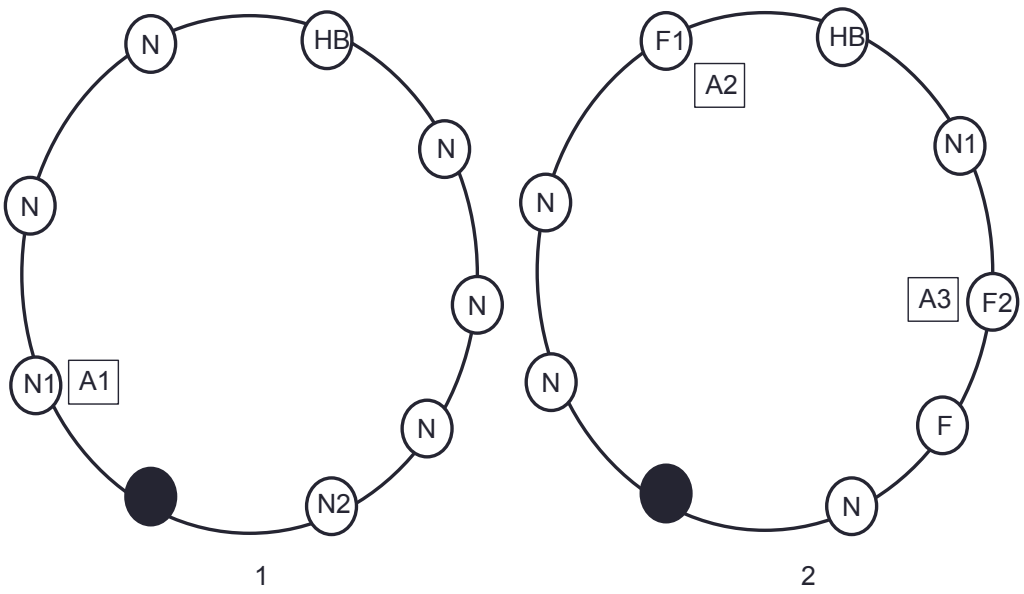

Figure 1: 1. A traditional black hole search algorithm terminates when an agent explores $n-1$ nodes (nodes N_1 to N_2). 2. No agent can pass nodes F_1 and F_2 since faulty nodes are treated as black holes, so that no agent can explore $n-1$ nodes. F: Faulty node, N: Normal node, R: Repaired node, A: Agent

To locate a black hole in traditional black hole search in asynchronous network, there is a commonly used technique called cautious walk (details in Section 2.2); a first agent has to leave a dangerous mark (a token/whiteboard messages) on the node before the black hole and enter it. When a second agent sees the mark, it will not enter the same node. This technique is used to minimize the loss of mobile agents. A traditional black hole search algorithm terminates when all nodes in the network have been explored except one, and this only unexplored one is the black hole. However, with multiple faulty nodes in the network in our new attack model, even when an agent can leave unlimited information on any node it visited (e.g., using a new technique that we introduced in Section 4.2), a traditional algorithm cannot solve the *Repair and Search* problem. This is because in traditional black hole search algorithms, there is no mechanism to distinguish a black hole from faulty nodes, thus,

all agents will treat a faulty node the same way as a black hole according to all pure black hole search algorithms. Namely, no other agent except for the one that died in it will further enter into a black hole. Since there are multiple faulty nodes that are all treated as black holes, no agent is able to explore $(n-1)$ nodes in the given network (see Figure 1). Hence even adding extra communication capability and unlimited local memory storage on each node in the network, none of the existing black hole search algorithms can locate and repair all faulty nodes as well as locate the re-infected black hole. We further use one of the best black hole search algorithms in asynchronous ring network proposed by Flocchini *et al.* [43] to illustrate this observation in Section 3.2.2.

1.3 Contributions

First of all, we have conducted an extensive literature review of the black hole search problem which was submitted to Journal of Parallel and Distributed Computing. This comprehensive literature review first includes all commonly used assumptions, classifications, and cost analysis metrics, which appears in Chapter 2. Following this in Chapter 3, we collect, classify, and analyse all papers which focus on single black hole search, multiple black hole search (MBHS), and other types of malicious hosts.

Secondly, after reviewing the existing research work, we introduce a new attack model that involves not only multiple faulty nodes in the network (a type of black hole), but also a gray virus that can again infect a previously repaired faulty node. Under such a model, the repair and search problem becomes more complex and realistic. We analyze the proposal model and highlight key observations regarding this problem. We introduce one-stop and multi-stop gray virus and study the faulty node repair and black hole search problem.

Thirdly, we propose solutions to solve the problem caused by a multi-stop gray

virus in an asynchronous arbitrary network topology using a token model. We conclude that $k > b$ agents (k is unknown a priori) are necessary and sufficient to repair and locate b faulty nodes in the network. Since the exist of a multi-stop gray virus may make the repair and search problem the same as the MBHS problem which is unsolvable without additional assumptions, we use this algorithm as an introduction of the multi-stop gray virus and identify some key observations for the impact of this virus.

Most importantly, we study the one-stop GV under the token model in an asynchronous ring network. We conclude that $b + 9$ agents can repair all faulty nodes as well as locate the black hole that is infected by the one-stop GV. Due to the nature of the dynamically spawned black hole, the *Repair and Search* problem becomes much more complex.

Lastly, after the studies using the token model, we propose a solution to solve the problem in an asynchronous ring network with only one whiteboard in the homebase node. We conclude that using our proposed algorithm, $b + 4$ agents can complete the repair and search task within finite time. Our algorithm works even when the number of faulty nodes b is unknown. We provide not only theoretical proof but also simulation results which can further support the correctness of this algorithm.

In addition, we also further pinpoint some future work on the study of black hole search, as well as for the problem of repair and search. We collect some remarkable future work mentioned in publications, and further analyze and classify some other future work by single black hole search, multiple black hole search and open problems with different types of agents. Particularly, we determine open problems with one-stop and multi-stop gray virus.

1.4 Thesis Organization

In Chapter 2, we will introduce the commonly used assumptions and complexity analysis metrics. We then have a comprehensive literature review in Chapter 3. The definitions, models, and assumptions for our algorithms are introduced in Chapter 4. We will present the algorithms for the repair and search problem using a token model in Chapter 5, and using a whiteboard model in Chapter 7.

Finally, in Chapter 8, we conclude this paper in Section 8.1 and offer future work on both single and multiple black hole search problem as well as other types of agents in Section 8.2.

Chapter 2

Background Introduction - Commonly Used Assumptions in Black Hole Search

Because none of the existing algorithms is omnipotent to solve the black hole search problem under any condition, it is crucial to collect all the assumptions that are made in the existing research and study the impact of each. In this section, we introduce a list of assumptions that are commonly used to solve the black hole search problem.

First of all, existing research assumes that the agents' initial wake-up nodes are safe; otherwise, all the agents may die before even starting the graph exploration, rendering the problem unsolvable. We further observe: unless the agents are extremely fortunate, namely, happen to explore all nodes in a graph except the black hole, in order to systematically identify a black hole, we must expect at least one agent to go in a black hole and somehow leave a hint to the other agents before it dies, which allows the other surviving agents to know the location of the black hole. All the remaining assumptions are listed in Table 2. We provide a detailed explanation of each of these assumptions in the following paragraphs.

Table 2: Assumptions that are frequently used in literature

Network synchronization	Communication model	Agent starting location	Network knowledge
Synchronous network	Pure token	Co-located	No knowledge (e.g. unknown)
Asynchronous network	Enhanced token	Dispersed	Edge-labelled (e.g. sense of direction)
	Whiteboard		Network topology (e.g. ring)
	Face-to-face		Complete knowledge (e.g. map)

2.1 Network Synchronization

2.1.1 Synchronous Network

Synchronous network means that in the network all the agents initially wake up at the same time, and it takes unitary amount of time (time unit) for an agent to traverse a link or explore a node (aka having a global clock). After each time unit, an agent must decide whether to move to a neighbouring node, to stay at a current node, or to terminate the algorithm. As such the complexity of the agent's algorithm in synchronous networks can be measured in terms of the number of time units.

In synchronous networks, a *time-out mechanism* has been introduced to enforce the time synchronization [17,23–25,59]. Such a mechanism allows us to easily identify which agents died in the black hole. Suppose a team of agents should meet at a node u after m time units, after this time-out, all other agents know that those who do not show up in node u died in the black hole.

Using the time-out mechanism, we can locate the black hole without even knowing the network size n (number of nodes) if there is only one black hole known a priori

in the network. For example, 2 agents, a and b, are at a safe node u, and agent a moves to the neighbouring node v and returns while agent b waits at node u. As each move takes 1 time unit, if agent a does not come back to node u after 2 units, then agent b knows that agent a is dead and node v is the black hole. Once agent b knows the location of the black hole, the algorithm can terminate immediately even if there are remaining unexplored nodes in the network. The same results follow when there are multiple black holes in the network. Furthermore, with this mechanism, it is also possible to know whether or not a black hole exists. More specifically, if all the n nodes in the network have been explored after a predefined time-out, we can conclude that there is no black hole in this network, providing n is known in advance. Following this observation, Klasing *et al.* [60] and Czyzowicz *et al.* [23] solve the black hole search problem by assuming that there is at most one black hole (there is one or no black holes in the network).

2.1.2 Asynchronous Network

Unlike the synchronous networks, there is no global clock mechanism in asynchronous networks. As such, the agents could initially wake up at different times. The time that an agent takes for every action (sleep or transit) is finite but unpredictable [51]. Therefore, it is impossible to distinguish whether an agent died in a black hole or is stuck in a slow link/node in the network since the latter takes an unpredictable amount of time [69]. As a result, the only way to locate a black hole in an asynchronous network is to explore the *entire* network [51]. Consequently, the network size n and the number of black holes b must be known in advance in order to count the total number of explored nodes (single or multiple black hole search); only when at least $n - b$ nodes are explored, the algorithm may terminate. For the remainder of the paper we focus on single black hole search and discuss the multiple black hole search problem separately in Section 3.3.

2.2 Communication Model

Since the location of the black hole is unknown, regardless of the network synchronization, an agent may die at any time during its exploration. As we have mentioned at the beginning of Section 2, in order to systematically identify a black hole, a team of agents are used to locate the black holes. Hence, collaboration between agents becomes necessary and essential. For collaboration, the agents are usually assumed to communicate with each other in four communication models summarized in [17, 66]. They are the *pure token model*, *enhanced token model*, *whiteboard model*, and *face-to-face model*. Communication is usually used to minimize the number of agents that died in the black hole(s). To this end, at most one agent should be allowed to enter the same node at the same time; that is, the first encountering agent should somehow inform the later ones of which node is under exploration, consequently considered *dangerous*. This will prevent other agents entering a dangerous node. More specifically, *Cautious Walk* is a commonly used technique in black hole search algorithms. This procedure is first introduced by Dovrev *et al.* [31] to minimize the number of agents that died in the black hole. At any point in time during the exploration, a port of a node can be classified as *unexplored* - no agent has ever passed through this port; *dangerous* - an agent left via this port but no agent has returned through it; and *safe* - an agent has left and returned through this port. In the cautious walk, ports are marked differently in different communication models. No agent leaves via a dangerous port. Consequently, if node v is a black hole, at most one agent will die via port p.

2.2.1 Whiteboard Model

In general, all mobile agents are invisible to each other even when they meet in the same node, let alone exchange direct information. In the *whiteboard model* introduced

by Dobrev *et al.* [31], each node has a bounded amount of storage where information can be written and read by the agents. All incoming agents can access a whiteboard in a node in a mutual exclusion way and communicate with each other via reading/writing on a whiteboard.

When executing cautious walk, an agent leaves a node u to a neighbouring node v via an unexplored port p. It marks port p dangerous by writing on the whiteboard of node u; after visiting node v, the agent immediately returns to node u to change the nodes on the whiteboard from dangerous to safe.

2.2.2 Pure Token Model

In the *pure token model*, each agent has a limited number of tokens which can be placed on or picked up at a node in the course of searching. An agent places a token at its current node to indicate that the next node is dangerous. It should be noted that the agents may need extra tokens to express this message: which node is the next node because a node may have several neighbouring nodes adjacent to it. The pure token model can be considered as a special whiteboard model with $O(1)$-bit memory on each node (assuming at the same time, only a constant number of tokens can be placed at a node). The tokens which can be picked up from a node and placed on another are defined as *movable tokens*. In contrast, Chalopin *et al.* [16] define *unmovable tokens* as those that cannot be picked up once placed on a node. If not specified, the tokens mentioned in this paper are movable and also identical by default.

2.2.3 Enhanced Token Model

Due to the limitations of the pure token model (e.g. limited number of messages that can be expressed using a constant number of pure tokens), many research efforts

[29, 38, 40] enhance the pure token model in order to increase the information carried out by the tokens. In the *enhanced token model*, the tokens can be left at not only the center of a node, but also the ports of a node. As the number of locations to hold the tokens increases, the memory footprint on each node also increases. Usually, the memory footprint is set to $O(\log n)$ bits in the whiteboard model, $O(\log \Delta)$ in the enhanced token model, and $O(1)$ in the pure token model, where n is the network size and Δ is the maximum node degree in the network graph[1] [17].

When executing cautious walk under this model, an agent marks a port as dangerous by placing a token at this port before moving to the next node. Upon its return, this agent will pick up the token to show that this port is no long dangerous. Similar to the whiteboard model, no agent will leave via a port, on which a token is left [38]. In the whiteboard model, the messages once written may be available for multiple access by different agents for a long time, before they are modified, if it happens). In token model, especially the moveable token model, in order to consume a minimal number of tokens, agents usually reuse tokens in different nodes to deliver different messages. This is why communication is usually much more complex in the token model (especially the moveable tokens) than in the whiteboard model.

2.2.4 Face-to-Face Model

Since agents may never meet in an asynchronous network due to their unpredictable moving and computing speed as well as wake up time, the *face-to-face model* is only considered in synchronous networks. In the *face-to-face model*, agents move through the network in synchronous steps and communicate only when they meet at a node [20]; no other communication method, such as whiteboard or token, is available. The three communication models mentioned above all require memories on

[1]Note, if not specified otherwise, n and Δ with the same meaning are used throughout this survey paper.

nodes. However, the face-to-face model does not require this.

2.3 Agent Starting Location

Another area of concern that significantly affects the black hole search solutions is the agent starting location. Since at least 2 agents are necessary to locate the black hole, the agents could start at the same node or different nodes.

- Co-located agents: all the agents initially wake up at the same node, and this node is referred to as *homebase*;
- Dispersed agents: the agents wake up at different nodes. The node, in which an agent wakes up is its *homebase*. Dispersed agents are also occasionally referred to as *scattered agents*. In this paper, we use the former through out.

In both cases, the homebases are assumed to be safe. Otherwise, the black hole search problem will be unsolvable.[2] Moreover, each dispersed agent only knows its own homebase. There is no communication between the dispersed agents upon waking up, which is different from the co-located case where the communication between the agents upon waking up can lead to guaranteed coordination [69].

In synchronous networks, if the face-to-face model is adopted, in order to guarantee face-to-face communication between agents, only co-located agents are used to solve the black hole search problem. This is because if the agents are dispersed there is a possibility that all will die in the black hole before they even meet.

[2] All agents would die immediately upon waking up if their homebases are black holes.

2.4 Network Knowledge

The network knowledge of the agents considerably affects both the design and complexity of the black hole search solution. This knowledge includes some if not all[3] of the following: network size, network topology, network direction, edge-labelling and sense of direction.

2.4.1 Network Size

Network size is the total number of nodes in the network (denoted by n in this paper). As mentioned before, if the agents do not know the number of nodes, the black hole search problem is unsolvable in an asynchronous network. In addition, the problem is also unsolvable in the asynchronous network if the number of black holes is not known.

2.4.2 Network Topology

Network topology is the topological structure of the graph (e.g., a ring network). Many algorithms are particularly designed for certain network topologies, for example, in [41] (see earlier version in [39]), Dobrev *et al.* provide a protocol called *shadow check* which only works on ring networks. Ring network is a fundamental network topology in the black hole search field since it can be a group of many other networks, such as torus, hypercube, and complete network.

When the agents have no topology knowledge, at least $\Delta + 1$ agents are needed in any generic solution (Δ denotes the maximal node degree) for asynchronous networks, even if the agents are given the network size and the maximum node degree [51]. If the black hole is a node with Δ degree, there are Δ ports leading to the black hole that have to be marked as dangerous ports. Since one agent dies for per dangerous

[3] An unrealistic case where the black hole search problem becomes much less complex.

port marking and at least one agent has to survive to eventually report the black hole location, at least $\Delta + 1$ agents are necessary. However, this situation is different in synchronous networks. With the time-out mechanism, only 2 agents are sufficient.

2.4.3 Network Direction

Network direction refers to whether a graph is directed or undirected (e.g. bi-directional). Although the results of exploration of directed graphs emerged since 1990s (e.g. [11, 12, 52]), the first study dealing with the black hole search was published by Czyzowicz *et al.* [22] in 2010. Most commonly used techniques such as the previously mentioned cautious walk can only be used in undirected graphs. As for the directed graphs, the cautious walk technique does not apply due to the uni-directional links. Hence, some agents have to enter potentially dangerous links, resulting in the need for as many as 2^d agents [22] (d denotes the indegree of the black hole node). Unless specified, the network graphs studied in this paper are mainly undirected. [22, 61, 62] are three papers that study the black hole search problem in directed graphs.

2.4.4 Edge-labelling and Sense of Direction

An *edge-labelled graph* is one where at each node x, there is a distinct label associated with each one of its ports and the incident link of each port. Let $\lambda_x(x, z)$ denote the label associated at x with the link $(x, z) \in E$, and λ_x denote the overall injective mapping at x. The set $\lambda = \{\lambda_x | x \in V\}$ of those mappings is called a *labelling* and we shall denote by (G, λ) the resulting edge-labelled graph. The nodes of G can be anonymous (e.g., without unique names) [36]. When visiting a node in an edge-labelled network, an agent can distinguish the ports in this node. However, this is not the case in an edge unlabelled network.

Table 3: Relationships between network direction and sense of direction

<table>
<tr><td rowspan="4">Directed Graph</td><td colspan="3">Undirected Graph</td></tr>
<tr><td rowspan="3">Edge Unlabelled</td><td colspan="2">Edge-labelled</td></tr>
<tr><td>Arbitrariely Labelled</td><td colspan="2">Consistently Labelled</td></tr>
<tr><td>Un-oriented, No Sense of Direction</td><td>Oriented, Sense of Direction</td><td>Un-oriented, No Sense of Direction</td></tr>
</table>

Sense of direction applies when in an edge-labelled undirected graph, if from any given node u, it is possible to determine whether or not different paths from node u will end in the same node. More precisely, in order to obtain the sense of direction, a *consistent coding function* and a *consistent decoding function* should be defined in the system [50]. For example in a ring network, the ring is considered to have a sense of direction when all the ports are consistently labelled as *left* and *right* (i.e., all ports going in the clockwise direction are labelled $left$) [16]; a ring with such a sense of direction is called an *oriented ring*. If the labelling of the two ports at a node in this ring is arbitrary, namely, there is no common understanding of left and right, we say that there is no sense of direction in this ring which is called an *un-oriented ring*.

We further clarify the relationships between the network direction and the sense of direction in Table 3.

2.4.5 Complete Knowledge

Complete knowledge is defined as that the agents know the network size, the topology type and sense of direction (e.g., torus with consistent and systematic "N-S-E-W" labelling). Sometimes, agents are equipped with a network map (beyond containing all the network knowledge, this map can also be used to mark the explored nodes during a black hole search.) [28]. In this situation, the black hole search problem

becomes much less complex, but it is also in fact a less unrealistic model.

2.5 Commonly used Cost Analysis Metrics

In order to compare different black hole search solutions, researchers conduct complexity analyses on the costs of their solutions. The following are the most frequently measured costs:

- Number of agents: the minimal number of agents used to solve the black hole search problem.
- Number of agent moves: the total number of moves performed by all agents from wake up till the black hole is located.
- Number of tokens: the minimal number of tokens used by each agent or the entire agent team in order to locate the black hole.
- Memory footprint of agents: the memory overhead of agents. Usually, in the token models the agents are designed with a small memory footprint (e.g., an agent can only carry a constant number of tokens at any point in time [15, 16]), while some other agents may have a very large memory footprint (e.g., the agents can carry a network map [43, 59]).
- Memory footprint of nodes: the memory overhead of each node in the network. For example, a $O(\log n)$-bits whiteboard is sufficient for all the algorithms in [7, 27]. Similarly, the black hole search under the pure token model may need $O(1)$-bit to use tokens [16]. However, when considering token models, it make more sense to measure the number of tokens rather than the memory footprint of nodes. Thus, memory overhead is mainly considered in the whiteboard model.
- Time cost: the time cost in synchronous networks is the total number of time units used from when algorithm starts until the black hole is found. In an asynchronous network, a move of each agent costs finite but unpredictable time.

Therefore it is impossible to measure time. However, some research [6, 7, 31] assumes a unitary time delay to each move, which enables the calculation of time complexity. Such a measure is referred to as *ideal time*. Under this assumption, time cost is almost the same as the number of agent moves.

Beyond measures of complexity, correctness evaluations are also a common component of black hole search papers. Most papers use mathematical proof [7, 15, 23, 29, 44, 45, 60], while only a few conduct simulations and use the experiment results to demonstrate the connectedness [26, 69]. Shi *et al.* [69] present their simulation results for three proposed algorithms after theoretical proofs. D'Emidio *et al.* [26] simulate and compare their own algorithms, and further analyse which one performs better.

Chapter 3

Literature Review

3.1 Black Hole Search in Synchronous Networks

Since no one used the enhanced token or the whiteboard model in synchronous networks, in this section, we overview the black hole search solutions in synchronous networks based on the agent communication models, the agent starting locations, as well as the network knowledge. We organize the research studies into three subsections as shown in Figure 2.

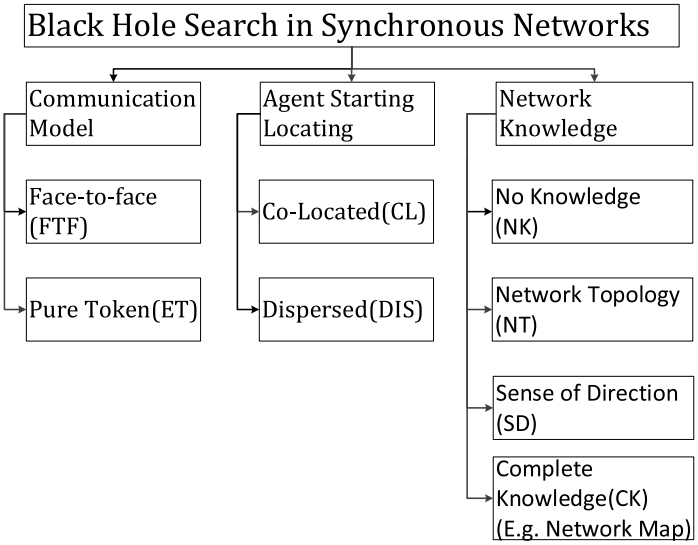

Figure 2: The organization structure of black hole search in synchronous networks

3.1.1 Communication Models

Face-to-Face Model

The face-to-face model is only possible in synchronous networks. In this model, the agents that are in the same node can communicate an unlimited amount of messages. Using the face-to-face model associated with the time-out mechanism, the black hole can be located in any network with just 2 co-located agents (as described in Section 2.1.1). [23–25,58–60] all consider the problem of finding the most efficient (in terms of time cost) solution for the black hole search under the same assumption: 2 co-located agents searching for a black hole in an edge-labelled undirected synchronous network. Chalopin *et al.* [16] study the problem using a hybrid communication model, namely, agents can carry and place a bounded number of pure tokens and can communicate with each other when they meet on a node. Since [16] focuses more on the impact of the tokens, we discuss the detail of this study in the section on the pure token model (Section 3.1.1).

Czyzowicz *et al.* [24] show the aforementioned optimal black hole search problem is NP-hard, and propose a 9.3-approximation algorithm for it. Klasing *et al.* [60] prove that this problem is not polynomial-time approximation within any constant factor less than $\frac{389}{388}$ (unless P=NP), and give a 6-approximation algorithm. In both [60] and [24], each agent carries a network map and starts from the same node. The difference is that the algorithm proposed by [24] can solve the problem when there is one and only one black hole in the network while the solution in [60] can detect whether there is a black hole (note, as previously mentioned, this is only possible in a synchronous network) and locate the black hole when present.

In [23, 25], Czyzowicz *et al.* present a $\frac{5}{3}$-approximation algorithm in an arbitrary tree without a map. This result demonstrates the impact of network knowledge: the knowledge of specific network topology reduces not only the time complexity but

also the memory footprint of each agent. The authors introduce algorithms for two "extreme" classes of trees; one is the class of lines and the other is the class of trees in which all internal nodes have at least 2 children. The algorithm in [59] follows a natural approach of exploring the network graph via a spanning tree. Later, Klasing *et al.* [59] probe that this approach cannot lead to an approximation ratio bound better than $\frac{3}{2}$. Furthermore, [59] provides a $3\frac{3}{8}$-approximation algorithm for an arbitrary network with the help of a network map. This result is a direct improvement from the $\frac{7}{2}-$ approximation algorithm presented in [58].

Pure Token Model

Chalopin *et al.* [15, 16] ([16] see a full version in [17]) and Markou *et al.* [66] focus on locating the black hole using a minimum number of agents and tokens, while the agents have $O(1)$ memory size and carry $O(1)$ pure tokens. Most importantly, the goal is achieved without the agents knowing n or k, where n is the number of nodes in the network and k is the number of agents. The authors consider both movable and unmovable tokens in ring [16] and torus [15, 66] respectively.

Chalopin *et al.* [16] consider the black hole search problem with agents that have hybrid communication capabilities: they can communicate with each other face-to-face when they are in the same node and can also carry either movable or unmoveable tokens. When using the movable tokens, 3 agents, each of which carries only 1 token, are necessary and sufficient for both oriented and un-oriented rings. In contrast, using unmovable tokens needs 4 agents, each with 2 tokens, for oriented rings and 5 agents, each with 2 tokens, when exploring un-oriented rings.

Expressing messages using unmoveable tokens is equivalent to writing messages on whiteboards with constant memory. At first glance, such a model should be more powerful (as the whiteboard model) than the moveable token model. Interestingly, the results show that using unmovable tokens is more costly than that using movable

tokens in terms of the number of agents and the number of tokens. In addition, more agents are necessary for un-oriented rings than for oriented rings.

In addition to rings, Chalopin *et al.* [15] also study the oriented torus under the same assumptions: dispersed agents, pure token model, and face-to-face communication. They prove that the black hole search problem is unsolvable in torus in three scenarios: 1) when the number of agents and unmovable tokens are constant in an oriented torus; 2) when using 2 dispersed agents in any synchronous torus, even if the tokens are movable and the agents have unlimited memory; 3) when using 3 agents with constant memory and 1 movable token each. Finally, they show that at least 3 agents, each with 2 movable tokens, are necessary and sufficient to solve the problem in any oriented torus.

Markou *et al.* study the black hole search problem in [66] under the same assumptions as [15] but in an un-oriented torus. The authors discuss four cases of un-oriented tori, from totally un-oriented to semi-oriented (without an agreement on the orientation in the horizontal or vertical axis). The authors prove that constant number of agents and tokens cannot solve the black hole search problem in all un-oriented tori using unmovable tokens. The authors further discussed the use of movable tokens. They prove that the problem is also unsolvable when using any constant number of dispersed agents with 1 movable token each. The authors then offer algorithms with 5 agents and 3 tokens each in all semi-oriented tori. Finally, they conjecture that at least 5 agents and at least 2 movable tokens each would be able to locate the black hole in a totally un-oriented torus. However, formal proofs of the correctness and complexity are not provided.

3.1.2 Agent Starting Locations

When all agents wake up in the same node, the coordination and communication are guaranteed to these co-located agents. This renders the graph exploration much

easier. Some researchers [23–25, 58–60] choose to study the problem using co-located agents, others choose to use the dispersed agents [15–17, 66].

Co-located Agents

All existing papers that adopt the face-to-face communication model also unanimously choose to use the co-located agents. This is because it is possible that all the dispersed agents may die before they even meet each other.

Since it is proven that 2 co-located synchronous agents are sufficient, all of the following studies [23–25, 58–60] focus on finding algorithms for the black hole search problem (see details in Section 3.1.1) to improve the time cost. It is important to note that with only 2 co-located agents, whether the agents are anonymous or not is irrelevant here as one agent can definitely distinguish the other when they meet.

Dispersed Agents

To extend the results of 2 co-located synchronous agents, Chalopin *et al.* [15–17] and Markou *et al.* [66] study the problem using dispersed agents under the pure token model (see details in Section 3.1.1). Contrary to the case of co-located agents, all these papers focus on the minimal number of dispersed agents and tokens they use, rather than the time complexity and agent moves. Beyond assuming that the network size is unknown a priori, another challenge here is that the pure tokens are used for the agents to communicate. Since tokens can be placed only at a node, not its ports, this makes the coordination among the team of dispersed agents much more complex. For example, in a torus, even when an agent sees a token at a node, it still cannot know from which node the previous agent left.

3.1.3 Network Knowledge

Further to the previous discussion, network topology is another component that significantly affects the algorithms and complexities. From the previous mentioned papers that study rings [16, 17] and study torus [15, 66], it is obvious that more agents and tokens are needed in torus than rings under the same assumption. Therefore, network topology not only affects the complexity of the network, but also affects the number of agents and the number of tokens necessary and sufficient to solve the black hole search problem. Another instance is that with a map of an arbitrary network, [59] gives a $3\frac{3}{8}$-approximation algorithm, when [25] gives a $\frac{5}{3}$-approximation algorithm without a map but knowing the topology is a tree. It is clear here that knowing the topology makes the algorithm much faster than having a map but not knowing the topology.

A sense of direction is an additional property adding to an edge-labelled graph. Without edge-labelling, an agent could not distinguish the edges incident to a node, and thus a whole part of the graph could be unreachable during an exploration.

Since the sense of direction offers not only a consistent edge-labelling, but also a guaranteed method of systematic exploration of the entire graph, studies under the same assumptions, other than the existence of a sense of direction, show a great difference. For example, compared to the study on un-oriented ring and torus networks in [16, 17, 66], results from oriented ring and torus networks in [15–17] demonstrate the power of a sense of direction.

In [66], Markou *et al.* discuss their solutions and results under different network knowledge of the given torus: 1) the agents have no agreement on anything regarding the orientation; 2) the agents perceive orthogonal links but they do not agree on which link; 3) the agents agree which link is horizontal and which is vertical, but there is no consensus on the orientation of each link; and 4) the agents agree which

link is horizontal and which is vertical and they also agree on the orientation in one of the links. Except for 1), all the other three types are called *semi-oriented*.

When the tokens are unmoveable, it cost 5 agents to locate the black hole in un-oriented rings rather than 4 agents in oriented ones. When the tokens are movable, with 3 agents and 2 tokens each, the problem can be solved in all oriented tori; while using 5 agents, each with 3 tokens, the problem is only solved in semi-oriented tori. It is clear that un-oriented networks need more agents and tokens to solve the black hole search problem than oriented networks.

3.2 Black Hole Search in Asynchronous Networks

Unlike the case in the synchronous network, the black hole search problem in asynchronous network is much more complex and more significant in practice. In this section, we overview the state of the art in the asynchronous networks based on the agent communication models, the agent starting locations, as well as the topological knowledge (see summary in Figure 3).

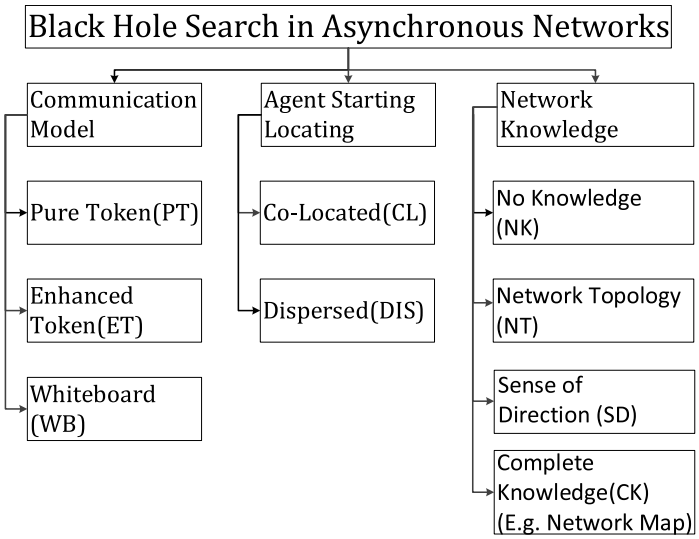

Figure 3: The structure of black hole search in asynchronous networks

3.2.1 Communication Models

In the asynchronous networks, agents may wake up at different times and die in the black hole before they even meet each other. Thus, face-to-face communication is not of great use to solve the black hole search problem. As such, we overview the results in the token and whiteboard models.

Pure Token Model

Flocchini *et al.* [43] (see full version [44]) first prove that the pure token model is as powerful as the whiteboard model and maintains the same complexity with the whiteboard model in an arbitrary network if each of the co-located agents carries a map. Flocchini *et al.* also show that 2 co-located agents, each with 1 token, can locate the black hole in a ring topology using a technique called *ping-pong*. In this specific case, when the network topology is known, the agents can achieve the goal without using a map. Additionally, they further demonstrate that this ping-pong technique can also be applied to an arbitrary network if a corresponding network map is available to each agent. In the latter case, it costs $\Theta(n \log n)$ moves to locate the black hole.

It is well known that $\Delta + 1$ agents are necessary to locate the black hole when the topology of an asynchronous network is unknown regardless of the number of tokens used. With the same number of agents and $O(1)$ tokens in total, it is sufficient to locate the black hole when each agent has a network map available. Balamohan *et al.* study whether $\Delta + 1$ agents, each with $O(1)$ tokens can still locate the black hole in an unknown network in [5]. They prove that in order to keep the total number of tokens used to $O(1)$, $\Delta + 1$ agents are not sufficient. They thus present a protocol that uses $\Delta + 2$ agents, each of which carrys 3 tokens to locate a black in an unknown network. It is important to note that the black hole search in asynchronous networks

are studied only in the co-located agents case when pure tokens are used.

Enhanced Token Model

Due to the limitations of the pure token model, Dobrev *et al.* [29, 38, 40, 41] and Shi *et al.* [68] use the enhanced token model to further improve the move and agent cost. In all these studies, each agent can carry and most importantly can place in the same node more than 1 token at any time. With these characteristics, Dobrev *et al.* [39, 41] introduce an algorithm to locate the black hole in an un-oriented ring network with dispersed agents. Obviously, coordinating dispersed agents is significantly more complex than using co-located agents. The proposed algorithm demonstrates that using $O(1)$ enhanced tokens enables the black hole search in asynchronous networks using dispersed agents. In their paper [38], Dobrev *et al.* demonstrate that the move cost of $O(kn + n \log n)$ [39, 41] can be reduced to $O(n \log n)$ by using 2 co-located agents with $O(1)$ tokens per agent, when the orientation of the ring is known.

Apart from the ring networks, Shi *et al.* [68] (see full version in [69]) prove that 2 co-located agents, each with $O(1)$ tokens, can locate the black hole in $\Theta(n)$ moves for hypercube, torus and complete networks. While using dispersed agents, 3 agents and 7 tokens in total, a black hole can be located within $\Theta(n)$ moves in an oriented torus. When the number of agents increases to $k(k > 3)$ and 1 token per agent, the moves become $O(k^2n^2)$. This result is interesting. It shows that if the number of dispersed agents in a torus increases, the communication between these agents becomes significantly more complicated. This is reflected in the increase of the move cost.

Moreover, for an arbitrary unknown network graph with known n, Dobrev *et al.* [29] present an algorithm using $\Delta+1$ agents and one token per agent and $O(\Delta^2M^2n^7)$ moves to locate the black hole. Here M is the total number of edges of the graph. This result has been improved by the same authors in [30] to $O(\Delta^2M^2n^5)$ moves.

In contrast, under the same assumption in the whiteboard model, the cost of the algorithm is $\Delta + 1$ agents and $\Theta(n^2)$ moves. For arbitrary unknown network graphs, the cost complexities of the enhanced token model are significantly greater than those of the whiteboard model [29]. However, when the network maps are available to the agents, the complexities of the enhanced token model can be reduced to the same cost as in the whiteboard model [38].

Whiteboard Model

In both types of token models, agents can only express very limited messages. This is why the whiteboard model is still the most popular agent communication model and has been studied by [6, 7, 22, 27, 28, 31–33, 35–37, 53].

In addition to presenting solutions to the black hole search in asynchronous arbitrary networks [32, 36, 37], Dobrev *et al.* [33] solve a multiple agents rendezvous problem in spite of a black hole in a ring network. In their paper, the final goal of the agents is not only to locate the black hole but also to collect all survived dispersed agents in one node. The authors offer a protocol that can rendezvous k agents in $\Theta(n)$ time units. When k is unknown, this protocol is also a solution to the black hole search problem. In terms of the time complexity in rings, Dobrev *et al.* [31, 35] show that at least $2n - 4$ time units (assuming the unitary traversing and exploring time unit) are needed in the worst case and give an algorithm, achieving it by $n - 1$ co-located agents. Apart from the time complexity, the authors also prove that 2 agents are necessary and sufficient and present an algorithm to locate the black hole in $O(n \log n)$ moves, regardless whether the agents are co-located or dispersed, but the orientation of the ring must be known a priori. If the ring is un-oriented, 3 dispersed agents are necessary and sufficient. Apart from rings, Dobrev *et al.* [27] (full version [28]) also present a general strategy to locate the black hole in $O(n)$ moves by using 2 co-located agents for some other common interconnected networks, such

as *cube-connected cycles, wrapped butterflies, star graphs, chordal rings, hypercubes, tori of restricted diameter*, and in *multidimensional meshes.*

Based on Dobrev's work, Balamohan *et al.* [6] prove that $3n \log_3 n - O(n)$ moves are necessary in an asynchronous ring when 2 co-located agents are used. As for the time complexity, Balamohan*et al.* [7] improve the algorithm of [31] to solve the problem in an average of $\frac{7}{4}n - O(1)$ time units when $n-1$ agents are used and 2 extra time units are required in the worst case. The authors also propose another algorithm to locate the black hole in $\frac{3}{2}n - O(1)$ time units on average, using $2(n-1)$ agents without increasing the time complexity in the worst case.

While all the above studies only consider the case of undirected graphs, Czyzowicz *et al.* [22] study the black hole search in directed graphs. They show that at least 2^d agents are necessary in the worst case, where d is the indegree of the black hole. If a planar graph with a planar embedding is known to the agents, $2d$ agents are needed, and $2d+1$ agents are sufficient.

3.2.2 Agent Starting Locations

As we have discussed in the synchronous networks, when the homebases of the agents are dispersed, the black hole search problem is more complex compared to the cases when all agents wake up in the same node. This is even more so in the asynchronous network case. Unlike synchronous networks when all agents may wake up at different times, coordinating all such agents in order to locate the black hole with minimal resource cost is a challenge. For example, 2 co-located agents suffice to solve the problem in a complete network in $\Theta(n)$ moves in [69]; while using dispersed agents costs $O(n^2)$ moves.

Co-located Agents

The co-located agent model is frequently used in the literature. Among those white-board based studies, [6, 7, 22, 28, 31, 32, 36, 37, 53] adopt this model. Similarly, for the token-based research, [5, 29, 30, 38, 43, 44, 68, 69] also choose to solve the problem under this model. Among the papers, [6, 7, 31, 38, 43, 44] delve into the ring networks, while [6, 7, 31] study time complexity. [7] offers an algorithm which improves the average time from [31]. Moreover, [38, 43, 44] only use 2 agents, and [38] studies the enhanced token model, while [43, 44] investigate the pure token model.

When the agents are initially co-located, they can establish many agreements before the exploration. This can greatly help the coordination between the agents and eventually reduce the resource costs. For example, in a ring network, when the agents are co-located, the orientation is no longer important. This is because when there are only two directions, the agents can certainly make an agreement at the beginning of the exploration on what is right and left. Furthermore, solving the problem using co-located agents in a ring without network maps is the same as having each agent carry a network map. The situation is different while using dispersed agents; they are not equivalent unless the orientation of the ring is known.

The following example described in [43, 44] illustrates how a pair of co-located agents can locate the black hole using such an agreement: 2 agents each with one token start to explore the ring using cautious walk; one goes right and the other goes left. However, only one agent at a time is allowed to explore. To ensure this, one agent must first steal the token from the other before its own exploration as stealing is possible because, during cautious walk, a token has to be placed before the agent goes to the next node. After stealing, the agent without a token cannot continue exploration and has to go back to look for a token, then keep repeating until one agent dies. For example, suppose the right agent goes first. Before the left agent

starts, it first goes right and steals the token of the right agent, and then goes left for exploring. Once the right agent finds its token has gone, it goes left and steals a token from the left agent, and then goes right again. Repeating this process can ensure that only one agent dies in the black hole, and the surviving one knows the location of the dead agent.

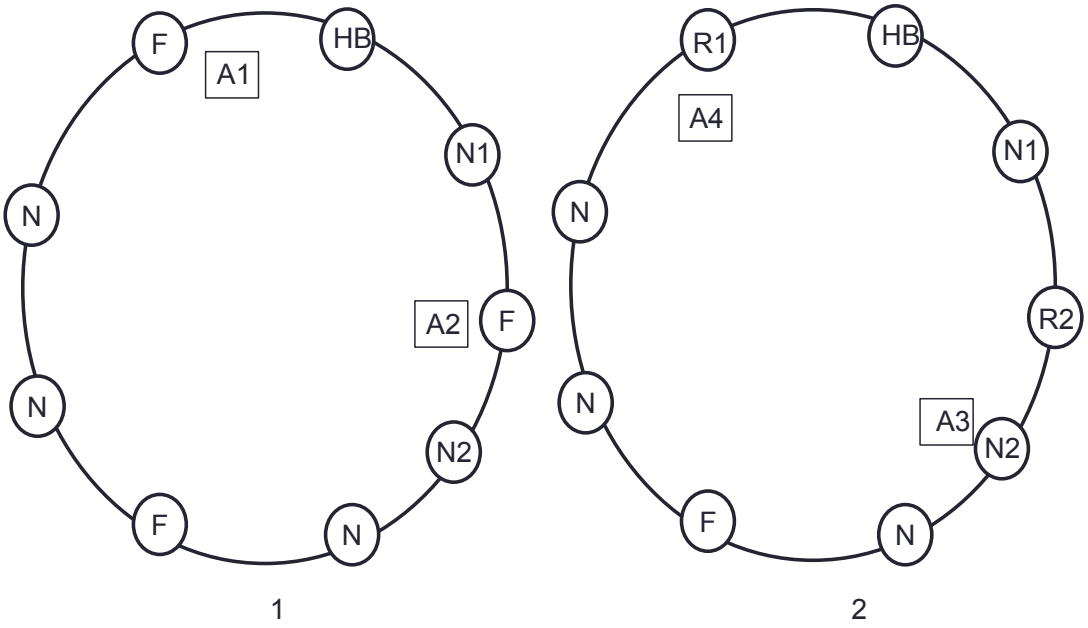

Figure 4: When using the *ping-pong* technique, after agents A_1 and A_2 both die in faulty nodes, this algorithm stops (Picture 1). If 2 more agents A_3 and A_4 enter the ring, this technique is stacked again when A_4 dies in R_2 (Picture 2). F: Faulty node, N: Normal node, R: Repaired node, A: Agent

For the study of our new attack model, Figure 4 shows an instance of using the above mentioned *ping-pong* technique [43,44] to solve the *Repair and Search* problem. Agents A_1 and A_2 leave the homebase in 2 directions and they have to steal each other's token after each move to make sure there is only one agent entering the black hole. This suffices in the traditional black hole search where there is only one harmful node, however, with multiple faulty nodes in our new attack model, both agents A_1 and A_2 may enter faulty nodes, thus, this algorithm is stacked when the 2 agents

both die.

Another issue of the *ping-pong* technique is that there is no method to distinguish a black hole from a faulty node, even with our new technique double cautious walk (in Section 4.2). Regardless the number of tokens an agent can leave on a node or a port, since an agent will not enter a node which is marked as dangerous, this node will block all the later agents. Therefore, there will be no agents passing nodes R_1 or R_2 (Picture 2 in Figure 4), so that no agents can explore the rest of the network.

Even if more agents A_3 and A_4 continue to explore the network (not possible when using *ping-pong* technique as the tokens of A_1 and A_2 will block A_3 and A_4. Assuming with some changes to the algorithm, these agents can pass R_1 and R_2), it is possible that when A_4 moves to steal A_3's token, the previously faulty node R_2 which repaired by A_2 turns in to a black hole and deletes A_4. Since no agent will steal A_3's token after A_4 dies, A_3 has to wait at node N_2 permanently. This again prove that a traditional black hole search algorithm cannot solve the *Repair and Search* problem, even with the double cautious walk technique, whiteboard on each node, or minor changes to the algorithm.

Dispersed Agents

The dispersed agents have been adopted by the research based either on the whiteboard model [31,33,35,53] or on the enhanced token model [40,41,68,69]. So far, no one has attempted to solve the problem using dispersed agents carrying pure tokens. The reason for this might be that it is unlikely to achieve the goal using the same or fewer pure tokens than the enhanced tokens when the agents are dispersed.

Glaus *et al.* [53] study the black hole search problem without the knowledge of incoming links in an un-oriented unknown graph using the whiteboard model, while all other research assumes that when an agent enters a new node, the agent automatically knows through which port it enters.

Both Shi *et al.* [68] and Dobrev *et al.* [31] study the problem using both co-located and dispersed agents. While [68] considers agent moves in *hypercube, torus*, and *complete networks*, [31] measures agent moves and time complexity in ring networks.

Dobrev *et al.* [40] solve the black hole search problem using an algorithm called *Pair Elimination* in oriented ring networks. The agents are initially dispersed in the ring and each endowed with $O(1)$ enhanced tokens. This algorithm is to let all the agents try to form pairs as soon as they wake up. All the paired agents will eliminate all the single agents they meet. Each pair has a level. A pair increases its level each time it eliminates another agent. When two pairs meet, the higher level pair always eliminates the lower level pair. Between pairs of the same level, the right pair eliminates the left pair. Eventually only one pair will survive, and one of the two agents forming that pair will locate the black hole. Compared to the co-located case that each agent carries only 1 pure token, pair elimination requires 4 tokens for each agent even when they use the enhanced token model in the dispersed case. This is because the communication/coordination among dispersed agents is significantly more complex than the co-located case.

3.2.3 Network Knowledge

Most research efforts (e.g. [7, 38, 40, 43, 44]) also assume the agents have *knowledge of incoming links*, which means that when an agent enters a node, the information on which port leads back can be given to it. However, Glaus *et al.* [53] study arbitrary, unknown distributed systems without the knowledge of incoming links, and present a lower bound on the size of the optimal solution, showing that at least $\frac{d^2+d}{2} + 1$ co-located agents are necessary and sufficient to locate the black hole. Here d denotes the number of links leading into the black hole (node degree of the black hole).

In an un-oriented network, all ports which lead to a black hole should be marked as dangerous, hence $\Delta + 1$ agents are necessary. However, in an oriented network,

the number of agents which are dead in the black hole can be reduced by limiting the agents to only entering a node in certain directions. For example, as the ports of a torus are labelled as *north, south, east*, and *west*, Shi *et al.* [69] assume an agent can only enter a node from the west and come out from the east, or enter from the north and come out from the south. With this assumption, only 3 agents are necessary. However, when the agents are allowed to enter a node from all four directions, at least 5 agents are necessary.

Dobrev *et al.* [34] prove that without any knowledge, $\Delta + 1$ agents are needed and the cost is $\Theta(n^2)$. However, with the sense of direction but lack of information of the network topology, only 2 agents are sufficient to maintain the same cost. The main idea of the algorithm is described as follows: the two agents start from the homebase *hb* and construct a spanning tree of the explored nodes (visited by one agent) at *hb*; an agent searches the tree, and if there is a node with unexplored ports, the agent goes to explore the node and makes all ports explored using cautious walk; after this, it comes back to *hb* and adds the node to the tree as an explored node; before the agent leaves the homebase, it leaves a navigational instruction for the other agent; when the number of explored nodes becomes $n - 1$, the algorithm terminates.

We also observe that the knowledge of the network topology (e.g., ring, hypercube, torus, complete, tree and arbitrary networks) has great impact on the black hole search results. Balamohan *et al.* [6, 7], Chalopin *et al.* [16] and Dobrev *et al.* [31, 35, 38, 40] propose algorithms based on ring networks, while [5, 27, 28, 32, 34, 37, 43, 44] search the black hole in arbitrary networks. In particular, Shi *et al.* [69] design algorithms for hypercube and torus networks with co-located agents, and for torus and complete networks with dispersed agents. In contrast, Dobrev *et al.* [27] (full version [28]) present a general strategy that allows 2 agents to locate the black hole with $O(n)$ moves in some common interconnected networks.

In an arbitrary network, Dobrev *et al.* [32,34] prove that: in the whiteboard model,

the black hole search problem can be solved with $\Delta+1$ agents in $\Theta(n^2)$ moves without network maps; this move complexity can be kept by using only 2 agents when there is a sense of direction. With the complete knowledge of the network, 2 agents are sufficient and the cost can be reduced to $\Theta(n \log n)$. In their other paper [36], Dobrev *et al.* present a universal protocol that locate the black hole using at most $O(n + d \log d)$ moves with 2 agents each carrying a network map. Here d is the diameter of the network. Still using 2 agents, the same authors [37] present a strategy which can locate the black hole in $O(\Sigma_{i=1}^{k}|C_i| \log |C_i|)$ moves, here $C = C_1, C_2, ..., C_i..., C_k$ is an open vertex cover by cycles of a 2-connected graph. [1] These results show that having a network map or the sense of direction can significantly reduce the cost complexity.

3.3 Multiple Black Hole Search

Most existing papers study the single black hole search problem. This is due to the nature of the black holes, that is, they delete any incoming agents without leaving any observable trace. Therefore, if a network contains more than one black hole, the network may be disconnected. For example, Figure 5 illustrates a ring network containing 3 black holes that disconnected the ring into 3 sub-graphs. Unless there are enough agents starting at the desired locations, locating multiple black holes cannot be guaranteed. In order to find multiple black holes, there are roughly three different ways, each with different assumptions and capabilities:

3.3.1 Best Effort Approach

The best effort approach tries to find as many black holes as possible, if not all. In a synchronous network, as we discussed earlier in Section 2.1, finding out whether

[1] An open vertex cover by cycles (C) is defined as a set of simple cycles that each vertex of G is covered by a cycle from C and the connectivity graph of these cycles (where each cycle is represented by a vertex, and 2 vertices are connected if the corresponding cycles share an edge) is connected.

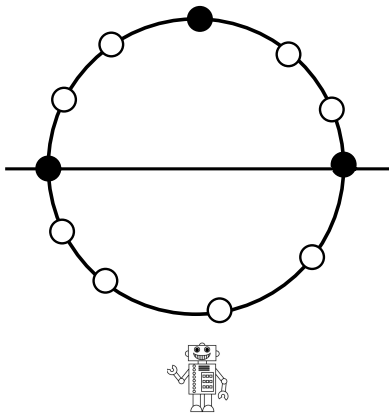

Figure 5: A ring network that is disconnected by multiple black holes (solid circles represent black holes). Consequently, an agent cannot explore the disconnected portion of the original ring network.

there is a black hole is rather trivial due to the time-out mechanism. When a network contains more than one black hole, the network may be disconnected. As such, some nodes may never be explored. In this case, finding all the black holes is impossible. Alternatively, Cooper *et al.* offer a solution to finding all the possible black holes (observe that a node can be identified as a black hole or as a safe node only if it can be reached following a path of safe nodes).

Cooper *et al.* [20] start studying the multiple black hole search problem in synchronous networks using the face-to-face model. The authors tackle the problem assuming that k co-located agents know the topology of the whole network including the size n and number of black holes b. They conclude that any exploration algorithm needs $\Omega(n/k+D_b)$ steps in the worst case to solve a multiple black hole search problem, while D_b is the diameter of the network with at most b nodes deleted. Cooper *et al.* provide a general algorithm which performs the exploration in $O(\frac{n}{k}\log n + bD_b)$ steps in an arbitrary network with network maps available to the agents, where $b \leqslant k/2$. In the case when $b \leqslant k/2$, $bD_b = O(\sqrt{n})$ and $k = O(\sqrt{n})$, Cooper *et al.* give a refined algorithm which performs the exploration in asymptotically optimal $O(n/k)$ steps.

3.3.2 Variants of Black Hole Search

In a regular black hole search, the existence of a black hole is persistent. Namely, it will not be affected by the arrival of any incoming agent. However, Cooper *et al.* [21] solve the multiple black hole search problem by changing the behaviour of the regular black hole. A *faulty node* is a weaker black hole which can be repaired by the first encountering agent, and once the fault has been repaired, the node will permanently behave as a normal one. Hence, when a network contains more than one faulty node, the agents are still able to explore the whole graph. If more than one agent enters the same faulty node at the same time, only one will repair the faulty node and die while the others can continue their explorations.

The agents used in [21] know the topology of the whole network, move synchronously, use the face-to-face model, and are initially co-located at the same node. With a network map, first, the whole network is divided into some equal parts of size $O(D)$, where D is the diameter of the network. Therefore, an agent should spend $O(D)$ time to explore one part. Moreover, all the agents start from the same homebase, and each agent explores one part. After $O(D)$ time, if an agent returns to the homebase, it implies that the explored portion of the graph contains no faulty node. However, if one agent does not show up on time, it is assumed to be dead in a faulty node, and its explored part is still marked as unsafe and needs further exploration. When all surviving agents come back to the homebase, they will start to explore the remaining parts of the network until there is no unsafe part. Eventually, the faulty node repair problem can be solved within $O(\frac{n}{k} + \frac{D \log f}{\log \log f})$ time steps, where $f = min(\frac{n}{k}, \frac{n}{D})$, assuming that the number of faulty nodes is at most $k/2$. However, in [21], because the face-to-face model leaves no mark on the nodes, once an agent dies, the other agents cannot know where it died. Therefore, after the repairing algorithm, the agents can only repair all the faulty nodes but are not able to locate the

repaired nodes.

D'Emidio *et al.* [26] study the same problem under the same condition as [21] with the change of one assumption: if more than one agent enters the same faulty node at the same time, all agents die. To make the problem more realistic, the authors consider another scenario by introducing a new number r: if one agent enters a faulty node u, all agents within distance r from u will disappear along with the faulty node. D'Emidio *et al.* first prove that the faulty node repair problem is NP-hard even when $b = k = 1$, where b is the number of faulty nodes and k is the number of agents. Second, when $r = 0$ which means the agents die only when they physically enter a faulty node, using a simple variation of the algorithm described above, the faulty node repair problem can be solved in $\Theta(\frac{n}{k-b} + \frac{D \log f}{\log \log f})$, and $k > b$ must be true in any condition. Otherwise, all agents will die. Third, for any $r > 0$, the faulty node repair problem requires $\Omega(n)$ time steps in the worst case. Fourth, when $r = 1$, the faulty node repair problem can be solved in $\Theta(n)$ time steps, and the authors provide two strategies for this bound. Finally, the authors report their experimental results to show their correctness.

Flocchini *et al.* [45, 48] solve the multiple black hole search problem via a *subway model* using co-located agents with the whiteboard model, and the number b of black holes is known to the agents. The authors use carriers (the subway trains) to transport agents (the passengers) from node to node (subway stops), and the carriers move asynchronously in a directed graph. When a carrier enters a node, the agents can either get off from the carrier and explore the node, or stay on the carrier to go to another node. In a regular black hole search, any incoming data will be deleted including the carrier. However, in this subway model, the black holes no longer affect the carriers and can only eliminate the agents. At the homebase, there is a white board that is used to record all explored, unexplored and dangerous nodes. Initially, all nodes are recorded as unexplored except the homebase. Once an agent chooses

to explore a node, the node will be marked dangerous until the agent comes back and marks it as explored. Finally, except for b nodes, all the other nodes have been explored, the algorithm terminates and the b dangerous nodes are the black holes. In [45, 48], when $k = r + 1$ agents are used (here r is the number of carrier stops at black holes), the number of carrier moves is $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$. Here n_C is the number of subway trains, and l_R is the length of the subway route with the most stops.

Under the same assumption and keeping the same carrier moves as [45, 48] , Flocchini *et al.* [47] solve the same problem with dispersed agents. Instead of having a whiteboard at the homebase the authors put the whiteboard on the carriers, thus, an agent only has to come back to the carriers to update its exploration information.

3.3.3 An Additional Assumption

As Figure 5 shows, it is impossible to visit the nodes which are disconnected from the main graph populated with agents. Thus it is also impossible to locate the black holes in the disconnected part. Hence, some papers add an assumption that the graph remains interconnected when the black holes have been removed.

Flocchini *et al.* add this assumption in their paper [46]:"after deleting all the black holes, the network still remains interconnected". It is obvious that without this assumption, it is impossible to locate all the black holes in any given network. Beyond a multiple black hole search, Flocchini *et al.* complicate the entire process by adding link deletion during the computation. An edge failure is locally detectable at an incident node only in the sense that, if information about that edge (identified by its port number) is written on the whiteboard, an agent can notice the absence of an edge with such a port number; otherwise, if no information is written, it is likely that an edge never existed. Under this assumption, the authors present an algorithm to solve the dangerous graph exploration with link deletions in an arbitrary unknown

graph with asynchronous dispersed agents using the whiteboard model.The algorithm can correctly solve the link deletion problem within finite time by marking all safe edges as such, and marking as dangerous every port that is on a safe node leading to a black hole or to a black edge. The total number of moves performed by the agents is at most $O(k^2 \cdot n_s + n_s \cdot m + k \cdot n_s \cdot D)$, where k is the number of agents, n_s is the number of safe nodes, and m is the number of edges or links.

By assuming that the graph is strongly connected after all black holes have been removed, Kosowski *et al.* [61,62] find out that $O(d \cdot 2^d)$ co-located agents are sufficient to solve the black hole search problem on a directed graph with arbitrarily large n, where the network is synchronous and d is the number of edges leading to the black holes. This bound is nearly tight: beyond showing that at least 2^d agents are required in most of the cases, the authors also provide a general strategy which requires $O(d \cdot 2^d)$ agents. Furthermore, the authors show that when $d = 2$, 4 agents are always sufficient in synchronous networks. However, in asynchronous networks, at least 5 agents are required when $d = 2$. In addition, 2 agents are required when $d = 1$ in both synchronous and asynchronous networks.

3.4 Other Types of Malicious Hosts

Beyond studying the traditional black hole and its variants, e.g. repairable black holes introduced in [21] by Cooper *et al.* and the new subway model presented by Flocchini *et al.* in [45, 48], other types of malicious hosts have also been studied. Chalopin *et al.* [18] study a rendezvous of mobile agents in a network with *faulty links*. In this model, some of the edges in the graph are dangerous for the agents such that any agent that attempts to traverse such an edge (from either direction) simply disappears, without leaving any trace. Notice that if all the edges incident to a node u are faulty, then node u can never be reached by any agent, and such a node

is equivalent to a black hole.

Kralovivc *et al.* [63] study a *periodic data retrieval problem* which is equivalent to *periodic exploration* in fault-free networks, and to a black hole location problem when there is only one black hole in the network. The aim of the periodic data retrieval problem is to infinitely deliver the data from any non-faulty node to the homebase (a node which collects all data) many times. They consider a ring network which contains one malicious host that can behave in an arbitrary way, except that it cannot change the internal state of an agent (i.e. contents of its local variables), nor create an agent with a given state.

Cai *et al.* [14] consider the problem of a *black virus* which also deletes any incoming agent, but unlike the black hole which is defined as a static host, a black virus moves from node to node, thus potentially increasing the number of dangerous nodes. In addition, unlike the black hole, which can only be located but not removed, the black virus would be destroyed when it enters a node which contains an anti-viral system agent. Thus, the only way to remove the black virus is to surround it by anti-viral system agents and force it to move to the neighbouring nodes which already contain at least one anti-viral system agent. Related to the black virus, some theoretical work has focused on an *intruder capture* problem (aka. *graph decontamination*): an intruder (a harmful agent) moves through the network infecting the nodes; the task is to remove the intruder from the network using mobile agents. Unlike the black virus, the intruder can only infect the nodes but not the agents. This problem has been extensively studied in [10, 13, 49].

Black hole attack [2, 9, 71] is also a research topic that is related to black hole search. The networks in black hole attack are different from those in black hole search. In black hole search, the networks are static, while in black hole attack, the networks can be MANET (Mobile Ad-Hoc network), wireless network, mobile and other dynamic networks. In MANET, the network topology is only formed once one

node needs to send a data package. Khari *et al.* [57] survey security attacks as well as secured routing protocols in MANET and offer a definition of black hole attack. In addition to black hole attack, the survey mentions a variation of black hole attack called *grey hole attack* [1, 8, 67] A black hole will delete any incoming data packages whereas a grey hole only deletes part of the packages.

Chapter 4

Model, Assumptions and Techniques

4.1 Model and Assumptions

The network is abstracted into an edge-labelled undirected graph $G = (E, V)$, where E denotes the edges, V denotes the network nodes (e.g., computer hosts) and n ($n = |V|$) denotes the number of nodes in G. For any $u \in V$ and $v \in V$, $(u, v) \in E$ represents the link from neighbouring nodes u to v. The links obey a FIFO rule; that is, the agents do not overtake each other when travelling over the same link in the same direction.

Let $V_f \subseteq V$ denote a set of faulty nodes, and b ($b < n$) denote the number of faulty nodes in the network. A faulty node deletes any incoming data. However, it can be repaired by the first visiting agent. After repair, a faulty node behaves like a normal node and is referred to as a *repaired node*. However, unlike a real normal node that is never a faulty node, a repaired node can be infected by a *gray virus* and consequently turned into a black hole. If one or more agents simultaneously enter a faulty node, one agent will die after repairing the fault, and all other agents will die immediately.

A *gray virus* (GV for brevity) is a piece of malicious software, which can infect a repaired node by residing in it and turning it into a black hole (agents cannot repair

a black hole); it has no destructive power on a normal node or link. There are two types of GVs: multi-stop and one-stop. A *multi-stop GV* can move from node to node or reside in a node for any duration, regardless of whether the node is normal, faulty, or repaired; if the node is normal or faulty, nothing will be changed; if the node is repaired, it will become a black hole until the GV leaves. After the GV leaves, a repaired node behaves like a normal one and may be reinfected. Unlike the multi-stop GVs, a *one-stop GV* can no longer move around once it infects a repaired node; that is, it turns the repaired node into a permanent black hole.

Let $\mathcal{A}$ denote a set of k ($k \geq 2$) identical agents in the network. These agents have limited computing capabilities and bounded storage[1], obey the same set of behavioural rules (the "protocol"), and can move from node to neighbouring node. We make no assumptions on the amount of time required by an agent's actions (e.g., computation, or movement) except that it is finite; thus, the agents are asynchronous. All agents initially wake up in the same node hb ($hb \in V$) which is referred to as their *homebase* and is assumed to be safe (e.g. not a faulty node or black hole).

Unlike Cooper *et al.* [21] who solve the MBHS problem in synchronous networks with the advantage of the time-out mechanism, in this paper we aim at solving the repair and search problem in an asynchronous network where every agent's move and computation cost a finite but unpredictable amount of time. In [21], the goal of the agents is only to repair all the faulty nodes without reporting their locations, when the network map is given in advance. In this thesis, we define the *Faulty Node Repair and Dynamically Spawned Black Hole Search* (*Repair and Search* for brevity) problem as the following: uses a team of mobile agents to repair all the faulty nodes in the entire network, and finally has at least one surviving agent which knows the location of the black hole(s) infected by gray virus.

[1]The storage is only enough to keep track the number of moves it has performed during the exploration of a new node.

Furthermore, Cooper *et al.* assume if two or more agents enter a faulty node at the same time, only one dies for the repair while the others can continue exploration. We consider a more challenging model that is after repairing a faulty node, all agents die if they simultaneously enter this faulty node.

In this thesis, we use two communication models to solve the *Repair and Search* problem in an asynchronous network: the enhanced token model and whiteboard model. Under the enhanced token model, we first solve the *Repair and Search* problem caused by a multi-stop *GV* in Chapter 5. We then provide solutions to the problem caused by a one-stop *GV* in Chapter 6 using a new technique double cautious walk. After these 2 algorithms, we offer a solution using only one whiteboard in the entire network in Chapter 7, and it does not need the cautious walk or double cautious walk technique.

4.2 A New Technique: Double Cautious Walk With Tokens

Double Cautious Walk With Tokens: an agent A (see Figure 6) marks a port in node u as dangerous by placing a token at this port before moving to the next node v. Once arriving at node v, the agent marks the entering port as dangerous by placing another token at the port and immediately returns to node u. Upon its return, this agent will pick up the first token in node u to show that this port is no longer dangerous and move again to node v. While arriving, the agent picks up the second token on the port through which this agent enters v. We call this port its *entering* port and the other port its *exiting* port.

Since the one-stop *GV* may appear and infect a vulnerable node at any point in time, a previously safe but vulnerable node u where an agent started the first step

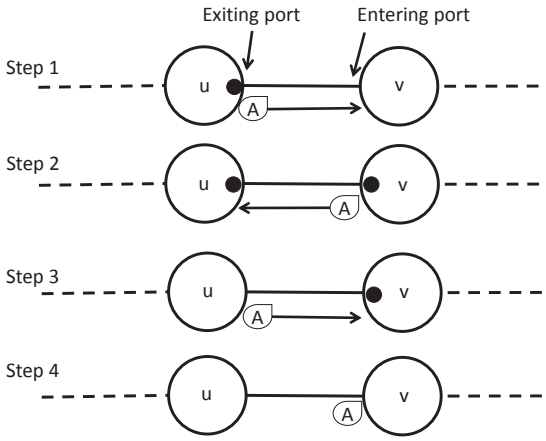

Figure 6: Double Cautious Walk with tokens: Agent A puts a token at node u and moves to v (Step 1). Agent A puts a token at v and immediately returns to u (Step 2). Upon return, A picks up a token and again moves to v (Step 3). Upon arriving, A picks up a second token (Step 4).

of a double cautious walk may be infected and become a black hole. This sudden change leads to the elimination of this agent while it returns to a previously safe node u during its double cautious walk (Step 2 in Figure 6). This is why the cautious walk with tokens [38] is no longer sufficient in terms of minimizing agent loss. In the new double cautious walk with tokens, even if an agent dies in the black hole while returning, it leaves a token at the second node, thus, further prevents extra agent loss.

During an exploration, some agents die after repairing faults, while some may die in a black hole. When an agent sees one token at a port, it knows that another agent is exploring the next node, but it cannot know whether the next node is a repaired node or a black hole. This agent needs to leave a second token at that port before entering the next node. If the next node is a black hole, the port which leads to the black hole will have two tokens on it since both agents died in it. Otherwise, if the next node becomes a repaired node after the previous agent dies, this agent will eventually pick up its first double cautious walk token. Thus, this mechanism

distinguishes the black hole from a repaired node. An agent will never leave via a port with two tokens (*a 2-token port*) unless it is the same port, via which it just entered a node.

The characteristic of the GV that can infect a repaired node at any time has significantly complicated the *Repair and Search* problem. For a link (u, v), it is possible that 2 agents, A and B, enter via port p1 sequentially (not simultaneously due to the FIFO rule) after each putting down 1 token; this leaves 2 tokens on port p1 (see Figure 7). Both agents A and B proceed to node v and leave 2 tokens in total on p2.

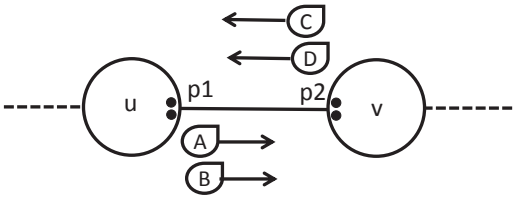

Figure 7: During a double cautious walk, after visiting nodes u and v, two agents A and B die in a previously visited node u that is now a gray virus infected black hole.

In traditional black hole search, any node from which an agent comes is safe. Therefore, when the 2 agents return to node u via link (u, v), they know that nodes u and v are safe. They will be able to safely return to their previous nodes regardless of the number of tokens left on the entering port. However, with a GV in the network, a previously visited node is no longer guaranteed to be safe. It is possible that the previously visited node u of agents A and B has been turned into a black hole. In this situation, both sets of 2 tokens on ports p1 and p2 will remain forever. This situation becomes more complex when there is one or more agent(s) traversing the ring in opposite directions (e.g., agents C and D in Figure 7).

Chapter 5

Repair and Search in the Presence of a Multi-stop Gray Virus in Arbitrary Networks Using Tokens

5.1 More on Model and Assumptions

In the network, Δ is the maximal node degree of G, and for each node $u \in V$, all the ports are randomly but distinctly labelled as 1, 2, 3, ..., Δ. All agents have no network topology knowledge a priori. Each agent is endowed with a limited number of tokens which can be put on or picked up from a port or the centre of a node. Any token on a node is visible to all agents on the same node. Since the agents are invisible to each other, thus, the only way to exchange information is through the use of tokens. All agents know n and b. As minimally one agents die in a faulty node in order to repair it, at least $b + 1$ agents are required to allow one surviving agent to report the locations of the repaired nodes (potential GV infected nodes).

It is important to note that in an asynchronous network, a GV's moving speed is unpredictable. This leads to a crucial observation:

Observation 1 *When a multi-stop GV moves much faster than the agents, from*

the agents' point, it could be the case that all the repaired nodes appear to be black holes. Consequently, the Repair and Search problem becomes unsolvable without any additional assumption (same as the MBHS problem: some parts of the network may become disconnected).

We assume there is only one multi-stop GV in the network and it can only appear after all faulty nodes have been repaired and all repaired nodes have been located. A repaired node cannot be turned into a black hole while there is an agent in the node. We offer an algorithm which can repair all the faulty nodes with a minimal number of agents, and construct a map of the unknown network with all the repaired nodes marked.

5.2 Solution and Algorithms

5.2.1 Overview

In this algorithm, each agent independently explores the graph and constructs its own map. Through out the execution of this algorithm, the *Cautious walk with token* technique which introduced in [68] is used: When executing cautious walk under this model, an agent marks a port as dangerous by placing a token at this port before moving to the next node. Upon its return, this agent will pick up the token to show that this port is no longer dangerous [38]. No more than one agent can put or pick up tokens at the same place at the same time.

When an agent builds its map, a node can be as follows:

- *completed* - all its neighbouring nodes have been explored;
- *incomplete* - the agent has visited it but some of its neighbouring nodes remain unexplored (it has unexplored ports);

- *repaired* - a repaired node;
- *suspected* - suspected to be a repaired node but not yet proved.

In addition to the nodes, a port can be

- *unexplored* - if the agent has never left via the port
- *explored*- if the agent left via the port
- *suspected* - suspected to lead to a repaired node. Let sp denote the number of suspected ports.

When entering a node (including wakes up at hb), the agent first numbers the node from 1 to n (hb is node 1) and explores all its neighbours. If the agent survives, it moves to an incomplete neighbour via the port with the smallest port number or to the incomplete node with smallest number; for example, if port 1 of node v leads to a completed node while port 2 leads to an incomplete node, the agent leaves via port 2, and if all ports of node v lead to completed nodes, the agent moves to the incomplete node with the smallest number. This process should be repeated until all n nodes have been explored, or the agent dies in a faulty node.

If the agent finishes the entire graph exploration, it moves from the largest node to the smallest (hb is the smallest) and marks all suspected ports in its map. If the number of suspected nodes is equal to b, the algorithm terminates. The agent finally arrives at hb. If there is no token in the centre of hb, the agent puts two tokens in the centre and waits for another agent to pick up one; the agent again goes to count and check the suspected nodes and returns to hb after rechecking; while returning, the agent puts a second token in the centre of hb; the algorithm terminates when the number of suspected nodes is equal to the number of faulty nodes. If there is one token in the centre, the agent waits until the number of tokens becomes two and picks

up one, and then becomes inactive. If there are two tokens in the centre, the agent picks up one and becomes inactive.

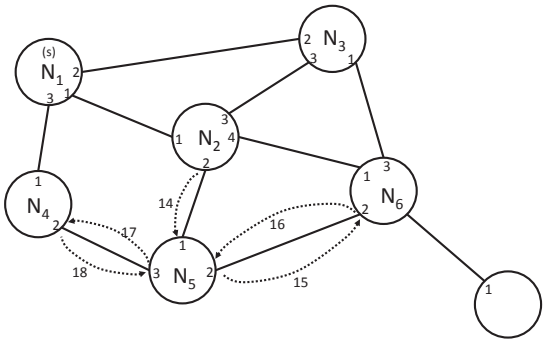

Figure 8: An agent moves to the next node through the smallest incomplete port (route 14). For example, an agent visits N_6 and N_4, and only adds N_6 as a new node since N_4 has already been added into its map during previous exploration of this node via another route

5.3 Correctness and Complexity Analysis

Lemma 2 *All the agents follow the same path, thus, only b agents die as a result of repairing the faulty nodes.*

Proof. When building its map, the agent chooses a neighbour or a next node without considering the tokens on the node and only concerning the port numbers. Since all the agents wake up at the same node *hb* and the port numbers do not change, all the agents will make the same decision on each node, thus, follow the same path. As all the agents follow the same path and no more than one agent can enter a faulty node at the same time through the same port, only 1 agent dies in each faulty node. Therefore, only b agents die as a result of repairing the faulty nodes. ■

Lemma 3 $b + 1$ *agents are necessary and sufficient to repair all the b faulty nodes and locate all the repaired nodes.*

Proof. Clearly, to repair b faulty nodes, b agents are necessary. In order to report the locations of the repaired nodes, at least 1 agent has to survive and return to hb. Therefore, $b+1$ agents are necessary. As proven, only b agents die in Lemma 2 and all faulty nodes have been repaired, the extra 1 agent will finish the graph exploration and know the locations of all the repaired nodes. Hence, $b+1$ agents are also sufficient. ■

Lemma 4 *Only one agent can put* 2 *tokens at hb, and count the number of repaired nodes. The algorithm can terminate within finite time.*

Proof. Assume agent X is the first agent which puts two tokens in the centre of hb. By the time X puts the two tokens, the other agent can only pick up one. While there is only one token left, an agent has to wait until X puts a second one. Therefore, no other agent can put tokens in the centre except X. Agent X is the only agent that counts the number of suspected nodes, and finally updates only b suspected nodes to repaired nodes.

Within finite time, all the survived agents will finish the entire graph exploration and return to hb. By the time all agents return to hb, only the dead agents have tokens left on suspected ports, and only the b agents have died. Therefore, $sp = b$, and the algorithm terminates. ■

Lemma 5 *In the worst case, the repair and locate task can finish within* $O(kn\Delta)$ *moves in total.*

Proof. Step 1: when $k = b+1$

As in Lemma 2, b agents die in the faulty nodes. This leaves only one agent A to finish the entire graph exploration. Therefore, when A finishes exploring the entire network, there are b suspected nodes. When the agent returns to hb, the algorithm terminates. During an exploration, an agent needs at most 2Δ moves to explore every

new node. For k agents to explore at most n nodes each, the worst case total number is $2kn\Delta$. During the counting step, in the worst case, the surviving agent A needs to traverse the entire network to count the number of suspected nodes. The moves for this step is n. In summary, the total moves are $2kn\Delta + n$ when $k = b + 1$.

Step 2: assuming the total number of moves is at most $2kn\Delta + kn + (k-b)n + 2(min(n,k) - b)n$ when $k = b + i$, we want to prove that the total number of moves is at most $2kn\Delta + kn + (k-b)n + 2(min(n,k) - b)n$, when $k = b + i + 1$.

Again, as proven in Lemma 2, b agents die in the faulty nodes. This leaves $k - b$ surviving agents that can finish exploring the entire network. During an exploration, an agent needs at most 2Δ moves to explore every new node. For k agents to explore at most n nodes each, the worst case total number is $2kn\Delta$. When one of the surviving agent A finishes exploring the entire network, it moves back to hb. When the agent returns to hb, it waits for the next surviving agents to come back and proceeds to a counting step upon the arrival of each proceeding surviving agents. $(k-b)n$ moves are taken by $k-b$ agents coming back from the last node in the network to hb. During the counting step, in the worst case, a surviving agent A needs to traverse the entire network to count the number of suspected nodes. The counting agent needs at most $2(sp - b)n$ moves. As $sp < min(n,k)$, $(sp-b)n < (min(n,k) - b)n$. In summary, it costs at most $2kn\Delta + kn + (k-b)n + 2(min(n,k) - b)n$ moves.

Hence, in the worst case, the *repair and search* task can finish within $O(kn\Delta)$ moves in total. ■

Theorem 6 *After $O(kn\Delta)$ moves, $b+1$ agents are necessary and sufficient to repair and locate b faulty nodes in an arbitrary unknown network.*

Chapter 6

Repair and Search in the Presence of a One-stop Gray Virus in Ring Networks Using Tokens

6.1 More on Model and Assumptions

We study the one-stop *GV* under almost the same assumptions as the multi-stop *GV* case. Since a one-stop *GV* can only infect one vulnerable node, one and only one vulnerable node will become infected and turn into a black hole. Another difference between the two sets of assumptions is that we study the *Repair and Search* problem caused by the one-stop *GV* in an un-oriented ring network with n nodes. In an un-oriented ring, the agents are not able to agree on a common sense of direction [41] (e.g. the *Left* direction to one agent might be the *Right* to another agent). All agents initially locate at homebase *hb* and know the network topology is a ring.

Observation 7 *If the one-stop GV appears after all faulty nodes have been repaired, the Repair and Search problem becomes a faulty node repair problem and a single black hole search problem with all the possible locations of the black hole known a priori.*

In this case, the *Repair and Search* problem is even easier than a traditional single black hole search problem. We are interested in studying the scenario in which a one-stop GV may appear and infect a repaired node at any time that is no later than all the faulty nodes have been repaired. Contrary to the traditional black hole search that has one black hole before the search starts, having a black hole that appears at a random time significantly increases the difficulty and complexity of the task. We will further explain this observation later in this section. We offer an algorithm that can repair all the faulty nodes, and locate the GV within finite time.

6.2 Algorithm Pair-Block

6.2.1 General Description

An agent, A_1, wakes up at homebase hb and randomly chooses one direction to explore the ring using the double cautious walk. If A_1 is blocked by a 2-token port on node N_x, A_1 marks N_x as left-block in its memory. A_1 also memorizes whether there is any token in the centre of this left-block. A_1 reverses direction and continues exploration until blocked by another 2-token port on node N_y; A_1 remembers N_y as its right-block. An agent is said to be *Blocked* when there are 2 or more tokens on the exiting port of a node in its current exploration direction. A_1 counts the distance between the left-block and right-block. If the distance is $n-2$, the algorithm terminates and the only unexplored node is the infected black hole; if the distance is smaller than $n-2$, A_1 executes the following:

If there is no token in the centre of either the left-block nor the right-block, A_1 puts 2 tokens in the centre of the right-block (2-token-centre node). A_1 then returns to look for its left-block until it arrives at its left-block or is blocked by another 2-token port. For the later case, the new node becomes the new left-block of A_1. A_1 puts 1

token in the centre of its left-block and waits. All other details of this procedure can be found in Procedure 2.

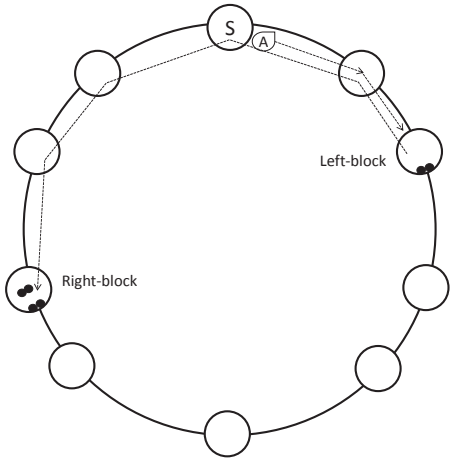

Figure 9: The simplest situation: there are no other agents between the left-block and right-block of agent A

If the left-block is not beside a black hole, at least 1 token at the 2-token port will disappear. When A_1 sees 1 token is gone, it looks for its right-block to pick up the 2 tokens in the centre. A_1 then returns to its left-block and continues its exploration (see all situations in Procedure 4).

Another scenario could be that another agent A_2 picks up 1 token from the 2-token port of the right-block, if this right-block is not beside a black hole. When A_2 sees 2 tokens in the centre left by A_1, A_2 picks them up and becomes a *sender*. The sender goes to a 1-token-centre node (e.g., left-block of A_1), and drops the extra 2 tokens in the centre to inform A_1. After this, A_2 reverses its direction and continues its previous exploration. When A_1 sees 3 tokens in the centre, A_1 picks up all the tokens and goes to its right-block to continue exploration. Such exploration continues until one agent finds there are $n-1$ nodes between its left and right blocks.

Detailed Explanation

Procedure "Pair-Block" An agent wakes up at hb and randomly chooses one port to explore the ring using the double cautious walk. When the agent returns to pick up its token during cautious walk, if it sees 2 tokens in the centre, it becomes a sender and sends the extra tokens to a 2-token port with 1 token in the centre. If an agent does not become a sender, it will meet a pair of left-block and right-block, and then either wait at its left-block, terminate the algorithm, or execute Procedure 2. In addition, clear memory means the agent deletes the information about left-block and right-block. After memory is cleared, an agent will memorize other nodes with 2-token ports as new left-block or right-block.

Algorithm 1 PAIR-BLOCK

```
1: Wake up at hb. Choose one port to explore the ring using double cautious walk.
2: loop
3:     if Blocked by a port has more than 2 tokens during exploration then
4:         Execute the same as being blocked by a 2-token port.
5:     end if
6:     if See 2 tokens in the centre when returning during double cautious walk then
7:         Pick up the 2 extra tokens in the centre. Become a sender.
8:         if Blocked by a 2-token port with one token in the centre then
9:             Put 2 tokens in the centre. Reverses direction. Clear memory.
10:        else if Blocked by a 2-token port then
11:            Reverses direction. Delete the 2 extra tokens. Clear memory.
12:        end if
13:    end if
14:    Continue exploration until blocked by a 2-token port. Mark this node left-block.
15:    Reverse direction and continue exploration until blocked by another 2-token port.
           Mark this node right-block.
16:    Count the distance d of left-block and right-block.
17:    if d = 0 then
18:        Wait until any 2-token port has fewer than 2 tokens. Leave through that port
               and continue exploration. Clear memory.
19:    else if 0 < d < n − 2 then
20:        Execute GoBack.
21:    else if d = n − 2 then
22:        Terminate the algorithm.
23:    end if
24: end loop
```

Procedure "GoBack" To execute this procedure, an agent has met its left-block and is currently at its right-block. Let C_l and C_r denote the number of tokens in the centre of its left-block and right-block respectively. The agent puts i $(i = 0, 1)$ token or picks up 1 token to make $C_l = 1 \& C_r = 2$. The agent then waits at its left-block. An agent may meet no/one/two tokens in the centre of its left-block and right-block. Exchange names means to exchange the names of its left-block and right-block; for example, nodes N_x and N_y are left and right blocks, respectively, so that after name exchange, N_y becomes left-block and N_x is right-block. All possibilities of C_l and C_r and actions 1 & 2 that an agent will take in these situations are shown in Table 4.

Algorithm 2 GoBack

if $C_l + C_r = 0$ **then**
    Put 2 tokens in the centre of right-block. $i = 1$. Execute **GoToOne**.
else if $C_l = 0 \& C_r = 1$ **then**
    Go to left-block
    if Arrive left-block and the 2-token port still has 2 tokens **then**
        Put 2 tokens in the centre of left-block. Exchange names. $i = 0$. Execute **GoToOne**.
    else if Arrive left-block and the 2-token port has fewer than 2 tokens **then**
        Clear memory.
    else if Blocked by a 2-token port before left-block **then**
        Update this node as left-block. Exchange names. Execute **GoBack**.
    end if
else if $C_l = 0 \& C_r = 2$ **then**
    $i = 1$. Execute **GoToOne**.
else if $C_l = 1 \& C_r = 0$ **then**
    Put 2 tokens in the centre of right-block. $i = 0$. Execute **GoToOne**.
else if $C_l = 1 \& C_r = 1$ **then**
    Put 1 token in the centre of right-block. $i = 0$. Execute **GoToOne**.
else if $C_l = 1 \& C_r = 2$ **then**
    $i = 0$. Execute **GoToOne**.
else if $C_l = 2 \& C_r = 0$ **then**
    Put 1 token in the centre of right-block. Exchange names. Execute **WaitToFind**.
else if $C_l = 2 \& C_r = 1$ **then**
    Exchange names. Execute **WaitToFind**.
else if $C_l = 2 \& C_r = 2$ **then**
    Pick up 1 token in the centre of right-block. Exchange names. Execute **WaitToFind**.
end if

Table 4: Number of tokens on nodes left-block and right-block. T: token. C: the centre.

Combi-nations	C_l	C_r	Action 1	Action 2
1	0	0	Put 2T in C of right-block. Go to left-block	Put 1T in C of left-block and wait.
2	0	1	Go to left-block. Put 2T in C of left-block	Go to right-block. Wait.
3	0	2	Go to left-block	Put 1T in C of left-block. Wait.
4	1	0	Put 2T in C of right-block	Go to left-block. Wait.
5	1	1	Put 1T in C of right-block	Go to left-block. Wait.
6	1	2	Go to left-block	Wait.
7	2	0	Put 1T in C of right-block	Exchange names. Wait.
8	2	1	Exchange names. Wait.	
9	2	2	Pick up 1T in C of right-block	Exchange names. Wait.

Procedure "GoToOne" The agent is currently at its right-block and needs to move to its left-block. It may arrive at its left-block successfully or meet another 2-token port. There are three main situations that may happen and should be discussed whenever an agent moves from its left-block to its right-block .

1. A_1 arrives at its left-block, and the 2-token port still has 2 tokens.

2. A_1 arrives at its left-block, and the 2-token port has fewer than 2 tokens.

3. A_1 is blocked by a 2-token port with no/one/two tokens in the centre before

arriving at its left-block.

Algorithm 3 GoToOne

```
1: if Arrive left-block and the 2-token port still has 2 tokens then
2:     Put i token in the centre of left-block. Execute WaitToFind.
3: else if Arrive left-block, and the 2-token port has fewer than 2 tokens then
4:     Go to right-block.
5:     if Arrive at right-block and see two tokens in the centre then
6:         Pick up all tokens in the centre. Reverse direction. Clear memory.
7:     else if Arrive at right-block then
8:         Reverse direction. Clear memory.
9:     else if Blocked by a 2-token port with 2 tokens in the centre then
10:        Pick up all tokens in the centre. Reverse direction. Clear memory.
11:    else if Blocked by a 2-token port then
12:        Reverse direction. Clear memory.
13:    end if
14: else if Blocked by a 2-token port with no tokens in the centre then
15:    Put 1 token in the centre, update this node as left-block. Execute WaitToFind.
16: else if Blocked by a 2-token port with 1 token in the centre then
17:    Update this node as left-block. Execute WaitToFind.
18: else if Blocked by a 2-token port with 2 tokens in the centre then
19:    Pick up 1 token. Update this node as left-block. Execute WaitToFind.
20: end if
```

Procedure "WaitToFind" An agent has met its right-block and now waits at its left-block. The agent waits until 1 token on the exiting port is gone or $C_l = 3$. For the former case, the agent looks for the 2 tokens in the centre of its right-block. For the later case, the agent picks up all 3 tokens and goes to its right-block to continue exploration.

6.3 Correctness and Complexity Analysis

Lemma 8 *Minimally $b + 2$ agents are necessary to repair all faulty nodes and locate the black hole in an asynchronous ring network.*

Proof. Since there are b faulty nodes in the ring network and 1 agent can only repair 1 faulty node, b agents are needed. To distinguish the black hole from the repaired

Algorithm 4 WaitToFind

```
1: Wait.
2: if 1 token on the exiting port is gone. then
3:     Go to right-block.
4:     if Blocked by a 2-token port with 2 tokens in the centre then
5:         Pick up 2 tokens in the centre. Return to left-block.
6:     else if Blocked by a 2-token port or arrive right-block. then
7:         Return to left-block.
8:     end if
9:     if Arrive at left-block and there is 1 tokens in the centre then
10:        Pick up the token in the centre. Clear memory.
11:    else if Blocked by a 2-token port. then
12:        Reverse direction. Clear memory.
13:    else
14:        Clear memory.
15:    end if
16: else if 1 token in the centre becomes 3 then
17:    Pick up all 3 tokens in the centre. Go to right-block and clear memory.
18: end if
```

nodes, at least one agent has to enter the black hole and die, thus, $b+1$ agents are required. To report the locations of the faulty nodes and the black hole, at least 1 agent has to survive, hence, $b+2$ agents are necessary. ■

Lemma 9 *$b+9$ agents are sufficient to report all repaired faulty nodes and locate the black hole in a ring network.*

Proof. Since there are b faulty nodes in the ring network and 1 agent can only repair 1 fault, b agents are necessary to repair all faulty nodes. In order to distinguish the black hole from the repaired nodes, each port which leads to the black hole will have 2 tokens, each of which was left by one agent. Because there is already 1 token left from the previous faulty node repair, 3 more tokens from three different agents will be needed. At least 1 agent has to survive and report the locations, thus, minimally $b+4$ agents are required to repair all the faulty nodes and locate the black hole that may appear at any random point in time.

If two agents simultaneously enter the same faulty node, they both die. Since the agents travelling on the same link in the same direction obey the FIFO rule, maximum 2 agents (one from each direction) can enter the same faulty node and simultaneously die. If 2 agents meet from different directions, all faults from their common homebase hb to the meeting node have been repaired. Therefore, in the ring network, there is only one black hole and no faulty nodes after this meeting, so that one more agent is needed in this situation and $b + 5$ agents are necessary.

As a GV can infect a repaired node at any time, it is also possible that node v turns into a black hole while an agent is returning to v after picking up its first token (Step 3 in Figure 6), which makes the agent die in the black hole without leaving any trace in the network. For a repaired node, 1 agent has already died for repairing it and left 1 token on a port that leads to this repaired node, so that 3 more agents can enter node v and die.

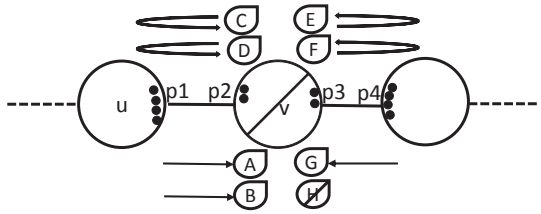

Figure 10: At most 8 agents die in the black hole

Assume agents A and B travel in the opposite direction as agents C and D. They left their first cautious walk tokens on port p1 and p2, respectively. While agents A and B are moving towards node v, node v becomes infected and turned into a black hole, so that agents A and B will immediately die when they arrive at node v. Agents C and D will also die in node v after they leave their second cautious walk tokens on port p1. None of these 4 agents can return to pick up their tokens on port p1 (see Figure 10). According to Line 4 in Procedure 1, all agents treat such a port the same

as a 2-token port.

If node v can be turned into a black hole, it is a repaired node, so that there has been an agent H who died for repairing it. Since agent H immediately dies after arriving at a faulty node, it cannot leave its second double cautious walk token, thus, it leaves only 1 token on port p4. If agents G, E, and F die in the same situation as agents A, C, and D, there will be 2 and 4 tokens on port p3 and p4 respectively. In summary, 7 agents die in the black hole and 1 agent dies for fault repair. This is true even if agents A, B, and G die while returning after picking up their first token instead of moving towards node v, because the second cautious walk tokens left on port p1 by agents C and D, as well as the 2 tokens on port p4 by agents E and F, will remain and block other agents. Since regularly only agents A, B, G, and H die in the black hole, 4 extra agents die in this situation, which requires more agents than the above situation where only 3 agents die without leaving any trace. These two situations cannot happen in the same network. Therefore, $b + 9$ agents are required in total.

■

Lemma 10 *Algorithm Pair-Block can correctly repair all faulty nodes and locate the black hole.*

Proof. For any port with 2 or more tokens, there are three situations in which the agents which left these tokens: all active agents, only one dead agent (for fault repair), at least two dead agents. For the first two cases, all active agents will return to pick up their tokens and leave at most 1 token on the port, so that the port can only temporarily block other agents. Since all links obey the FIFO rule, a port with 2 or more dead agents leads to a black hole. An agent will be permanently blocked by such a port in one direction, hence, it can only explore the ring in the opposite direction and either die or arrive at the other side of the black hole. If an agent has

been blocked by 2 ports that lead to the black hole, it has visited $n-1$ nodes and located the black hole. ■

Lemma 11 *In the worst case, the Repair and Search task can finish within $O(kn^2)$ $(k \geqslant b+9)$ moves in total.*

Proof. As proved in Lemma 9, in the worst case all accidents happen and leave only 1 surviving agent. As there is a black hole in the network and each port that leads to the black hole will have at least 2 tokens, the surviving agent will be blocked by the two ports and the distance is $n-2$, thus, the algorithm is terminated.

For any agent, one move of cautious walk needs 3 actual moves. According to Procedure 1 lines 14 and 15, forming a pair of left-block and right-block needs up to $2*3(n-2)$ moves. Also in Procedure 1 or Procedure 4, removing a pair may also need up to $2*3(n-2)$ moves. If every exploration of a new node by an agent creates a pair of left and right blocks, up to $(2*3+2*3)*(n-2)*n$ moves are required. Furthermore, if this happens to every agent, a total of $12*k*n*(n-2)$ moves are required to form and remove a pair of blocks for all agents. Hence, $O(kn^2)$ moves are required to repair all faulty nodes and locate the black hole in the worst case. ■

Theorem 12 *After $O(kn^2)$ $(k \geqslant b+9)$ moves, $b+9$ agents are sufficient to repair and locate b faulty nodes and the black hole infected by a one-stop gray virus at a random time.*

Chapter 7

Repair and Search in the Presence of a One-stop Gray Virus in Ring Networks Using only one Whiteboard

7.1 More on Model and Assumptions

The *Repair and Search* problem is studied in an anonymous ring network with a one-stop *GV* in this chapter. The agents know the network topology is a ring a priori. All agents execute the same protocol and know the number of nodes n, however have no knowledge about the number of faulty nodes b. A *whiteboard* [31] (e.g. shared memory) in the homebase is the only means of communication between agents. This whiteboard available in hb can be accessed in fair mutual exclusion. No other communication methods (e.g., a whiteboard on other nodes or agents which can sense or talk to each other when they meet in the same node) are otherwise available.

It is known that in an un-oriented ring, there is no agreement on a common sense of direction among the agents [31] (e.g. the *Left* direction to agent a_1 might be the *Right* of another agent a_2). But, since all agents are co-located and can access the whiteboard, the agents can agree on the clockwise direction of the ring even if the ring

is un-oriented. Therefore, with the help of this whiteboard in the common homebase, all agents know that the clockwise direction is the left of the ring, and the counter-clockwise direction is the right of the ring. In order to ease the understanding of the algorithm description, N_0, N_1, ... N_{n-1} are used to refer to the nodes of the ring sequentially in left direction starting from the homebase hb.

7.2 Algorithm Dynamically Spawned Black Hole Search

7.2.1 General Description

When an agent A wakes up at hb, it first checks the whiteboard. If the whiteboard is blank, agent A initializes the whiteboard as shown in Table 5. Agent A puts a *leaving* mark (?) in the cell of First Agent for node N_1, and then goes to visit node N_1 which is the node 1 step away from hb in the left direction. Upon its arrival, agent A returns to hb immediately. Once A returns to hb, agent A changes the leaving mark to a *returned* mark ($\surd$) (Table 6 shows an example). This agent immediately clears its memory and acts as a newly waking up agent.

By repeating this process, agent A explores nodes $N_2, N_3, ..., N_i$ that are $2, 3, ..., i$ steps away from hb in the left direction. Other agents such as B may wake up any time during A's exploration. Upon waking up, agent B sees a leaving mark for node N_i, it knows that A has either died after exploring a faulty node, was eliminated by a black hole, or is still doing its normal exploration on node N_i. Agent B then goes to node N_i to confirm the status of agent A. B puts a leaving mark under Second Agent on node N_i. Upon its arrival, agent B returns to hb immediately and changes the leaving mark into a returned mark. By the time agent B returns, agent A may have returned, which means node N_i is not a faulty node. Since we assume that all

links and nodes are FIFO, agent B can return before agent A only when agent A has died after repairing a faulty node. B can then conclude that node N_i is a repaired faulty node. In this situation, B will change the *leaving* mark (?) of A into a *died* mark ($\times$) (see Scenario S5 in Table 7).

When another agent C wakes up, it observes from the node list on the whiteboard in *hb* that two agents have left in the left direction, it immediately starts exploring the ring in right direction to visit node $N_{n-1}, N_{n-2}, ..., N_{n-j}$. This is because if the black hole has already appeared, going into the opposite direction will avoid unnecessary loss of agents. When the Fourth Agent D wakes up before agent C returns from node N_{n-j} and there are still two leaving agents in the left direction, agent D goes to node N_{n-j} in the right direction and returns to *hb* immediately. If there are 2 agents leaving in each of the two directions, then the Fifth Agent, after waking up, will wait at *hb* until at least one agent comes back from either of the two directions. All agents start again executing the above mentioned procedures in a FIFO fashion until every node in the network has at least one agent returned from it. After this point, we know that all faulty nodes have been repaired. In addition, each such repaired node is indicated on the whiteboard with a *returned* mark ($\surd$) under the Second Agent column and a *died* mark ($\times$) under the First Agent column on a node.

At this moment, the only work left is to identify the black hole that is one of the repaired nodes which is newly infected by the GV. Obviously if there is only one repaired node, it is the black hole. When there is more than one repaired node, the remaining task can be achieved by letting one agent A visit in the left direction and another agent B visit in the right direction all nodes one by one. Upon checking each node, either agent A or agent B needs to return to *hb* and update the whiteboard. When all but one of the repaired nodes have been checked by an agent, the remaining node is the black hole. It is important to notice that this is the simplest situation that may happen, that is, the black hole does not appear and eliminates agents on

Table 5: Homebase Whiteboard Initial State

Node List	First Agent	Second Agent	Third Agent	Fourth Agent	Repaired Node List
N_1					
N_2					
...					
N_{i-1}					
N_i					
...					
N_{n-j}					
N_{n-j+1}					
...					
N_{n-2}					
N_{n-1}					

their way returning to *hb*. It is possible that an agent dies in a newly spawned (i.e. infected by a *GV*) black hole on its way returning to *hb* from exploring a new node, say N_x. If two agents visiting N_x died this way, it leads to the scenario in which N_x appears (on the whiteboard) to have no agent returned from it, even though it is not the black hole. We discuss the details of procedures *Find the Meeting Node* and *Black Hole Search* in the following subsections.

7.2.2 Procedure "Homebase Initialization"

When an agent wakes up at homebase *hb*, if the whiteboard is blank, this agent initializes the whiteboard as shown in Table 5. Recall that N_0, N_1, ... N_{n-1} denote the nodes of the ring in clockwise direction, *hb* is node N_0, and for description purpose, we say that the clockwise direction is the *left* of the ring.

Each agent may write three kinds of notes in the whiteboard. An agent writes a

leaving mark ? when it leaves for a node as well as the direction it leaves using l (left) or r (right). When an agent returns to its homebase, it changes its leaving mark ? into a returned mark $\surd$ along with the direction from which it has left hb. When a second agent returns before its first agent, apart from changing its own leaving mark ? to a returned mark $\surd$, it also changes the ? of the first agent to a died mark $\times$. Table 6 shows an example of how an agent indicates its leave $?(l/r)$ and return $\surd(l/r)$ notes.

Table 6: An example of how agents indicate their status.

Node List	First Agent	Second Agent
N_1	$\surd(l)$	
N_2	$\times$	$\surd(l)$
...		
N_i	$?(l)$	$?(l)$
...		
N_{n-2}	$?(r)$	
N_{n-1}	$\surd(r)$	$?(r)$

Algorithm 5 Procedure HOMEBASE INITIALIZATION

```
1: wake up
2: if whiteboard is blank then
3:     initialize the whiteboard to Table 5
4: end if
5: execute Procedure NEW NODE EXPLORATION
```

7.2.3 Procedure "New Node Exploration"

After the First Agent initializes the whiteboard in hb upon waking up or each time an agent returns to hb after exploring a new node, it starts a new task. It goes through the node list from the top to the bottom. The agent may find a node:

1. *unexplored*, that is a node that has no mark on the whiteboard (i.e. the row of the node in Table 5 is empty); or

2. *repaired*, that is a node has a $\times$ under the column before the column with a $\surd$ (i.e. the Second Agent returned but the first one did not. See Scenario S5 in Table 7); or

3. *safe*, that is a non-faulty node that has a $\surd$ under the First Agent column (i.e. the First Agent has returned. See Scenario S2, S4 and S6 in Table 7); or

4. *unknown*, that is a node that has a ? under under the First Agent column or both First and Second Agent columns. (i.e. both agents have left but no agent ever returned. See Scenarios S1 and S3 in Table 7).

The status of a node is considered to be *known* if it is either *safe* or *repaired*. In Scenario S5, the First Agent is confirmed to be dead as soon as the Second Agent exploring the same node returns. This Second Agent will turn the ? mark left by the First Agent into a $\times$. In Scenarios such as S1, S3 and S4, the ? mark can only indicate that an agent has left to explore a node, but whether it is dead remains unknown.

Table 7: Agents leaving and returning to the homebase Scenarios as marked on the whiteboard

Scenarios	First Agent	Second Agent	Targeted node is	No. of status unknown agents
S1	? (l/r)		Unknown	1
S2	$\surd$ (l/r)		Safe	0
S3	? (l/r)	? (l/r)	Unknown	2
S4	$\surd$ (l/r)	? (l/r)	Safe	1
S5	$\times$	$\surd$ (l/r)	Repaired node	0 (1 died)
S6	$\surd$ (l/r)	$\surd$ (l/r)	Safe	0

?: a status unknown agent that left to explore a node.
$\times$: an agent died either in a black hole or after repairing a faulty node.
$\surd$: an agent that has returned to hb

While scanning the nodes list in the whiteboard, an agent A counts the number of status unknown agents pd until it finds an unexplored node N_i. It determines the next step accordingly. If A cannot find an unexplored node, the agent will finish searching the whole list and execute Procedure Find The Meeting Node.

- When $pd = 0$, the agent goes to the next unexplored node N_i.
- When $pd = 1$, there are two possibilities:
    1. When node N_{i-1} is in Scenario S1, the agent goes to node N_{i-1}; or
    2. When node N_{i-1} is in Scenario S4, the agent goes to node N_i.
- When $pd = 2$, there are also two possibilities:
    1. When node N_{i-1} is in Scenario S3; or
    2. When nodes N_{i-1} and N_{i-2} are in Scenarios S1 and S4 respectively.

In both cases, A restarts the search from bottom to top of the node list and also counts the number pd_2 of status unknown agents until it finds an unexplored node N_{n-j}. If $n - j \geqslant i$, A executes the following:

1. When $pd_2 = 0$, A goes to node N_{n-j}.
2. When $pd_2 = 1$, there are two possibilities:
   (a) When node N_{n-j+1} is in Scenario S1, A goes to node N_{n-j+1}; or
   (b) When node N_{n-j+1} is in Scenario S4, A goes to node N_{n-j}.
3. When $pd_2 = 2$, A waits at hb until $pd < 2$ or $pd_2 < 2$, then execute Procedure New Node Exploration again.

It is important to know that when $n - j = i$, node N_i is the last unexplored node. Once N_i is explored, it becomes a *meeting node* (this will be explained in the next subsection).

7.2.4 Procedure "Find the Meeting Node"

This procedure gets executed when an agent A searches for a new task at hb and finds that all nodes have been explored. Agent A starts counting the number of status unknown nodes. Agent A executes Procedure Black Hole Search when there are no more unknown nodes in the node list. Otherwise, agent A counts the number of status unknown agents pd in the entire list and executes the following:

1. When $pd > 4$, A waits at hb.
2. When $pd = 4$ and the 4 status unknown agents are not on the same node, A waits at hb.
3. When $pd = 4$ and the 4 status unknown agents are marked on the same node, A starts Procedure Black Hole Search immediately.

Algorithm 6 NEW NODE EXPLORATION

loop
  clear previous memory and search the node list from top to bottom
  if an unexplored node N_i is found **then**
    count the number pd of status unknown agents
  else if no unexplored node is found in the whole node list **then**
    execute Procedure FIND THE MEETING NODE
  end if
  if $pd = 0$ **then**
    put a $?(l)$ in First Agent column and go to node N_i
  else if $pd = 1$ and node N_{i-1} is in Scenario S1 **then**
    put a $?(l)$ in Second Agent column and go to node N_{i-1}
  else if $pd = 1$ and node N_{i-1} is in Scenario S4 **then**
    put a $?(l)$ in First Agent column and go to node N_i
  else if $pd = 2$ **then**
    search from bottom to top of the node list and count the number pd_2 of status unknown agents until an unexplored node N_{n-j} is found
    if $pd_2 = 0$ **then**
      put a $?(r)$ in First Agent column and go to node N_{n-j}
    else if $pd_2 = 1$ and node N_{n-j+1} is in Scenario S1 **then**
      put a $?(r)$ in Second Agent column and go to node N_{n-j+1}
    else if $pd_2 = 1$ and node N_{n-j+1} is in Scenario S4 **then**
      put a $?(r)$ in First Agent column and go to node N_{n-j}
    else if $pd_2 = 2$ **then**
      wait at hb until $pd < 2$ or $pd_2 < 2$. Execute Procedure NEW NODE EXPLORATION
    end if
  end if
  upon arriving, return to hb immediately
  upon returning, change own $?(l/r)$ into $\surd(l/r)$
  if the current agent is the Second Agent and the First Agent is ? **then**
    change the ? of the First Agent to $\times$
  end if
end loop

4. When $pd < 4$ and there are no nodes in Scenario S1:

   - If all status unknown agents are not on the same node and one node has both $?(l)$ and $?(r)$, A goes to the same node as the Third Agent;
   - If all status unknown agents are on the same node, A puts a ? in the empty slot of its own agent column, then goes to that node.

   Furthermore if only one status unknown agent is marked going in the left direction, A leaves in the left direction; if two status unknown agents are already marked going in the left direction, A leaves in the right direction.

5. When $pd < 4$ and there is only one node that is in Scenario S1, A's decision will be based on whether this node can be reached without passing through any other status unknown agent.

   - If this node can be reached by A in the same direction as the first visiting agent to this node, A leaves in the same direction;
   - If this node can only be reached in the opposite direction as the first visiting agent to this node, A leaves in the opposite direction;
   - If this node cannot be reached, A waits at hb.

6. When $pd < 4$ and there is more than one node in Scenario S1:

   - If only one of these nodes can be reached without passing through a status unknown agent, A goes to the reachable node;
   - If more than one of these nodes can be reached, A goes to the node that can be reached in the left direction.

A immediately returns to hb after reaching the destination node. Upon arriving in hb, A changes its own ? to $\surd$. If A sees a ? in the Third Agent column after changing,

A also changes the Third Agent's ? into $\times$ and writes "Last" in the repaired node column on this row and executes Procedure Black Hole Search. If A is the Third Agent returning and sees its ? has been changed, it starts executing Procedure Black Hole Search without changing the whiteboard. [1]

7.2.5 Procedure "Black Hole Search"

As detailed in Procedure Find the Meeting Node, an agent A will only start executing Procedure Black Hole Search when A sees that all nodes' statuses are known (either safe or repaired) or all nodes' statuses are known save for one. In this latter scenario, according to line 10 in Procedure Find the Meeting Node, four ?s are on that node.

Agent A continues this task by marking all repaired nodes in Table 5. A then searches the Third Agent column:

1. If there are 2 status unknown agents in this column, A wait at hb until one of them returns;

2. If there is only 1 status unknown agent, A searches this column from top to bottom until it finds an empty cell. If the empty cell is above the status unknown agent, A puts a $?(l)$ in the cell, and then goes to the node. Otherwise A leaves in the right direction after putting down a $?(r)$. A returns to hb immediately after visiting this node. It changes the ? to a $\surd$.

3. If there is only 1 status unknown agent and no empty cell in the Third Agent column, the node with the only status unknown agent is the black hole.

[1] In Algorithm Find the Meeting Node, wait at hb: the agent waits at hb until another agent returns and changes the whiteboard. A node can be reached: this node can be reached without passing a status unknown agent.

Algorithm 7 FIND THE MEETING NODE

```
1: loop
2:     search the node list from top to bottom and count pd
3:     if every node is known to be safe or repaired then
4:         execute BLACK HOLE SEARCH
5:     end if
6:     if pd > 4 or pd = 4 and the 4 ? are not on the same node then
7:         wait at hb and execute MEETING NODE
8:     else if pd = 4 and the 4 status unknown agents are on the same node then
9:         execute BLACK HOLE SEARCH
10:    else if pd < 4 and no node is in Scenario S1 then
11:        if all status unknown agents are not on the same node then
12:            if a node has both ?(l) and ?(r) then
13:                put a ? in the Third Agent column
14:            else
15:                wait at hb and execute MEETING NODE
16:            end if
17:        else if all status unknown agents are on the same node then
18:            put a ? in the Third/Fourth Agent column
19:        end if
20:        if there is only one ?(l) in total then
21:            change own ? into ?(l)
22:        else if two ?(l) in total then
23:            change own ? into ?(r)
24:        end if
25:    else if pd < 4 and only one node is in Scenario S1 then
26:        if this node can be reached in the same direction as the First Agent then
27:            go to this node in the same direction as the First Agent
28:        else if this node can be reached in the opposite direction as the First Agent
               then
29:            go to this node in the opposite direction as the First Agent
30:        else if this node cannot be reached then
31:            wait at hb and execute MEETING NODE
32:        end if
33:    else if pd < 4 and more than one node is in Scenario S1 then
34:        if only one can be reached then
35:            go to the reachable node
36:        else if more than one can be reached then
37:            go to the node that can be reached from left
38:        end if
39:    end if
40:    upon arriving, return to hb immediately
41:    if hb is reached and the cell of the Third Agent for the same node is ? then
42:        change the third ?(l/r) into ×, mark this node "Last"
43:    end if
44:    change own ?(l/r) into √(l/r)
45: end loop
```

Algorithm 8 Black Hole Search

```
1: search the repaired node list
2: if the list is blank then
3:     mark all repaired nodes in the list
4: end if
5: while the black hole has not been located do
6:     search the Third Agent column
7:     if there are 2 ? in this column then
8:         wait at hb until an agent returns
9:     else if there is 1 ? in this column then
10:        search this column from top to bottom until an empty cell is found
11:        if the empty cell is above the ? then
12:            go left to the node, upon arriving, return to hb immediately
13:        else if the empty cell is below the ? then
14:            search this column from top to bottom until an empty cell is found, go to
                   the node
15:        else if an empty cell cannot be found then
16:            the black hole is determined to be the node with ?
17:        end if
18:    else if there is no ? in this column then
19:        search this column from top to bottom until an empty cell is found, go to the
               node
20:    end if
21: end while
22: ALGORITHM TERMINATES
```

7.3 Correctness and Complexity

Lemma 13 *Minimally $b+2$ agents are necessary to repair all faulty nodes and locate the black hole in an asynchronous ring network.*

Proof. Since there are b faulty nodes in the ring network and 1 agent can only repair 1 faulty node, b agents are needed. To distinguish the black hole from the repaired nodes, at least one agent has to enter the black hole and die, thus, $b+1$ agents are required. To report the locations of the faulty nodes and the black hole, at least 1 agent has to survive, hence, $b+2$ agents are necessary. ■

Lemma 14 *There can be no more than* 4 *status unknown agents as long as at least one node is unexplored. At least* 1 *of these* 4 *agents will return to hb.*

Proof. In the homebase *hb*, as long as an agent A can find an unexplored node in the node list, it always needs to explore a new node (by executing Procedure New Node Exploration) before it executes any other procedure.

An agent A always searches the node list from top to bottom first, if one or zero agents have left in the left direction, A will also leave in the left. Otherwise it searches the node list from bottom to top. Hence, there will never be more than 2 status unknown agents leaving in the left direction. When A searches from bottom to top of the node list, A may leave in the right direction if one or zero agents have left in the right. If A finds 2 agents have left in both left and right directions, A will wait at *hb* until Table 5 is changed by a returned agent (see Line 14 and 22 in Procedure New Node Exploration). Consequently, there will never be an occasion in which any agent will leave *hb* when there are two ?s on each side of it. Hence, there cannot be more than 4 status unknown agents as long as at least one node is unexplored.

We now want to prove that at least 1 of these 4 agents will return to *hb* eventually. It is trivial to observe that all the explored nodes are in one or two consecutive

sections in a ring: when there are no unexplored nodes remaining in the ring, all explored nodes are in one consecutive section; otherwise, the two sections of explored nodes are separated at each end by *hb* and a consecutive section of unexplored nodes. We call these two sections the left part and the right part. When there are 4 status unknown agents in the network, it can only be the case that 2 are in the left part and 2 in the right part. According to our assumption that there is only one gray virus at any point in time during the execution of this algorithm and that once this *GV* infects a faulty or repaired node, it cannot move again, we know that at most one faulty node gets infected and turns into a black hole. Therefore, once the black hole appears, it can only exist either in the left part, or in the right part.

Clearly if the black hole has not appeared yet, the two second-agents (1 on each side) in both parts will return to *hb* traversing through the section of the ring with consecutive explored nodes while the two first-agents (1 on each side) may die after repairing faulty nodes. If the black hole is in the left part, the second-agent in the right part will return while the first-agent may die after repairing a faulty node and the two agents in the left part may die in the black hole. Similarly, if the black hole is in the right part, the second-agent in the left part will return while the other three may die. In summary, no matter when the black hole appears and no matter where the black hole is, as long as there is an unexplored node, at least 1 of the 4 status unknown agents will return to *hb*. ■

Lemma 15 *At most* 5 *status unknown agents coexist when there is at least one status unknown node. At least* 1 *of these* 5 *status unknown agents will return to hb.*

Proof. As mentioned in Lemma 14, when 2 status unknown agents are exploring in the same direction, they must be either aiming to explore the same node or two neighbouring nodes. More specifically, these one or two nodes must also be next to an unexplored node. That is, all status unknown agents must be consecutively located

next to either the left part or the right part of unexplored nodes section. According to Procedure New Node Exploration lines 14 and 22, only when there are fewer than 4 status unknown agents which exist in the network, a newly waking up agent will decide accordingly to go to the last unexplored node. After this agent has left, there are no more unexplored nodes in Table 5 and at most 4 status unknown agents exist in the ring at this moment. Since all previously status unknown agents can only be located at either or both edges of the left and right parts of the explored nodes, all status unknown agents should be on continuous nodes after this agent leaves for the last unexplored node.

As proven in Lemma 14, if there are 4 status unknown agents, at least 1 will return to *hb*. The same situation applies to the last unexplored node, when there are 4 status unknown agents that are not on the same node. At least 1 of them will finally return to *hb* when there is no unexplored node, given there is only one black hole.

Furthermore, according to Procedure Find The Meeting Node lines $3, 4, 10$ and 11, Procedure Black Hole Search can be executed when either all nodes' statuses are known or when 4 status unknown agents are on the same node. This latter case is where the Fifth Agent is needed in the network. In all other cases, a newly waking up agent waits at *hb*. Consequently, it will never be the case that a fifth ? shows up in the whiteboard when the 4 existing status unknown agents are on different nodes. Therefore, at most 4 status unknown agents exist when they are not on the same node.

When 4 status unknown agents are on the same node, if this node is not a black hole, at least 1 agent will return to *hb*. In Procedure Find the Meeting Node lines 27 to 40, when there is a node in Scenario S1, a second agent will go to this node, so that eventually there will be no nodes left in Scenario S1. Regardless of the direction in which the two first agents have left to a node, the third and fourth agents will take

the opposite direction (see Procedure Find the Meeting Node lines 21 to 25). This ensures that 2 agents leave in the left direction and 2 leave in the right towards the same node. Again, because there is only one black hole, at least 1 of the 4 status unknown agents will return to *hb* when this node is not a black hole.

It is possible that this node is a black hole, so that none of the 4 agents can return. Now Procedure Black Hole Search gets executed when a new agent enters the network. It goes to check each node as the fifth status unknown agent. If the node with 4 status unknown agents is the black hole, the Fifth Agent will survive because it will never enter a node with a ? in the column of Third Agent (see Lines 15 and 16 in Procedure Black Hole Search). If this node is not a black hole, the Fifth Agent may die, however, at least 1 agent on this node will survive. Therefore, at least 1 of the 5 status unknown agents will return to *hb*. ■

Lemma 16 *All faulty nodes will be repaired within finite time.*

Proof. If a faulty node N_x has not been repaired, its status shown in the whiteboard in *hb* must be either unexplored or unknown, that is the exploring agent either died after repairing a faulty node or in a back hole or has not returned to *hb* yet. If N_x is unexplored, according to Line 3 in Procedure New Node Exploration, an agent will explore N_x and any other unexplored node before it executing any other procedure that can lead to the termination of the algorithm.

If N_x is a status unknown node, it can be either in Scenario S1 or S3. When N_x is in Scenario S1, according to linse 10-13 in Procedure New Node Exploration, it is either the case that the First Agent returns to *hb* after exploring N_x and marks this node safe in the whiteboard; or a Second Agent will explore N_x and consequently change the marking in the whiteboard into Scenario S3.

When N_x is in Scenario S3, it may become S4-safe, S5-repaired, S6-safe, or stay S3-unknown. As proven in Lemma 14, there can be at most 4 status unknown agents

when they are not on the same node. In other words, at most 2 nodes may be in Scenario S3. When 2 nodes are in Scenario S3, at least one agent will return to *hb*, since there is only one black hole. This returning agent will change one of the two Scenario S3 nodes. Consequently, at most 1 node remains in Scenario S3.

For this last unknown node, a third and a fourth agent will go to this node according to Line 15 to 25 in Procedure Find the Meeting Node. As proven in Lemma 15, as long as this node is not a black hole, one of the 4 agents will return to *hb*. If this node is the black hole, it must have been a repaired node first. Therefore, we conclude that all faulty nodes will be repaired within finite time. ■

Lemma 17 *Procedure Black Hole Search locates the black hole correctly.*

Proof.

Procedure Black Hole Search gets executed in two only conditions: 1) all nodes have known status, 2) only one node is unknown and it has 4 status unknown agents. In the former case, according to Line 18 to 20 in Procedure Black Hole Search, each new agent or a newly returned (to *hb*) agent simply leaves to check each node one by one, and the last repaired node that has no agent returned is the black hole. In the latter case, the Fifth Agent is needed to continue the black hole search. As previously proven in Lemma 15, at least 1 of these 5 status unknown agents will return to *hb*. If the returning agent is this Fifth Agent, it will continue checking another node until it returns to *hb* and notices that there is only one repaired node with no agent has returned. If the returning agent is one of the 4 agents that were marked on the last status unknown node, according to Line 44 in Procedure Find the Meeting Node, the status of this unknown node becomes known and is marked "Last". Consequently, this latter case is turned into the former case, and the black hole is located. ■

Lemma 18 $b+4$ *agents suffice to repair all faulty nodes and locate the black hole in a ring network using only one whiteboard in the homebase.*

Proof. To repair $b-1$ faulty nodes, $b-1$ agents are necessary and sufficient. In the worst case, the last unknown node is the black hole and all 4 status unknown agents die in it, and one more agent is needed to perform the Procedure Black Hole Search. In other cases, as proven in Lemma 15, at least 1 of these 5 agents will return to hb and finally locate the black hole. Therefore, $(b-1)+5 = b+4$ agents suffice. ■

Lemma 19 *In an arbitrary network that contains b faulty nodes and one one-stop GV, $b+2$ agents are necessary to repair all faulty nodes and locate the black hole.*

Proof. b agents are required to repair all b faulty nodes and 1 extra agent has to die in the black hole in order to locate it, while 1 agent needs to survive and report. Therefore, $b+2$ agents are necessary to repair all faulty nodes and locate the black hole. ■

Lemma 20 *Our algorithm can locate at least $b-1$ repaired nodes when b is unknown. All repaired nodes can be located when b is known.*

Proof. Since Procedure Black Hole Search can only be executed when all nodes are known to be safe/repaired or only one node remains unknown, at the beginning of this procedure, b or $b-1$ repaired nodes can be marked in the repaired list. If only $b-1$ repaired nodes have been marked, the last unknown node will either be marked "last" by a returning agent or be located as the black hole. If it is marked "last", the node remains unknown until the algorithm terminates; if it is the black hole, it is a repaired node before it turns to be a black hole.

In the case that b is known, we can certainly know whether the "last" node is a faulty node or not by comparing b with the number r of repaired nodes. If $r = b-1$, the "last" node is a repaired node; if $r = b$, the "last" is a safe node.

■

Lemma 21 *All faulty nodes can be repaired and the black hole can be located within* $O(n^2)$ *moves.*

Proof. In the worst case, the b faulty nodes are the nodes from N_{n-1} to N_{n-b} and each node in the ring has been visited by 2 agents in Procedure New Node Exploration. Therefore, it costs $2 * 2 * (1 + 2 + 3 + 4 + \cdots + (n-1)) = 2(n-1)(n-2)$ moves. The last unknown node may be explored by 4 agents in Procedure Find the Meeting Node. Hence, at most $4 * 2(n-1)$ moves are performed. In Procedure Black Hole Search, each node needs to be visited again which costs $2(n-1)(n-2)$ moves. In total, $4 * (n-1)(n-2) + 4 * 2(n-1) = O(n^2)$ moves are needed. ■

Theorem 22 *Algorithm Dynamically Spawned Black Hole Search (DSBHS) can repair all faulty nodes and locate the black hole with* $b + 4$ *co-located agents in* $O(n^2)$ *moves using only one whiteboard in the homebase.*

7.4 Simulation Results

In this section, we present the experimental results obtained from a series of Java simulations of this algorithm. The experiment is done in a ring network with only one whiteboard in the homebase node which can only be accessed when the agents are in the homebase. All agents start from this homebase and execute the same protocol as described above. For the number of faulty nodes, we use a variable ($faulty\_posb$) to present the possibility that whether or not a node in the experimental network is faulty. This possibility varies with $20\%, 30\%$, and 40%. Thus, at the beginning of the exploration, the agents do not know the number of faulty nodes or their locations. In addition to the possibility of a node is faulty or normal, whether a repaired node becomes a black hole is also under a certain percentage. This follows the behaviour of the GV that a repaired node can be infected at any time during the exploration. Hence, even ourselves do not know the location of the black hole until it is generated.

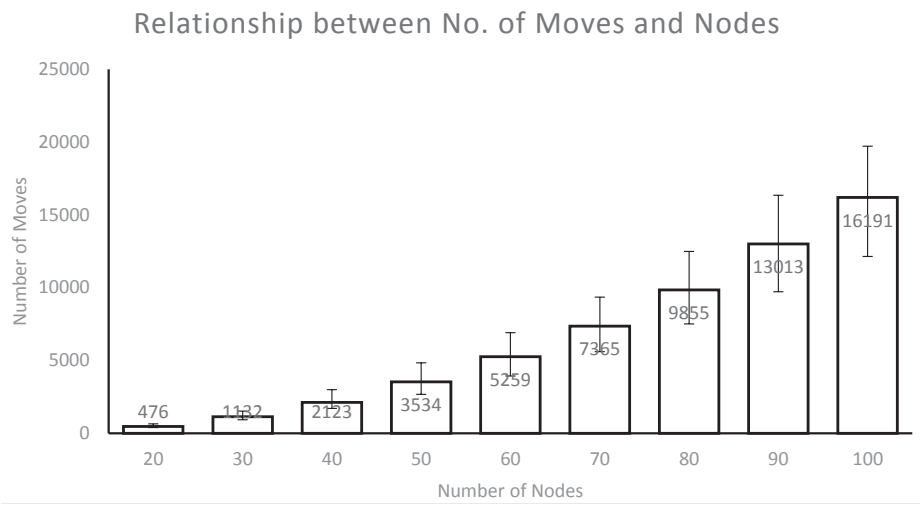

Figure 11: The Relationship between Number of Moves and Nodes

To make the implementation more realistic, the distance of each link between two neighbouring nodes are randomly designed, which is more similar to the feature of an asynchronous network; that is, the time an agent spends on a link is unpredictable but finite. The implementation has a task scheduler, which will wake up a sleeping agent after a certain amount of time. However, since there is a possibility that the agent becomes active or keep sleep, the task scheduler may fail to wake up a sleeping agent. This makes the simulation obey the rule that the time an agent sleeps is unpredictable in an asynchronous network.

Our simulation sets consist of 20 to 100-node networks. The execution of a simulation is considered to be successful if the location of the black hole is correctly marked on the homebase whiteboard. Otherwise, the simulation is counted as a failure. Since our algorithm guarantee to correctly locate the black hole, there is no such failure in all executions. For each successful simulation, we count the total number of moves that are used to repair all faulty nodes and locate the black hole. All data is calculated from 100 independent successful runs of each setting with random generated faulty nodes and a black hole. For each $faulty\_posb = 20\%, 30\%, 40\%$ and

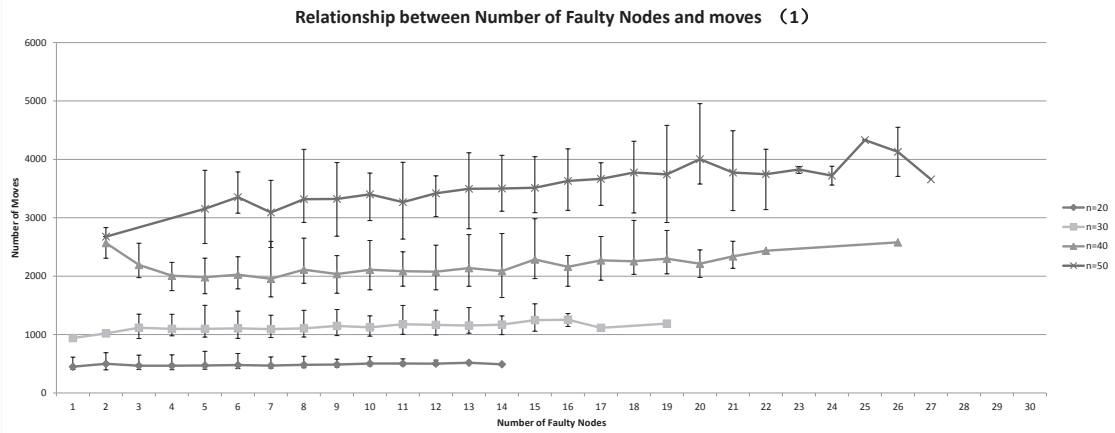

Figure 12: The Relationship between Number of Moves and Faulty Nodes

$n = 20, 30, 40, ..., 90, 100$, we provide 100 independent runs which are 2700 runs in total. All these results show that $b + 4$ agents are sufficient to finish the repair and search task. Additionally, there is a 14.8% possibility that the task can be finished using only $b + 3$ agents or fewer.

Figure 11 illustrates the average move results as well as the lower and upper bound of the total number of moves for each setting. Results confirm that $O(n^2)$ moves suffice to repair all faulty nodes and locate the black hole in all simulations. It is obviously shown in Figure 11 that the larger the network is, the more moves are necessary for the task to complete.

We further analyze whether the number of faulty nodes will affect the number of moves. Figures 12 and 13 show that as the number of faulty nodes increases, the total number of moves also has a slight increase. However, the same as above mentioned, the increase is not continuous and obvious. Thus, we can only conclude the number of faulty nodes does not have a direct effect to the number of moves.

The theoretical analysis and simulation results can both prove that, all faulty nodes can be repaired and the black hole can be located with $b + 4$ agents in $O(n^2)$ moves using only one whiteboard in the homebase. Furthermore, this simulation

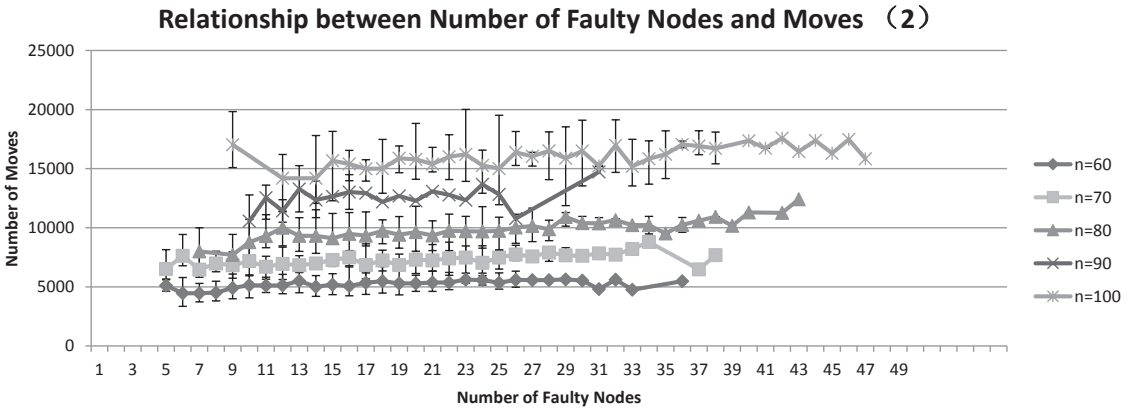

Figure 13: The Relationship between Number of Moves and Faulty Nodes

study can further prove the correctness of the algorithm. It should be noticed that, our algorithm requires no knowledge of the number b of faulty nodes. Specifically, when b is known, our algorithm can additionally locate all repaired nodes.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In Chapter 1, we first investigate the state of the art of the black hole search problem using mobile agents under various network setups. We then introduce a comprehensive list of assumptions that are frequently used in different black hole search studies in Chapter 2. These assumptions are network synchronization, including synchronous and asynchronous networks, communication models, agent starting locations, and topological knowledge. Such a complete list of assumptions captures all facets of variability in black hole search. We also study the complexity measures and evaluation methods of all surveyed research studies.

In Chapter 3, we classify all the selected studies into two categories according to their network synchronization model, namely, black hole search in synchronous networks and in asynchronous networks. We analyze and compare in detail the studies under each of these categories and conclude the impact of each assumption on the solutions and their cost complexity. We further discuss multiple black hole search, and additionally introduce some other types of malicious hosts.

Following the literature review, the models and assumptions for the algorithms studied in this paper are introduced in Chapter 4. We then present a new attack model

containing both faulty nodes and gray virus that can repeatedly infect a repaired faulty node in Chapter 5. We introduce the faulty node repair and search problem caused by a multi-stop GV using the enhanced token model, present an algorithm that can repair all the faulty nodes and locate all the possible locations of the GV, and construct a map of the unknown network with all repaired nodes marked. It is important to note that in an asynchronous network, the speed of a GV is unknown and could be much faster than the speed of the mobile agents. Consequently, it could be the case that all vulnerable nodes (previously repaired faulty nodes) appear to be black holes to the mobile agents. Even with enough mobile agents, it is impossible to locate all black holes a within finite time since we cannot differentiate the case that an agent died in a black hole or was just stuck in a slow link. We further prove the correctness of the algorithm and offer its complexity analysis. Our algorithm shows that $b + 1$ agents are necessary and sufficient to repair and locate b faulty nodes in an arbitrary unknown network within $O(kn\Delta)$ moves.

Also using the token model, we then present an algorithm that can repair all faulty nodes and locate a black hole that is infected by a one-stop GV in a ring network. We further analyze how the appearance of the GV changes the network behaviour; that is, one of the previously visited nodes, unexplored nodes and even the the current node of an agent may suddenly be infected by a GV and turned into a black hole. This drastic and dramatic network behaviour change significantly increases the difficulty and complexity of the solution to the traditional black hole search that contained a black hole among the unexplored nodes since the start. We introduce the new technique of double cautious walk with tokens, which can mark both the previously visited and unexplored dangerous nodes. This technique guarantees the minimal loss of mobile agents. We then present solutions for the problem caused by this one-stop GV in an asynchronous ring network. We conclude that $b + 9$ agents can complete the repair and search task in $O(kn^2)$ moves.

After discussing the solutions with tokens, in Chapter 7, we offer a solution to the problem caused by this one-stop GV in an asynchronous ring network with only one whiteboard in the entire network. Our algorithm even works when the number of faulty nodes are not known a priori. We first theoretically analyze the correctness of this algorithm and offer simulation results to further prove that, using $O(n^2)$ moves, $b+4$ agents can repair all faulty nodes as well as locate the black hole that is infected by a one-stop GV.

Finally, after investigating possible combinations of the list of assumptions that we have collected, we also identify some potential open research issues in this field.

8.2 Further Analysis and Future Work

In this section, we analyze all the reviewed studies and highlight some future work for:

- single black hole search in both synchronous network (Section 8.2.1) and asynchronous network (Section 8.2.1) and
- multiple black hole search (Section 8.2.2) and
- black hole search using different types of agents (Section 8.2.3) and
- faulty node repair and black hole search for one-stop and multi-stop GVs (Section 8.2.4).

8.2.1 Further Analysis and Future work for Single Black Hole Search

In this subsection, we list all possible combinations of different assumptions and organize all the single black hole search studies under each such combination. The results

can be found in Tables 8 and 9. We then further analyze the remaining combinations/open problems not yet studied and identify some future research possibilities.

Single black hole search in Synchronous Networks

Single black hole search in synchronous networks is not studied as often as in asynchronous networks, which is a more realistic model. As shown in Table 8, the pure token model is only used with dispersed agents (Combinations $1-3$ in Table 8). These 4 papers only study the minimal number of agents and tokens required to solve the black hole search problem without offering any specific algorithm or complexity analysis. Hence, finding the agent moves as well as the time cost can be further studied. Additionally, only ring and torus have been studied under the pure token model with a focus on the number of agents and tokens used. Studying the problem in other topologies, such as hypercube or mesh, needs to be further studied. We have observed from the results for black hole search in asynchronous networks, when agents are co-located, it always costs fewer moves than the scenarios when the agents are dispersed. Studying the problem using co-located agents under the pure token model in a synchronous network will further prove this observation. In asynchronous arbitrary networks, it is proven that the pure token model can keep the same complexity with the whiteboard model while having a network map. Whether it is the same in synchronous networks also needs to be studied.

One of the several advantages of solving the black hole search problem in synchronous networks versus in asynchronous networks is the possibility of using face-to-face communication. The combinations of face-to-face communication and the use of whiteboards or tokens usually lead to further reduction on both time costs and agent moves. For example, [16, 17] study both oriented and un-oriented rings without knowing the network size under the pure token model with the assumption that the agents can also communicate with each other when they meet. However, in the

Table 8: Existing work on black hole search in synchronous networks. PT: pure token; FTF: face-to-face; CL: co-located; DIS: dispersed

Combination	Communication model	Agent starting location	Network knowledge	Paper
1	PT	DIS	Unknown n, oriented torus	[15]
2	PT	DIS	Unknown n, unoriented torus	[66]
3	FTF + PT	DIS	Unknown n, oriented or unoriented ring	[16, 17]
4	FTF	CL	tree	[23, 25]
5	FTF	CL	Complete Knowledge	[24, 58–60]

face-to-face model, the agents leave no marks on nodes. Consequently, it is possible that all dispersed agents could die in the black hole before they even meet with each other. Therefore, it is of very little interest to study the problem using the dispersed agents with face-to-face communication.

Czyzowicz *et al.* [25] study the black hole search in synchronous networks without using network maps. They tackled the problem in tree networks using the face-to-face model and the co-located agents. There is thus a need for further investigation of the minimum time complexity required for other commonly studied topologies, such as hypercube, or complete network. Finding the time optimal algorithm for an arbitrary unknown graph using co-located agents and only the face-to-face communication should be studied. Additionally, in some conditions, face-to-face communication may lead to higher number of agent moves. For example, let 2 co-located agents explore the ring network in different directions. In the case of letting them communicate with each other (about their partial exploration results) face-to-face, the agents must

come back to the homebase after every move; in the token models, the agents' last safe location is marked by token(s) (e.g. using cautious walk). Agents can first divide the ring into 2 parts, then each explore one part. After $\frac{n}{2}$ time steps the survived agent goes to look for the other and find out the location of the black hole. Hence, studying the problem under the same setup instead using the whiteboard and token models is also of interest.

Single Black Hole Search in Asynchronous Networks

We do not include edge-labelling in our discussion because it is widely adopted in the field. Since network size must be known a priori in asynchronous networks, we do not further mention n in this subsection. Again, when using the co-located agents to explore a ring, whether or not the ring is oriented does not affect the move cost of the algorithm. Therefore, we also do not discuss them separately here. As ring is a special topology, the sparsest bi-connected graph, we list it separately in Table 9.

Glaus *et al.* [53] study the black hole search problem without the knowledge of incoming links in an unknown un-oriented arbitrary network. Under these assumptions, Glaus *et al.* solve the problem when both the agents and the network nodes have distinct IDs. Whether under the same assumptions with an anonymous network and agents, the *Repair and Search* problem is still solvable remains to be an open problem. Furthermore, if we change an asynchronous network to synchronous, whiteboard to tokens, or dispersed agents to co-located, the nature of the complexity is another issue. Balamohan *et al.* identify an open problem in their paper [7]: whether or not an algorithm that uses $n-1$ co-located agents (using the whiteboard model) to locate the black hole in $\frac{3}{2}n - O(1)$ time (average case) and $2(n-1)$ time (the worst case) exists.

Dobrev *et al.* consider a very difficult condition, namely, no topology knowledge is assumed in [30] (Combination 8 in Table 9), and their algorithm locates the black

hole in $O(\Delta^2 M^2 n^5)$, using the enhanced token model with $\Delta + 1$ co-located agents. Under the same conditions, solving the problem in the whiteboard model only costs $\Theta(n^2)$ moves. The question raised is whether we can find a solution for the problem using the enhanced token model without any topology knowledge with a lower move cost. In addition, relaxing the topology knowledge assumption (Combinations $9 - 10$ in Table 9) may also help reduce the move cost when solving the proposed problem. Moreover, only ring, hypercube, torus, and complete network have been studied under the enhanced token model. Studying the problem under the same assumptions in other topologies is another future direction.

The pure token model is introduced and studied in [5, 43, 44]. Using two agents [43,44] can locate the black hole in $\Theta(n \log n)$ moves in an arbitrary network with the help of network maps. Whether increasing the number of agents can further reduce the moves or not, whether known a special topology the moves can be reduced, and whether the problem is solvable in an arbitrary unknown graph can also be studied in the future. Furthermore, as the pure token model is only studied using co-located agents, the case of dispersed agents remains an open problem.

8.2.2 Future Work for Multiple Black Hole Search

For the multiple black hole search problem, Cooper *et al.* [21] and D'Emidio *et al.* [26] solve a weaker black hole model where an agent can repair a black hole by sacrificing its life. In these two papers, the authors also assume that each agent carries a network map of the given synchronous network. An interesting direction for further research is to consider the case when the topology of the network is not known or only partially known in a priori. This new black hole repair model also opens research avenues to asynchronous networks. Furthermore, in asynchronous networks, this repairing problem should be studied in the whiteboard model or the two types of token models.

D'Emidio *et al.* [26] assume that if an agent enters and repairs a black hole, all

agents within a distance r of the black hole will disappear along with it. The authors only solve the problem when $r = 0$ and $r = 1$, thus, the case $r > 1$ could also be a future direction.

Flocchini *et al.* [48] solve the black hole search problem in a *subway model* in an asynchronous network using the whiteboard model and co-located agents. These initial results prompt us to think what would be the impact on the complexities of the solutions if we change the whiteboard model to other communication models and/or change the asynchronous network to synchronous. For example, in the subway system, it usually takes the same amount of time for a carrier to travel between two stations. This can be modelled as a synchronous network. In terms of communication, assuming there is no whiteboard available and the cell phone network does not work in subways, the agents can only communicate with each other when they meet in the same subway station or in the same carrier. Even in the case that each of the agents has an available WalkyTalky, which functions only when two are within a short distance, it can still be seen as a face-to-face model. In addition, [48] considers a directed graph and measures the carrier moves which are different from the agent moves. Hence, whether an undirected graph will help to reduce the carrier move complexity, as well as the total move cost of the agents both remain open problems.

8.2.3 Open Problems with Different Types of Agents

After discussing the different possible combinations of the assumptions on the black hole search problem, we now go through a different type of agent that can be used to solve the problem. Like the memory consumption in each network node, the mobile agents should also have a memory limitation. Usually, the agents are endowed with unlimited memory so that they can carry a network map or build a map during the network exploration. An agent endowed with very limited memory has been introduced by Flocchini *et al.* [42]. An agent is referred to as being oblivious when all the

information contained in the workspace is cleared at the end of each computing cycle. In other words, the agents are memoryless, having no memory of any past actions and computations. Each computation is based solely on what has been determined in the current computing cycle. For example, the agents used in [42] are oblivious and asynchronous. They can view the environment but are unable to communicate with each other. Using such oblivious agents should be an interesting direction that requires further investigation. Obviously, the absence of memory in each agent can be compensated by using a whiteboard or tokens.

8.2.4 Future Work for One-stop and Multi-stop GV

As we mentioned in the observations, without any additional assumptions, the *Repair and Search* problem becomes a MBHS problem and remains unsolvable in an asynchronous network. As future work, instead of assuming that the GV appears only after all the repaired node have been located, other assumptions can also be added to make the problem solvable in the presence of multi-stop gray virus in an arbitrary unknown network topology. For example, a multi-stop GV can only move to another node after deleting at least one agent instead of being able to move at any point in time. Furthermore, if one or more GVs may show up at any time, what would be the impact of one-stop GVs vs. the multi-stop ones? In addition, solving the problem in a synchronous network could also be a future direction.

For the one-stop GV, apart from ring networks, other common networks or arbitrary networks could also be a further concern. As in our algorithms, it is possible that 2 agents enter the same faulty node and die simultaneously. An algorithm that can avoid this problem could also reduce the number of agents used. Also, apart from the enhanced token and whiteboard model, other communication models such as face-to-face or pure token may also be considered.

Table 9: Existing work on black hole search in asynchronous networks. WB: whiteboard; ET: enhanced token; SD: sense of direction; NT: network topology

Combination	Communication model	Agent starting location	Network knowledge	Paper
1	WB	CL	No knowledge	[22, 32, 34, 53]
2	WB	CL	NT	[27, 28]
3	WB	CL	SD and no NT	[32, 34]
4	WB	CL	ring	[6, 7, 31, 35]
5	WB	CL	Complete Knowledge	[27, 28, 32, 34, 36, 37]
6	WB	DIS	un-oriented ring	[31, 33, 35]
7	WB	DIS	oriented ring	[31, 33, 35]
8	ET	CL	No Knowledge	[29, 30]
9	ET	CL	oriented ring	[38]
10	ET	CL	SD and NT	[68, 69]
11	ET	DIS	SD and NT	[68, 69]
12	ET	DIS	oriented ring	[40]
13	ET	DIS	un-oriented ring	[39, 41]
14	PT	CL	ring	[43, 44]
15	PT	CL	Complete Knowledge	[43, 44]
16	PT	CL	No Knowledge	[5]

References

[1] Agrawal, P., Ghosh, R. K., and Das, S. K. Cooperative black and gray hole attacks in mobile ad hoc networks. In *Proceedings of the 2Nd International Conference on Ubiquitous Information Management and Communication* (New York, NY, USA, 2008), ICUIMC '08, ACM, pp. 310–314.

[2] Al-Shurman, M., Yoo, S.-M., and Park, S. Black hole attack in mobile ad hoc networks. In *Proceedings of the 42nd annual Southeast regional conference* (2004), ACM, pp. 96–97.

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

[4] Almorsy, M., Grundy, J., and Müller, I. An analysis of the cloud computing security problem. In *Proceedings of APSEC 2010 Cloud Workshop, Sydney, Australia, 30th Nov* (2010).

[5] Balamohan, B., Dobrev, S., Flocchini, P., and Santoro, N. Asynchronous exploration of an unknown anonymous dangerous graph with o (1) pebbles. In *Proceedings of the 19th International Conference on Structural Information and Communication Complexity* (2012), SIROCCO'12, Springer, pp. 279–290.

[6] Balamohan, B., Flocchini, P., Miri, A., and Santoro, N. Improving the optimal bounds for black hole search in rings. In *Proceedings of the 18th International Conference on Structural Information and Communication Complexity* (2011), SIROCCO'11, Springer, pp. 198–209.

[7] Balamohan, B., Flocchini, P., Miri, A., and Santoro, N. Time optimal algorithms for black hole search in rings. *Discrete Mathematics, Algorithms and Applications 3*, 04 (2011), 457–471.

[8] Banerjee, S. Detection/removal of cooperative black and gray hole attack in mobile ad-hoc networks. In *proceedings of the world congress on engineering and computer science* (2008), pp. 22–24.

[9] Banerjee, S., Sardar, M., and Majumder, K. Aodv based black-hole attack mitigation in manet. In *Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2013* (2014), Springer, pp. 345–352.

[10] Barrière, L., Flocchini, P., Fraigniaud, P., and Santoro, N. Capture of an intruder by mobile agents. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures* (2002), ACM, pp. 200–209.

[11] Bender, M. A., Fernández, A., Ron, D., Sahai, A., and Vadhan, S. The power of a pebble: Exploring and mapping directed graphs. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing* (1998), ACM, pp. 269–278.

[12] Bender, M. A., and Slonim, D. K. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on* (1994), IEEE, pp. 75–85.

[13] Blin, L., Fraigniaud, P., Nisse, N., and Vial, S. Distributed chasing of network intruders. In *Structural Information and Communication Complexity*. Springer, 2006, pp. 70–84.

[14] Cai, J., Flocchini, P., and Santoro, N. Network decontamination from a black virus. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International* (2013), IEEE, pp. 696–705.

[15] Chalopin, J., Das, S., Labourel, A., and Markou, E. Black hole search with finite automata scattered in a synchronous torus. In *Proceedings of the 25th International Conference on Distributed Computing* (Berlin, Heidelberg, 2011), DISC'11, Springer-Verlag, pp. 432–446.

[16] Chalopin, J., Das, S., Labourel, A., and Markou, E. Tight bounds for scattered black hole search in a ring. In *Proceedings of the 18th International Conference on Structural Information and Communication Complexity* (Berlin, Heidelberg, 2011), SIROCCO'11, Springer-Verlag, pp. 186–197.

[17] Chalopin, J., Das, S., Labourel, A., and Markou, E. Tight bounds for black hole search with scattered agents in synchronous rings. *Theoretical Computer Science* (2013).

[18] Chalopin, J., Das, S., and Santoro, N. Rendezvous of mobile agents in unknown graphs with faulty links. In *Proceedings of the 21st International Conference on Distributed Computing* (Berlin, Heidelberg, 2007), DISC'07, Springer-Verlag, pp. 108–122.

[19] Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., and Molina, J. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security* (2009), ACM, pp. 85–90.

[20] Cooper, C., Klasing, R., and Radzik, T. Searching for black-hole faults in a network using multiple agents. In *Proceedings of the 10th International Conference on Principles of Distributed Systems* (Berlin, Heidelberg, 2006), OPODIS'06, Springer-Verlag, pp. 320–332.

[21] Cooper, C., Klasing, R., and Radzik, T. Locating and repairing faults in a network with mobile agents. *Theoretical Computer Science 411*, 14-15 (Mar. 2010), 1638–1647.

[22] Czyzowicz, J., Dobrev, S., Královič, R., Miklík, S., and Pardubská, D. Black hole search in directed graphs. In *Proceedings of the 16th International Conference on Structural Information and Communication Complexity* (2010), SIROCCO'09, Springer, pp. 182–194.

[23] Czyzowicz, J., Kowalski, D., Markou, E., and Pelc, A. Searching for a black hole in tree networks. In *Proceedings of the 8th International Conference on Principles of Distributed Systems* (Berlin, Heidelberg, 2005), OPODIS'04, Springer-Verlag, pp. 67–80.

[24] Czyzowicz, J., Kowalski, D., Markou, E., and Pelc, A. Complexity of searching for a black hole. *Fundamenta Informaticae 71*, 2,3 (Aug. 2006), 229–242.

[25] Czyzowicz, J., Kowalski, D., Markou, E., and Pelc, A. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability & Computing 16*, 4 (July 2007), 595–619.

[26] D'Emidio, M., Frigioni, D., and Navarra, A. Exploring and making safe dangerous networks using mobile entities. In *Ad-hoc, Mobile, and Wireless Network*. Springer, 2013, pp. 136–147.

[27] Dobrev, S., Flocchini, P., Kralovic, R., Prencipe, G., Ruzicka, P., and Santoro, N. Black hole search by mobile agents in hypercubes and related networks. In *OPODIS* (2002), vol. 3, pp. 169–180.

[28] Dobrev, S., Flocchini, P., Královič, R., Ružička, P., Prencipe, G., and Santoro, N. Black hole search in common interconnection networks. *Networks 47*, 2 (2006), 61–71.

[29] Dobrev, S., Flocchini, P., Královič, R., and Santoro, N. Exploring an unknown graph to locate a black hole using tokens. In *Fourth IFIP International Conference on Theoretical Computer Science-TCS 2006* (2006), Springer, pp. 131–150.

[30] Dobrev, S., Flocchini, P., Královič, R., and Santoro, N. Exploring an unknown dangerous graph using tokens. *Theoretical Computer Science 472* (2013), 28–45.

[31] Dobrev, S., Flocchini, P., Prencipe, G., and Santoro, N. Mobile search for a black hole in an anonymous ring. In *Proceedings of the 15th International Conference on Distributed Computing* (London, UK, UK, 2001), DISC '01, Springer-Verlag, pp. 166–179.

[32] Dobrev, S., Flocchini, P., Prencipe, G., and Santoro, N. Searching for a black hole in arbitrary networks: Optimal mobile agent protocols. In *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing* (New York, NY, USA, 2002), PODC '02, ACM, pp. 153–162.

[33] Dobrev, S., Flocchini, P., Prencipe, G., and Santoro, N. Multiple agents rendezvous in a ring in spite of a black hole. In *Proceedings of the 7th International Conference on Principles of Distributed Systems* (2004), Springer, pp. 34–46.

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

[35] Dobrev, S., Flocchini, P., Prencipe, G., and Santoro, N. Mobile search for a black hole in an anonymous ring. *Algorithmica 48*, 1 (Mar. 2007), 67–90.

[36] Dobrev, S., Flocchini, P., and Santoro, N. Improved bounds for optimal black hole search with a network map. In *Proceedings of the 11th International*

Colloquium on Structural Information and Communication Complexity (2004), Springer, pp. 111–122.

[37] Dobrev, S., Flocchini, P., and Santoro, N. Cycling through a dangerous network: a simple efficient strategy for black hole search. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems* (Washington, DC, USA, 2006), ICDCS '06, IEEE Computer Society, pp. 57–57.

[38] Dobrev, S., Královič, R., Santoro, N., and Shi, W. Black hole search in asynchronous rings using tokens. In *The 6th Conference on Algorithms and Complexity (CIAC '06)* (2006), Springer, pp. 139–150.

[39] Dobrev, S., Santoro, N., and Shi, W. Locating a black hole in an un-oriented ring using tokens: The case of scattered agents. In *The 13th International Euro-Par Conference European Conference on Parallel and Distributed Computing (Euro-Par 2007)* (2007), Springer, pp. 608–617.

[40] Dobrev, S., Santoro, N., and Shi, W. Scattered black hole search in an oriented ring using tokens. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS-APDCM 2007)* (2007), IEEE, pp. 1–8.

[41] Dobrev, S., Santoro, N., and Shi, W. Using scattered mobile agents to locate a black hole in an un-oriented ring with tokens. *International Journal of Foundations of Computer Science 19*, 06 (2008), 1355–1372.

[42] Flocchini, P., Ilcinkas, D., Pelc, A., and Santoro, N. Computing without communicating: Ring exploration by asynchronous oblivious robots. In *Proceedings of the 11th International Conference on Principles of Distributed Systems* (Berlin, Heidelberg, 2007), OPODIS'07, Springer-Verlag, pp. 105–118.

[43] Flocchini, P., Ilcinkas, D., and Santoro, N. Ping pong in dangerous graphs: Optimal black hole search with pure tokens. In *Proceedings of the 22nd International Symposium on Distributed Computing* (2008), Springer, pp. 227–241.

[44] Flocchini, P., Ilcinkas, D., and Santoro, N. Ping pong in dangerous graphs: Optimal black hole search with pebbles. *Algorithmica 62*, 3-4 (2012), 1006–1033.

[45] Flocchini, P., Kellett, M., Mason, P. C., and Santoro, N. Mapping an unfriendly subway system. In *Proceedings of the 5th International Conference on Fun with Algorithms* (2010), Springer, pp. 190–201.

[46] Flocchini, P., Kellett, M., Mason, P. C., and Santoro, N. Fault-tolerant exploration of an unknown dangerous graph by scattered agents. In *Proceedings of the 14th International Conference on Stabilization, Safety, and Security of Distributed Systems* (2012), SSS'12, Springer, pp. 299–313.

[47] Flocchini, P., Kellett, M., Mason, P. C., and Santoro, N. Finding good coffee in paris. In *Fun with Algorithms* (2012), Springer, pp. 154–165.

[48] Flocchini, P., Kellett, M., Mason, P. C., and Santoro, N. Searching for black holes in subways. *Theory of Computing Systems 50*, 1 (2012), 158–184.

[49] Flocchini, P., Luccio, F. L., and Song, L. X. Size optimal strategies for capturing an intruder in mesh networks. In *Communications in Computing* (2005), pp. 200–206.

[50] Flocchini, P., Mans, B., and Santoro, N. Sense of direction in distributed computing. *Theoretical Computer Science 291*, 1 (2003), 29–53.

[51] Flocchini, P., and Santoro, N. Distributed security algorithms by mobile agents. In *Distributed Computing and Networking*. Springer, 2006, pp. 1–14.

[52] Fraigniaud, P., and Ilcinkas, D. Digraphs exploration with little memory. In *STACS 2004*. Springer, 2004, pp. 246–257.

[53] Glaus, P. Locating a black hole without the knowledge of incoming link. In *Algorithmic Aspects of Wireless Sensor Networks*. Springer, 2009, pp. 128–138.

[54] Greenberg, M. S., Byington, J. C., and Harper, D. G. Mobile agents and security. *Communications Magazine, IEEE 36*, 7 (1998), 76–85.

[55] Grobauer, B., Walloschek, T., and Stocker, E. Understanding cloud computing vulnerabilities. *Security & privacy, IEEE 9*, 2 (2011), 50–57.

[56] Hashizume, K., Rosado, D. G., Fernández-Medina, E., and Fernandez, E. B. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications 4*, 1 (2013), 1–13.

[57] Khari, M., et al. Mobile ad hoc netwoks security attacks and secured routing protocols: A survey. In *Advances in Computer Science and Information Technology. Networks and Communications*. Springer, 2012, pp. 119–124.

[58] Klasing, R., Markou, E., Radzik, T., and Sarracco, F. Hardness and approximation results for black hole search in arbitrary graphs. In *Proceedings of*

the 12th International Conference on Structural Information and Communication Complexity (2005), SIROCCO'05, Springer, pp. 200–215.

[59] Klasing, R., Markou, E., Radzik, T., and Sarracco, F. Hardness and approximation results for black hole search in arbitrary networks. *Theoretical Computer Science 384*, 2 (2007), 201–221.

[60] Klasing, R., Markou, E., Radzik, T., and Sarracco, F. Approximation bounds for black hole search problems. *Networks 52*, 4 (2008), 216–226.

[61] Kosowski, A., Navarra, A., and Pinotti, C. M. Synchronization helps robots to detect black holes in directed graphs. In *Proceedings of the 13th International Conference on Principles of Distributed Systems* (2009), Springer, pp. 86–98.

[62] Kosowski, A., Navarra, A., and Pinotti, C. M. Synchronous black hole search in directed graphs. *Theoretical Computer Science 412*, 41 (2011), 5752–5759.

[63] Královič, R., and Miklík, S. Periodic data retrieval problem in rings containing a malicious host. In *Proceedings of the 17th International Conference on Structural Information and Communication Complexity* (2010), SIROCCO'10, Springer, pp. 157–167.

[64] Lange, D. B., and Oshima, M. Seven good reasons for mobile agents. *Communications of the ACM 42*, 3 (1999), 88–89.

[65] Markou, E., et al. Identifying hostile nodes in networks using mobile agents. *Bulletin of the EATCS*, 108 (2012), 93–129.

[66] Markou, E., and Paquette, M. Black hole search and exploration in unoriented tori with synchronous scattered finite automata. In *Proceedings of the 16th International Conference on Principles of Distributed Systems* (2012), Springer, pp. 239–253.

[67] Sen, J., Chandra, M. G., Harihara, S., Reddy, H., and Balamuralidhar, P. A mechanism for detection of gray hole attack in mobile ad hoc networks. In *Information, Communications & Signal Processing, 2007 6th International Conference on* (2007), IEEE, pp. 1–5.

[68] Shi, W. Black hole search with tokens in interconnected networks. In *The 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2009)* (2009), Springer, pp. 670–682.

[69] Shi, W., Garcia-Alfaro, J., and Corriveau, J.-P. Searching for a black hole in interconnected networks using mobile agents and tokens. *Journal of Parallel and Distributed Computing 74*, 1 (2014), 1945–1958.

[70] Shkapenyuk, V., and Suel, T. Design and implementation of a high-performance distributed web crawler. In *Data Engineering, 2002. Proceedings. 18th International Conference on* (2002), IEEE, pp. 357–368.

[71] Su, M.-Y. Prevention of selective black hole attacks on mobile ad hoc networks through intrusion detection systems. *Computer Communications 34*, 1 (2011), 107–117.

[72] Szor, P. *The art of computer virus research and defense*. Pearson Education, 2005.

[73] Zarrad, A., and Daadaa, Y. A review of computation solutions by mobile agents in an unsafe environment. *International Journal of Advanced Computer Science & Applications 4*, 4 (2013).