NETWORK DECONTAMINATION FROM BLACK VIRUSES

by

Jie Cai

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

School of Computer Science

Carleton University

Ottawa, Ontario

Abstract

In this thesis, the problem of decontaminating networks from *Black Viruses* (BVs) using a team of system mobile agents, i.e., the BVD problem, is investigated. The BV is a dynamic harmful process which, like the extensively studied black hole (BH), destroys any agent arriving at the network site where it resides; when that occurs, unlike a black hole which is static by definition, a BV moves, spreading to all the neighbouring sites, thus increasing its presence in the network. The initial location of BV is unknown a priori. The objective is to permanently remove any presence of the BV from the network with minimum number of site infections (and thus casualties) and prevent any previously decontaminated node from becoming infected again.

The BVD problem is first studied in the systems with only one BV. Initial investigations are for some common classes of interconnection networks: (multidimensional) *grids*, *tori*, and *hypercubes*. Optimal solutions are proposed and their complexities are analyzed in terms of node infections, agent team size, and movements. After understanding the basic properties of the decontamination process in these special graphs, the BVD problem is studied in arbitrary networks. Finally research is extended to Multiple BV Decontamination problem (MBVD) both in arbitrary graphs and in special topologies.

To help understand the behavior of the protocol developed and support complexity analysis, an experimental study is performed using the simulator for reactive distributed algorithms DisJ. A large number of simulations are carried out on various sizes of graphs with many connectivity densities. The simulation runs show that the propose protocol beats random search; they also disclose many interesting behaviors, and validate the analytical complexity results. The simulation results also provide

deep understanding on the influence of graph connectivity density and graph size on complexities, i.e., movement, time, and agent size.

Finally conclusion remarks are presented and future researches are proposed.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisors Professors Nicola Santoro and Paola Flocchini for their enthusiastic motivation, judicious and enlightening advices, and patient guidance throughout this research work! Their passion for distributed computing, dedication to knowledge acquiring and science developing, and an unrelenting attitude to detail all inspire me. On the other hand, their modest academic style creates an excellent research environment. They carefully selected my courses and topics, and at the same time, provided me enough flexibility to suit my work and life. Our meetings and discussions always take away their precious lunch-break time... Without their continuous support of my Ph.D. study and related research, it would not be possible for me to complete this work.

Last but not the least, I would like to thank my wife, Yan Gong, and my daughter, Jessica Cai. Working full-time and supporting a family, and at the same time, studying and doing research part-time for my Ph.D. degree are definitely challenge to me. Without my family's understanding, encouragement, support, and sacrifices, I would not achieve this. I also like to thank my parents for their life-time inspiration and encouragement.

Table of Contents

List of Figures

Abbreviations

API Application Programming Interface

BH Black Hole

BHS Black Hole Search

BV Black Virus

BVC Black Virus Clone

BVD Black Virus Decontamination

DisJ Distributed Algorithm Simulation Java

EA Exploring Agent

IC Intruder Capture

IZ Impacting Zone

LC Locality of Casualty

LEA Leader Exploring Agent

MBVD Multiple Black Virus Decontamination

MF MBV Decontamination - Fertile

MS MBV Decontamination - Sterile

SA Shadowing Agent

SD Standard Deviation

Chapter 1

Introduction

This thesis deals with distributed computing by mobile agents in networks. In particular, we investigate the problem of deploying a team of mobile agents to explore the network and decontaminate one or more dangerous viruses (called Black Virus) present on network nodes.

In this chapter, the motivations of the problem are provided, followed by a brief summary of the contributions. Afterwards, a short description of the contents in each chapter is given.

1.1 Problem and Motivation

Mobile agents are widely used in distributed and networked systems; however, the applications of mobile agents can cause security issues and threats to the networks. Two main security issues popularly studied recently are categorized as *harmful agent* and *harmful host* [52]. In particular, a malicious agent can cause computer nodes to malfunction or crash by contaminating or infecting them; additionally, a contaminated or infected host can destroy working agents for various malicious purposes.

The harmful hosts, often called *Black Holes* (BH), are network nodes infected by a process that destroys any incoming agent without leaving any detectable trace. In this situation, the main focus is to locate their positions, because this is the first step to solve the issue. Indeed, the problem of *Black Hole Search* (BHS), has been extensively studied in many variants, e.g., different topologies (special or arbitrary

networks), settings (synchronous and asynchronous), agent distribution (co-located or dispersed), communicating mechanisms, and etc. More detailed literature review will be provided in Chapter 2. BH is harmful to agents but it is *static*, that is, it does not propagate in the network and so it is not harmful to other sites. The number of BHs does not increase or decrease during the search.

The theoretical work related to harmful agents has focused on the problem called *Intruder Capture* (IC) (also known as *graph decontamination* and *connected graph search*): an extraneous mobile agent, the intruder, moves arbitrarily fast through the network infecting the visited sites; the task is to decontaminate the network using a team of system agents avoiding recontamination. The IC problem has been investigated for a variety of network classes, for example, trees, hypercubes, multi-dimensional grids, pyramids, chordal rings, and arbitrary graphs. Chapter 2 will present more detailed literature review on this. Let us point out that, in this problem, the harmful presence (the intruder) is *mobile* and harmful to the network sites, but does not cause any harm to the system agents.

The previous studies on BHS have not considered the transitioning or dynamic characteristics of the harmful nodes: BHs do not move and their number does not change; similarly, the previous studies on IC have not considered the case of a mobile intruder harmful for both sites and agents. The problem we consider in this thesis combines the characteristics from both BHS and IC: the harmful process is mobile (like intruders) and harmful also to the system agents (like black holes).

The harmful presence we consider is called *Black Virus, (BV)*, a dangerous process initially resident at an unknown network site. Like a BH, a BV destroys any arriving agents. When this occurs, unlike a black hole, the original node containing the BV becomes cleaned, i.e., not contaminated any more. On the other hand, the BV multiplies and spreads to all neighbouring sites, thus potentially increasing its number,

presence, and damage in the network. If however one of these neighbour sites contains a system agent, that clone of the BV is destroyed, that is, a BV is destroyed when it moves to a node that contains an anti-viral system agent; in this case, the agent is able to deactivate and permanently remove that instance of the BV.

The problem we study is Black Virus Decontamination (BVD) from networks, which consists of permanently removing any presence of the BV from the network using a team of system agents. Solving this problem is dangerous for the agents, since any agent arriving at a node where an instance of a BV resides will be destroyed; it is obviously dangerous for all the nodes where the BV spreads into. Since every time an agent arrives at a BV node, it causes the cloned BV to move and damage the neighbouring nodes. A protocol defining the actions of the agents solves the BVD problem if, within finite time, at least one agent survives and the network is free of BVs. A solution protocol will specify the strategy to be used by the agents; that is, it specifies the sequence of movements across the network that will enable the agents, upon all being injected in the system at a chosen network site, to decontaminate the whole network. The goal of a solution protocol is to minimize the *spread* of the BVs, i.e., the number of node infections by the BVs. Note that, since each instance of a BV has to be eventually removed and each removal requires the destruction of at least one agent, the spread also measures the number of agent *casualties*. The other important cost measure is the *size* of the team, i.e., the number of agents employed by the solution. An additional cost measure is the number of movements required by agents.

In addition to the potential applications in computer network as described in the above, the BVD problem may find its applications in military battle operations, medical science (i.e., virus locating, isolation, and removal), poisonous gas or material locating and removal in tunnel, and computer games.

A desirable property of a decontamination protocol is to prevent the nodes which have already been explored or cleaned by mobile agents from being re-contaminated by the BV spreading; note the re-contamination of a decontaminated site will occur if the virus is able to return to that site in absence of an agent. Following the literature on classical network decontamination (e.g., see [53]), a solution protocol with such a property will be called *monotone*.

1.2 Contributions

The main contributions are summarized below:

1. In this thesis, the *BVD* problem is introduced for the first time. This problem integrates in its definition the harmful aspects of the classical BHS problem (where however the dangerous elements are static) with the mobility aspects of the classical IC or network decontamination problem (where however there is no danger for the agents). Thus, it is the first attempt to model mobile/dynamic intruders, which are harmful not only for the sites but also for the agents. The main focus of the thesis is on the protocols for agents to complete this task by causing minimum network damage and by sacrificing the minimum number of agents.

2. The terminology and the model are described. We distinguish between *Sterile* and *Fertile* BV. Then some basic facts and properties of the problem are established. Basic tools and the general strategy to solve the problem are provided. The general strategy consists of a careful exploration of the network to find BVs and clean the BV clones.

3. The BVD problem is investigated for three important classes of interconnection network topologies: (multi-dimensional) *grids*, *tori*, and *hypercubes*. For each

class, a monotone solution protocol, which decontaminates the networks regardless of the number of dimensions, is provided and the complexity of the solution is analyzed. All the protocols are *optimal* both in terms of spread and size, and are asymptotically optimal in the total number of movements. A summary of the results is shown in Table 1.1, where q denotes the dimension of the network, and n and m denote the numbers of nodes and of links, respectively. The results appeared in [18, 19].

Table 1.1: Summary of Results in Special Graphs

Network	Spread	Size	Movements
q-Grid	$q+1$	$3q+1$	$O(m)$
q-Torus	$2q$	$4q$	$O(m)$
Hypercube	$\log n$	$2\log n$	$O(n \log n)$

4. The BVD problem is investigated in arbitrary graph in presence of a single BV. The main challenges of the BVD problem in arbitrary graph are discussed. Monotonic solution protocols are developed to achieve the objective of locating and cleaning the BV with optimal spreads in asynchronous settings. We notice that knowledge of the topology is not necessary for optimality; in fact, it is sufficient that each node has knowledge of its neighbours at distance 2. We devise two variants of the exploration strategy: Greedy and Threshold. For both protocols, the worst case complexity (spreads, size, and movements) is analyzed. A summary of the results is shown in Table 1.2, where $\triangle$ and n denote the maximum degree of the network and the numbers of nodes respectively.

   In addition, an interesting connection is established between the solutions of the BVD problem and the problem of determining rooted acyclic orientations of unoriented graphs with minimum outdegrees. An algorithm is developed to provide a distributed solution to this graph optimization problem. The complexity

Table 1.2: Summary of Results in Arbitrary Graphs

Protocol	GREEDY	THRESHOLD
Spread	Δ	Same as Greedy
Size	$\Delta + 1(sterile); 2\Delta(fertile)$	Same as Greedy
Movements	$O(\triangle n^2)$	$O(\triangle^2 n)$

in terms of agent movements, is analyzed. The results appeared in [20, 21].

5. The research extends the investigation from networks with single BV to networks with multiple BVs, i.e., MBVD problem. The challenges due to the existence of multiple BVs in graphs and the basic characteristics of the MBVD problem are discussed. In particular, *Separation* and its impacts on spread, and the distribution of original BVs in graphs and its influence on maximum casualties are investigated. In presence of multiple BVs, there is a drastic difference depending on weather the clones are sterile or fertile (i.e., generate other clones). After designing solutions to the MBVD problem in arbitrary graph, both for sterile and for fertile clones, the investigation continues to the MBVD problem in the special graphs previously studied, i.e., tori, hypercube, and q-dimensional Grids. The solution protocols in these graphs are presented, and the complexities of these solution protocols are obtained.

6. The BVD problem is finally investigated experimentally by using Distributed Algorithm Simulation Java (DisJ), a simulation platform. One of the main advantages is that the intended protocol (i.e., algorithm) and network topology can be separately developed, defined, and built. A large number of simulations on different sizes of graphs with many connectivity densities are carried out to implement the GREEDY algorithm and compare it to an algorithm based on random exploration. The 'GREEDY algorithm beats random exploration for all

graphs at every connectivity level. The simulation results disclose many interesting behaviors and confirm the worst case complexity analysis of the solution protocol. In addition to demonstrating the analytical results, the simulation also provides deep understanding on influence of graph connectivity density and size on complexities. The results appeared in [17].

1.3 Organization of the Thesis

The thesis is organized as follows:

Chapter 2 contains a literature review on related problems. We focus especially on the two closest problems, Black Hole Search and Intruder Capture. Then the simulation work related to agents is reviewed. Graph orientation, which is related to the solution of the BVD problem, is also reviewed.

Chapter 3 introduces terminology, definitions, and model for the BVD problem used in the rest of the thesis. Monotone property is the necessary condition for spread optimality, and the basic bounds on BV spread and the team size of agents are also formalized. The concepts of the residual degree of a node and a solution protocol are formally defined in this chapter.

Chapter 4 presents the general strategy for solving the BVD problem and a special tool for saving casualty are presented. The behavior of synchronous and asynchronous agents are discussed.

Chapter 5 focuses on the BVD problem for three classes of common network topologies: (multi-dimensional) *grid*, *tori*, and *hypercube*. For each kind of network, an optimal algorithm based on its structural properties is developed, and its complexity analysis is performed. Although described for a synchronous setting, the solutions can be easily adapted to asynchronous ones (requiring only coordination among the shadows and the exploring agents), all the results hold also in this case, and the extra

cost consists of an additional $O(n)$ movements in total for coordinating the activities.

Chapter 6 deals with the BVD problem for arbitrary graphs. The main challenge for solving the BVD problem in arbitrary graph is discussed. Exploring protocols are proposed based on residual degree to minimize the damage of the BVs during exploration. The solution protocols are proved to be spread optimal in asynchronous environments. Complexity analysis in terms of agent size, and movements are performed. An interesting connection is made and analyzed between the BVD problem and the problem of determining rooted acyclic orientations of unoriented graphs with minimum outdegrees.

Chapter 7 extends the investigation of the BVD problem to multiple-BV systems, i.e., arbitrary graph, and three classes of special networks: *grids*, *tori*, *hypercubes*. Due to the fact that the distribution of BVs in system is unknown and arbitrary, any pre-determined exploring paths (based on special structural properties) can be disturbed by encountering BVs. As a consequence, the agents need to continuously calculate and update the exploring sequence. The main focus of the chapter is the design of a solution for arbitrary graphs, then the general results are applied to the case of special topologies. Complexity analysis in terms of casualty and agent size are also performed and obtained.

Chapter 8 introduces DisJ, which is a Java based Distributed Algorithm Simulator, by discussing its capabilities, graphic interface, the method of modeling, execution and debugging, and its Application Programming Interfaces (APIs). Then we experimentally study the BVD problem by using DisJ. Algorithm and pseudo-code for BVD in arbitrary network in Chapter 6 is implemented in Java. Details of simulation work are also presented. Simulations are carried out on different sizes of graphs with many connectivity densities. The simulation results are collected and analyzed.

Chapter 9 summarizes the main conclusions from this research and proposes future

research activities.

Chapter 2

Literature Review

The mobile agent paradigm has been commonly employed and investigated in the artificial intelligence and software engineering communities; it also has been employed in distributed computing. A mobile agent has the features of autonomy, social ability, learning, and most importantly, mobility. In contrast to the conventional message-passing mechanism, mobile agents are active in that they can choose to migrate between computers at any time during their execution. This makes them a powerful tool for implementing distributed applications in a computer network.

A group of mobile agents can be used to perform various tasks, for example, network exploration, maintenance, and etc. Particularly mobile agents can be used to solve network security problems; however the introduction of mobile agent into networks also cause security issues and threats to networks. Flocchini and Santoro [52] summarized various security issues and solution algorithms by mobile agents. In particular, malicious agents can cause computer nodes malfunction or crash by contaminating or infecting them; on the other hand contaminated or infected hosts in turn can destroy working agents for various malicious purposes. The former problem is categorized as harmful agents, and the later is categorized as harmful hosts. In the rest of this chapter, we mainly review the studies in black hole search and intruder capture. Agent simulation platform and work are also reviewed because they are related to our simulation work. In addition, graph orientation review is also done due to its relation to our solution protocol.

2.1 Black Hole Search, BHS

In the case of a harmful host, the primary concerns are on locating harmful hosts, protecting agents against malicious hosts, and fixing hosts. Among the three aspects, the main focus has been on locating its position, for isolation and later deactivation. This is called *Black Hole Search* (BHS). The BHS problem assumes there is a BH or multiple static BHs residing at certain network nodes. BH can be virus, malicious software, or simply node crash failure. The word "static" means that the black hole does not move nor multiply. A black hole destroys any agent arriving at the node where it resides without any detectable trace. The task is for a team of system agents to locate the black hole(s). The solution algorithms are based on graph/network exploration since only exploration allows to locate the black holes. This task is dangerous for the agents; the detection method totally depends on the sacrifice of some agents and on the observation and deduction by surviving agents. The task is completed when at least one agent survives and reports the location(s) of the black hole(s).

The problem of BH search are closely related to a fundamental problem, graph exploration. Many investigation/research activities focus on the exploration and navigation problems for mobile robots in anonymous environment because exploring labelled graphs is always possible by travelling it, e.g., using DFS algorithm because every node has a unique identification. Initially the exploration was studied in networks where there are not dangers to exploring agents. This type of exploration is called safe graph exploration. Many algorithms were proposed for different settings. In [34], Das et al. considered a model for unknown environment with dispersed agents under the weakest possible setting. Various exploration models and works were summarized in the article. These investigations include different ways to mark the node

and to communicate among agents (pebble, marker, and whiteboard), one or multiple agents, asynchronous agents, different types of graphs (ring, tree, directed or undirected), different agent memory size limitation, and etc.

The BHS problem has been extensively investigated in different topologies and settings: by Chalopin [22, 23] in rings and tori, Czyzowicz et al. [31] in directed graphs, Dobrev et al. [40] in arbitrary graph, [41] in anonymous ring, and [38] in common interconnection networks, Cooper et al. [29] using multiple agents, Glaus [58] locating a black hole without the knowledge of incoming link, and Shi [101] with tokens in interconnected networks. In addition, Czyzowicz et al. [32] investigated the complexity of searching for a black hole, Klasing et al. [73] studied the hardness and approximation results for black hole search in arbitrary networks, and [74] studied the approximation bounds for black hole search problems.

The main distinctions made in the literature is based on if time-out can be used or not in model, i.e., whether the system is *synchronous* or *asynchronous*. The majority of the work focuses on the asynchronous model. In synchronous model, one agent is sent to check a node, other agents just wait until a bounded timer expires. Usually the bounded timer is treated as unitary time. Agent movements can be synchronized and it is assumed that it takes one unit of time for agents to traverse a link. Synchronous model allows detection of unknown number of multiple BH's. Asynchronous model presents more challenges than synchronous one because there is not bounded agent's travelling time on link (i.e., there is not concept of time-out). Therefore there is no way to distinguish between a slow link and black hole. The famous "cautious walk" [52] is the main exploring method to check if a node is safe or not. If one agent is sent to explore a node, before the node is confirmed safe, no more agents are sent to the node. Asynchronous model only allows detection of known number of BH's with an additional assumption that the total number of nodes in network has to be

known. The termination condition is to check if the sum of the number of BH's and the number of explored nodes is equal to the number of total nodes in system.

In asynchronous settings, a variation of the models, where communication among agents is achieved by placing tokens on the nodes, have been investigated in [39, 42, 101]. The case of black links in arbitrary networks has been studied in [24, 48], respectively for anonymous and non-anonymous nodes. The study of BHS in *time-varying graphs* has been started in [49].

In synchronous setting, Czyzowicz et al. [33] considered locating a black hole in a (partially) synchronous tree network for a given starting node. The minimum number of agents capable of identifying a black hole is two. They focused on the fastest possible black hole search by two agents. For arbitrary trees they gave a 5/3-approximation algorithm for this problem. In [32], Czyzowicz et al. considered the task of locating a black hole in an arbitrary synchronous network. For a given graph and given starting node they are interested in the fastest possible black hole search by two agents. They showed that the problem of finding the fastest possible black hole search scheme by two agents is NP-hard, and they gave a 9.3-approximation for it. Klasing et al. [73, 74] also studied the same problem of designing the fastest black hole search. They also proved the problem is NP-hard in arbitrary graph (even in planar graphs). They showed better approximation algorithm, thus improving on the 9.3-approximation algorithm from [32]. Their algorithm follows a natural approach of exploring networks via spanning trees. Cooper et al. [29] considered a setting of b black holes explored by k agents. If the network has n nodes, then any exploration algorithm needs $\Omega(n/k + D_b)$ steps in the worst case, where D_b is the worst case diameter of the network with at most b nodes deleted. In [30], Cooper and the same group of authors considered locating and repairing faults in network. They considered the case where one agent can repair only one faulty node. After repairing the fault,

the agent dies. They showed the number of steps necessary and sufficient to complete this task is $\Theta(n/k + D)$, where n is the number of nodes in the network, D is the diameter of the network, and k is the number of agents. Note in all the above studies, the main complexity is the smallest number of steps needed to complete the tasks. The impact of synchronicity in *directed graphs* has been studied in [76]. Finally, recent investigations have dealt with scattered agents searching for a black hole in *rings* and *tori* [22,23]. Finally Hohl, Ng, and Tander in [28,64,65,96] discussed various methods of protecting mobile agents against malicious hosts. It is worth to point out that a BH is *static*, that is, it does not propagate in the network and so it is not harmful to other sites. The number of BHs does not increase or decrease.

2.2 Intruder Capture

In the *intruder capture* problem, an intruder moves from node to node at arbitrary speed through the network, contaminating the sites it visits; a team of agents has to capture the intruder; the intruder is captured when it comes in contact with an agent. This problem, first introduced in [11], is equivalent to the problem of decontaminating a network infected by a virus while avoiding any recontamination. This problem, also called *connected graph searching*, is a non-trivial variant of the well known *graph searching* problem, which has been extensively studied (see [53] for a recent comprehensive survey), and is closely related to some basic graph parameters and concepts, including tree-width, cut-width, path-width, graph minor, and etc. The intruder capture problem has been investigated for a variety of network classes: *trees* [10,11,35], *hypercubes* [45], *multi-dimensional grids* [50], *pyramids* [100], *chordal rings* [46], *tori* [46], *outerplanar graphs* [54], *chordal graphs* [87],*Sierpinski graphs* [81], *star graphs* [68], *product graphs* [67], graphs with large clique number [102], while the study of arbitrary graphs has been started in [16,66]. The study of the problem

with stronger requirements for recontamination has been done for *toroidal meshes* and *trees* with strong majority threshold [83], and for *grids*, *tori* and *hypercubes* with arbitrary threshold [51, 82].

2.3 Simulation Work on Mobile Agents

In the section, we consider mobile agent network applications and their simulation work. The purpose is to see what simulation work has been done and what tools are used in these network applications. Next we review some existing simulators for reactive distributed algorithms in network applications. The reasons are: (1) to compare simulation and platforms in network systems; and (2) to have references for DisJ (simulation tool used in the thesis) which we introduce in later section of this work.

2.3.1 Network Applications

Mobile agents are used in some practical network applications, for example, distributed data mining [103], network management ([75, 95]), routing [2], consensus problems ([90, 91]), network mapping [85], multiagent coordination for connection, and etc.

In [2], Kaizar et al. attempted to use mobile agents to design and implement Agent based Distance Vector Routing (ADVR) to reduce the overhead and overcome the robustness issues associated with conventional routing protocol. ADVR borrowed some basic biologically inspired principles to assist coordination among the mobile agents that implement the routing task. Simulation work is done based on an event driven simulator. However, except mentioning the fact that Object-Oriented paradigm is adopted, no details on the simulator were provided. The article pointed out that agent-based solutions are suitable for many other network centric applications, for

example, network monitoring, management of large networks, resource management, and distributed cluster scheduling in support of scientific applications in grid computing.

Rubinstein and Duarte [95] investigated a mobile agent based network management solution to address scalability and efficiency issues. In contrast to the centralized network management systems, the authors proposed decentralized processing and controlling by using two-level managers, domain managers and manager of domain managers, which send mobile agents to network elements to perform required tasks. In their simulation work, Network Simulator (NS) is used. The considered performance parameters are number of bytes transmitted and received by domain manager for the mobile agents.

In [90, 91], Olfati-Saber et al. studied consensus problems for networks of dynamic agents with fixed and switching topologies. Simulations results are provided to demonstrate the effectiveness of their theoretical results. Again the details of the simulation were not presented.

Minar et al. [85] investigated the cooperation of mobile agents for mapping networks. Again to overcome the shortcomings of generating routing maps in a centralized manner, the mobile-agents approach is chosen to obtain routing maps in a distributed and decentralized strategy. The agents move around the network and discover topology information. When meeting on a node, they exchange information with each other, so an agent can acquire knowledge about parts of the network that it has never visited. Their simulation system consists of Java code implementing a discrete event scheduler, a graphical view, a data-collection system, and the simulated objects themselves: network nodes and mobile agents.

In [69], Ji and Egerstedt address the connectedness issue in multiagent coordination, i.e., the problem of ensuring that a group of mobile agents stays connected while

achieving some performance objective. In particular, they study the rendezvous and the formation control problems over dynamic interaction graphs.

It seems to be true that there is not much simulation work done in relation to mobile-agent theoretical and practical network applications. The simulations are usually performed on non-general purpose and non well known simulators; the details on the simulation work are usually not provided. This situation of parcity must be contrasted wih the extremely rich literature on Agent-Based Modeling (ABM), that uses powerful intelligent agents to simulate complex problems, and the abundance of and related simulation platforms (e.g., see [63, 71, 84, 86, 92] for surveys) which are unfortunately not relevant to this thesis.

2.3.2 Simulation Platforms for Distributed Computing

Finally, it is necessary to survey some existing simulators for reactive distributed algorithms in network applications. The list includes DAJ (Distributed Algorithms in Java) by Mordechai Ben-Ari [14], T-DAJ (Toolkit for Distributed Algorithms in Java) by Wolfgang Schreiner [99], DAP (Distributed Algorithms Platform) by Chatzigiannakis et al. [27], SinAlgo (Simulation of Network Algorithm) by Distributed Computing Group, Switzerland [37], and DisASTer (Distributed Algorithm Simulation Terrain) by University of Applied Science, Germany [89].

DAJ (Distributed Algorithms in Java) [14] is a framework, which helps user to implement distributed algorithms in Java. The platform enables users to interact with the execution of an algorithm to display the state of each node and construct and control scenarios. A log file of commands can be created, so users can automatically replay scenarios for complete understanding them. Ten algorithms have been implemented for commonly taught algorithms.

T-DAJ [99] is a toolkit for designing, implementing, testing, simulating, and visualizing distributed algorithms in Java. DAJ provides Java library with simple programming interface that allows to develop distributed algorithms. DAJ supports message passing model. The generated programs may be executed as a standalone Java application or embedded as applets into HTML pages and executed by Web browsers. DAJ is freely available over the World Wide Web.

DAP (Distributed Algorithms Platform) [27] is aiming at providing a generic simulation environment for implementing, simulating, and testing distributed algorithms for wired and wireless networks. DAP allows users to design and implement a distributed protocol by creating a customized environment. Users can use provided graphical user interface to monitor, control, and visualize the execution of simulations, and gather statistics.

SinAlgo [37] is a simulation framework for testing and validating network algorithms. It is mainly suited for packet-oriented wireless network (but not limited to it). SinAlgo does not simulated different layers of the network stack, but focuses on the verification of network algorithms abstracted from the underlying layers. It supports a message passing view of the network. A node may send a message to a specific neighbor or all its neighbors, react to received messages, set timers to schedule actions in the future, and so on. SinAlgo offers a broad set of network conditions, under which users may test their algorithms.

DisASTer [89] is a platform for the implementation of distributed algorithms. It provides a Java class library that eases the programming of distributed algorithms in Java. Moreover, DisASTer constitutes an execution environment that enables the user to specify interactively a topology at runtime. It allows the user to control the execution of the algorithm (start, stop, suspend, resume, go back and forth). In addition, the execution of a distributed algorithm can be observed through some

built-in visualization panels (topology view, sequence view, message queue view). DisASTer supports the implementation of further application-specific views.

The surveyed simulators (platforms or toolkits) provide/support various good features and advantages (however these simulators usually don't provide all of following):

- Object-Oriented Design and Implemented by popular languages C++ and mostly in Java;
- Some level of friendly GUI interfaces;
- Synchronous or asynchronous;
- Some level debug capabilities.

They have some common limitations or disadvantages:

- Supporting only message passing model;
- Supporting only bi-directional link;
- Creating network topology needs some level of configuration or coding;
- Algorithm implementation and network environment are tightly coupled;
- Statistics are not always provided, so pre-coding or post processing are needed to obtain these statistics;
- Usually do not support adversary events;
- Limited information display in the middle of simulation;
- No always support full capable debugging, and record and replay capabilities;

2.4 Graph Orientation

In our quest for an optimal solution to the BV decontamination problem in arbitrary networks, we establish (in Chapter 6.5) an interesting connection between solutions of the BVD problem and the problem of determining rooted acyclic orientations of unoriented graphs with minimum outdegrees. As a consequence, our protocols provide a distributed optimal solution to this graph optimization problem.

The earliest study we found on graph orientation is by Robbins [94]. He showed that a graph G has a strong orientation if and only if G is connected and bridgeless. Gallai [57] settled a conjecture of Erdös by showing that a graph G has an orientation containing no path of length κ if and only if G is κ-colourable. Harary et al. [62] proved that a graph has an orientation that is not self-converse if and only if G is not one of $K_1,\ K_2,\ C_3,\ C_4\ or\ C_5$. In [44] Erdös characterized those graphs having an orientation containing no cycle and further containing no semicycle which becomes a cycle if one of its arcs is reversed. Entringer [43] wrote a good survey on early graph orientation and studied on the characterizations of graphs having orientations satisfying local degree restrictions.

Hakimi [60] investigated the realizability of a finite set of positive integers as degrees of the vertices of a linear graph. He then [61] studied the problem on the realizability of a finite set of pairs of non-negative integers $(d_i^+, d_i^-) : i = 1, 2, ..., n$ as degrees of the vertices of a directed graph in [61]. The directed graphs considered in his study are allowed to have parallel elements but it is assumed to contain no self-loop elements. The integers ($d_i^+\ and\ d_i^-$) specify the number of arrowheads directed toward and away from vertex v_i respectively. Other related problems such as: realizability of a given set of non-negative integer pairs as a connected directed graph, strongly connected directed graph, and cycle-less directed graph were also discussed. Kim et al. [72] recently studies the realizing all simple graphs with a given degree sequence.

They gave a necessary and sufficient condition for a sequence of non-negative integers to be realized as a simple graphs degree sequence such that a given (but otherwise arbitrary) set of possible connections from a node are avoided. They then use this result to present a procedure that builds all simple graphs realizing a given degree sequence.

Frank and Gyárfás [55, 56] studied how to orient the edges of a graph. Let $u(x)$ *and* $l(x)$ denote integer functions defined on the vertices of an undirected graph, and $\delta(x)$ be the outdegree of a node x. Frank gave the necessary and sufficient conditions for a orientation satisfying $l(x) \leq \delta(x) \leq u(x)$ for any $x \in V$, and also gave the necessary and sufficient condition for a bridgeless graph have a strongly connected orientation satisfying $\delta(x) \leq u(x)$. Finally he studied the orientations with a rooted tree with outdegree bounds.

Kowalik [77] recently studied an approximation scheme for lowest outdegree orientation and graph density measures. He dealt with the problem of finding such an orientation of a given graph that the largest number of edges leaving a vertex (called the outdegree of the orientation) is small. For any $\varepsilon \in (0, 1)$ he showed an $\tilde{O}(|E(G)|/\varepsilon)$ time algorithm which finds an orientation of an input graph G with outdegree at most $\lceil (1+\varepsilon)d^* \rceil$, where d^* is the maximum density of a sub-graph of G. It is known from [56] that the optimal value of orientation outdegree is $\lceil d^* \rceil$. Kowalik pointed out the algorithm has applications in constructing labelling schemes, in approximating such graph density measures as arboricity, pseudoarboricity and maximum density. His results improve over the previous, 2-approximation algorithms by Aichholzer et al. for orientation pseudoarboricity, by Arikati et al. for arboricity, and by Charikar for maximum density. His idea of the approximation algorithms is very simple. He reduced of finding a d-orientation of a given graph (if it exists) to finding a maximum flow in some network. The algorithm starts Dinics maximum flow

algorithm in some network which depends on the input graph and some parameter d, and stops when augmenting paths grow too long.

Asahiro et al. [3,5,6] studied the problem of orienting the edges of a weighted graph such that the maximum weighted out-degree of vertices is minimized. This problem, which has applications in the guard arrangement for example, can be shown to be NP-hard generally. Particularly in [6] they first gave optimal orientation algorithms which run in polynomial time for the following special cases: (i) the input is an unweighted graph, or more generally, a graph with identically weighted edges, and (ii) the input graph is a tree. Then, by using those algorithms as sub-procedures, they provide a simple, combinatorial, minw_{max}/w_{min}, $(2-\varepsilon)$ approximation algorithm for the general case, where w_{max} and w_{min} are the maximum and the minimum weights of edges, respectively, and is some small positive real number that depends on the input.

As pointed by Kowalik in [77], it follows from a theorem by Frank and Gyárfás [56] that $\lceil d^*(G) \rceil$ equals the outdegree of the lowest outdegree orientation of the graph G. Finding the lowest outdegree orientation of the graph G is equivalent to finding the maximum density of graph. Goldberg [59] reduced the problem of finding maximum density subgraph to a series of minimum capacity cut computations $O(\log n)$, which in turn can be done using network flow techniques. In [98], Schaeffer made a very good survey on algorithms for dense subgraph discovery. He pointed out that this problem is closely related to clustering though the two problems have some differences.

In [70], Khuller et al. showed without any size constraints, a sub-graph of maximum density can be found in polynomial time. When requiring the subgraph to have a specified size, the problem of finding a maximum density subgraph becomes NP-hard. In the paper they focused on developing fast polynomial time algorithms for several variations of dense subgraph problems for both directed and undirected

graphs. Charikar [26] studied the optimization problems of finding subgraphs maximizing these notions of density for undirected and directed graphs. His paper give simple greedy approximation algorithms for these optimization problems. Intuitively, in order to produce a dense subgraph, it should throw away low degree vertices. This suggests a fairly natural greedy algorithm. In [80], Liu et al. observed that many algorithms to mine quasi-cliques from undirected graphs have not fully utilized the minimum degree constraint for pruning. They proposed an efficient algorithm to find maximal quasi-cliques from undirected graphs by using effective pruning techniques based on the degree of the vertices to prune unqualified vertices as early as possible.

2.5 Summary

The BV decontamination problem is closely related to black hole search and intruder capture. Previous studies on BHS have not considered the transition characteristics of the harmful nodes/agents. BH are static, i.e., BH's do not move and their numbers does not changes. The BVD problem is a combination of black hole search and graph decontamination by mobile agents. It introduces the dynamic characteristic for black hole, and we called it BV. A (or more) BV resides in any vertex in a graph. At first, mobile agents need to explore the graph to locate it with some restrictions. Once detected, more mobile agents are deployed to surround the BVs. BVD model allows BVs to spread upon weaken up by agents' exploration. One major restriction is to minimize the spreads of the BV upon its detection. As an example, if one BV causes a serious failure on one high-capacity router in a backbone network, we always need to minimize the number of such failures. Another major restriction is to protect all the nodes, which have been explored and checked by mobile agents, from being infected or contaminated by BV spreading. This problem may have significance in some practical applications, for example, distributed, military strategy/operation,

poisonous substance exploration, and biochemistry.

In this chapter, we also reviewed agent related simulation and their platforms. First we surveyed some mobile-agent based network applications and their simulation work. It seems to be true there are not much simulation work done in mobile-agent related network applications. The existing simulation are usually performed on non-general purpose and non well known simulators. The survey of some existing simulators for reactive distributed algorithms in network applications show they provide/support various good features and advantages. The common limitations or disadvantages are reviewed and summarized. The purposes are: (1) to compare simulation and platforms in network systems; and (2) to have references for DisJ (Distributed Java, a simulation tool, which we introduce in later chapter of the thesis.

The solution protocols for BV decontamination essentially need agents to explore the network nodes in certain way to minimize the node damage. The exploration from homebase results in a special orientation of all links. Therefore graph orientation work has been surveyed.

In the next chapter, the basic terminology and the model for the Black Virus Decontamination problem are introduced. Fundamental properties are established, as well as bound on the costs of solving the BVD problem.

Chapter 3

Model, Properties and Bounds

In this chapter, the basic terminology and the model for the Black Virus Decontamination problem are introduced.

3.1 Model

3.1.1 Network, Agents, Black Virus

The environment is a network supporting mobile agents; its topology is modelled as a simple undirected connected graph $G = (V, E)$ with $n = |V|$ nodes (or sites) and $m = |E|$ edges (or links). We denote by $E(v) \subseteq E$ the set of edges incident on $v \in$ V, by $N(v) \subset V$ the set of its neighbours, by $d(v) = |E(v)|$ its degree, and by $\triangle(G)$ (or simply $\triangle$) the maximum degree in G. We denote by $Diam(G)$ (or simply $Diam$) the diameter of G.

Every node v has a distinct identity $id(v)$. The links incident to a node are labelled with distinct port numbers (*local orientation*). The labeling mechanism could be totally arbitrary among different nodes. Without loss of generality, we assume that the link labeling for node v form the set $l_v = 1, 2, 3, ..., d(v)$.

A team $\mathcal{A} = \{A_1, \ldots, A_k\}$ of mobile system agents, called *cleaners*, provided with decontamination capabilities, is injected in the network at a node h called the *home base*. Each agent $A \in \mathcal{A}$ is a computational entity with its own local memory and a unique identifier $id(A)$ from some totally ordered set. It can move from node to neighbouring node; when at a node v, the agent can see the node's identity $id(v)$,

as well as the labels of the incident edges. The agents can have different roles (i.e., states), but they all operate according to the same protocol.

More than one agent can be at the same node at the same time. Communication among agents (e.g., to coordinate, synchronize or update their maps, and etc.) occurs when they are at the same node (*face-to-face communication*); there are no a priori restrictions on the amount of exchanged information.

We say that the agents have *full topological knowledge* if each agent has a map of the entire network indicating the identities of the nodes and the labels of the edges. We say that the agents have *k-hop visibility* if, when at a node v an agent sees the k-neighbourhood $N^k(v)$ of v, including the node identities and the edge labels. Note that *Diam*-hop visibility is equivalent to full topological knowledge.

In the network, there is a node (or more than one nodes) infected by *black virus* (BV), an extraneous process endowed with reactive capabilities for destruction and spreading. It is harmful not only to the node where it resides but also to any agent arriving at that node. More precisely, like a black hole, a BV destroys any agents arriving at the network site where it resides.

An active BV can be deactivated only by an arriving cleaner; even then, it destroys the cleaner and it can harm the neighbouring networks sites. More precisely, when being deactivated, the BV releases *clones* (BVC) that spread to all the neighbouring nodes.

The clones of a BV have the same harmful capabilities of the original BV. We distinguish between *fertile* and *sterile* clones. A fertile clone is a complete black virus; thus the number of BVs in the network may increase over time. Sterile clones are unable to produce clones. At each node at any time there is at most one BV(c): multiple BVC arriving at the same node merge; in the merge between a BV and a sterile clone, the result will be a BV.

If a BVC arrives at a node with no cleaner, it infects the node and becomes resident there; if the node is occupied by a cleaner, the BV clone is destroyed before it can cause any harm. Thus, the only way to eliminate a BV from the system with the loss of a single cleaner is to first have cleaners surround it completely and then have a cleaner move to the BV node.

The process of one or more cleaners moving to a node occupied by a black virus and the resulting effects is referred to as *BV triggering*.

Summarizing, there are five possible situations when cleaners or BVCs arrive at a node v:

1. Cleaners arrive at v which is empty or contains other cleaners; v remains clean and the agents there can communicate with each other.

2. Cleaners arrive at v where there is a BV: the cleaners are destroyed, BVCs move to each of v's neighbours, but v becomes clean.

3. Cleaners arrive at v where there is a sterile BVC: the cleaners are destroyed, but v becomes clean.

4. BVCs arrive at v which is empty or where there is a BV(C): the node becomes/stays contaminated; the BVC merge into one BV(C).

5. BVCs arrive at v where there are cleaners: the node stays clean, the BVCs are destroyed, and cleaners are unharmed.

The simultaneous arrival at a node of cleaner(s) and clone(s) is treated as their arrival in an arbitrary sequential order; hence the outcome is non-deterministic.

With respect to time and synchronization, networks can be synchronous or asynchronous. In fully *synchronous* networks, there is a global clock indicating discrete time units; It takes one unit of time for an agent or a clone to move from one node

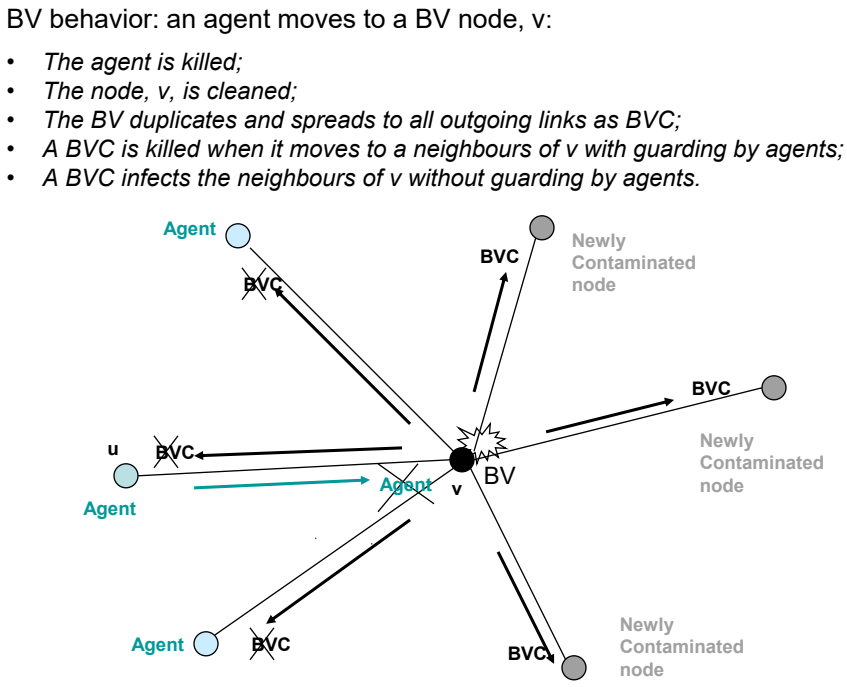

Figure 3.1: BVD Model

to another, and computing and communication time is assumed to be negligible compared to moving time. In *asynchronous* networks, there are no global clocks, and the duration of any activity (e.g., processing, communication, moving) by the agents, the BV, and its clones is finite but unpredictable; any needed synchronization must be achieved by the agents' protocol. We denote by `Asynch` and `Fsynch` the class of Asynchrounous and fully-synchronous networks, respectively.

3.1.2 Problem, Solutions, Costs

The Black Virus Decontamination (BVD) problem is to permanently remove the BV, whose location is not known a priori, and its clones from the network using the team of cleaners starting from a given arbitrary node, called home base (HB).

A protocol defining the actions of the cleaners solves BVD in $G \in$`Fsynch` if, within finite time, at least one cleaner survives and the network is free of BVs, regardless of

the location of the home base and of the black virus. A protocol defining the actions of the cleaners solves BVD in $G \in$`Asynch` if, within finite time, at least one cleaner survives and the network is free of BVs, regardless of the location of the home base and of the black virus, and regardless of the duration of the actions of the agents, of the BV, and of its clones.

A protocol solves BVD in `Fsynch` (resp. `Asynch`) if it solves it in every network $G \in$`Fsynch` (resp. $G \in$`Asynch`). Let $\mathcal{P}_{Fsync}$ (resp., $\mathcal{P}_{Async}$) denote the set of all solution protocols of the BVD problem in `Fsynch` (resp., `Asynch`); Let $\mathcal{P} \in \{\mathcal{P}_{Fsync}, \mathcal{P}_{Async}\}$.

The goal of a solution protocol $P \in \mathcal{P}$ is to decontaminate the network minimizing the *spread* of the black virus, i.e., the number of node infected by the BV and its clones. Note that, since each instance of the BV (original or clone) has to be eventually removed and since each removal requires the destruction of at least one cleaner, the spread also measures the number of cleaner *casualties*. We defined two types of optimality, overall and everywhere.

Given a solution protocol $P \in \mathcal{P}$ and a network $G = (V, E)$, let $spread(P, G)$ denote the maximum number of casualties incurred when executing P in G in the worst case (i.e., over all possible initial locations of the BV and of the home-base, and all possible execution delays); then

$$spread(G) = Min_{P \in \mathcal{P}}\{spread(P, G)\}$$

denotes the minimum amount of casualties possible to decontaminate G in the worst case. A solution protocol P is *worst-case optimal* if $spread(P, G) = spread(G)$ for all G.

A finer cost measure is the maximum number of casualties $spread(P, G, h)$ incurred over all possible executions and locations of BV starting from home-base $h \in V$; thus

$$spread(G,h) = Min_{P\in\mathcal{P}}\{spread(P,G,h)\}$$

denotes the minimum amount of casualties possible to decontaminate G when starting from h. A solution protocol P is *overall optimal* if for all $G = (V,E) \in \mathcal{G}$, and all $h \in V$, $spread(P,G,h) = spread(G,h)$; clearly overall optimality implies worst-case optimality.

An even more stringent form of optimality would be the requirement that a solution protocol makes every node have a minimum chance of contamination. Here we just claim that everywhere optimality is impossible. We formally prove it after we introduce "executions and explorations" in next section.

For a (worst-case or overall) optimal protocol P, an important cost measure is the size of the team of cleaners $size(P,G)$; we denote by $size(G) = Min\{size(P,G)\}$ the smallest team-size over all optimal solution protocols P. Additional cost measures are the number of movements performed by the agents, and execution time (if in a synchronous network).

3.1.3 Executions and Explorations

To decontaminate the network, the team of cleaning agents must first find the location of the BV; this task requires the agents to explore the network until the BV is found (it could be the very last node visited). In particular, the execution of any solution protocol in a network where no BV is present, called a *BV-free execution*, generates an *exploration* of all the network nodes.

Consider a BV-free execution of a solution protocol P in network G with homebase $h \in V$. Let $V_{ex}(t)$ and $V_{ux}(t)$ be the explored and still unexplored nodes of the graph at time t respectively. Initially $V_{ex}(0) = \{h\}$ and $V_{ux}(0) = V \setminus \{h\}$. Let $E_{ex}(v,t)$ and $E_{ux}(v,t)$ indicate the explored and unexplored edges incident to v at time t

respectively. Thus $G_{ex}(t) = (V_{ex}(t), E_{ex}(t))$ is the subgraph induced by the set of explored nodes at time t. Analogously, $G_{ux}(t) = G \setminus G_{ex}(t)$ be is unexplored graph at time t. Let $N_{ex}(v)$ and $N_{ux}(v)$ be the explored and unexplored neighbours of node $v \in V$, respectively. The set $B(t) = \{v \in V_{ex}(t) : N_{ux}(v) \neq \emptyset\}$ of explored nodes having one or more unexplored neighbours is called the *border* of G at time t, while the set $F(t) = \{v \in V_{ux}(t) : N_{ex}(v) \neq \emptyset$ of still unexplored nodes neighbouring on a border node is called the *frontier* of G at time t;

Let $t_1 < t_2 <, ... < t_s$ be the times when, in that BV-free execution, cleaners move to still unexplored nodes, and let $V_i = V_{ex}(t_i) \setminus V_{ex}(t_{i-1})$ be the new nodes explored at time t_i; clearly $\bigcup_i V_i = V$, $V_i \bigcap V_j = \emptyset$ for $i \neq j$, and $V_i \subseteq F(t_{i-1})$. The execution is *sequential* if $\forall i \ |V_i| = 1$, that is, each V_i contains a single node (and thus $s = n - 1$).

We say that protocol P is *sequential* if all its BV-free executions are sequential. In other words, a solution protocol P is sequential if it prescribes the nodes to be explored one a time, and the exploration of a new node to be started only after the exploration of the previous node has been completed. Note that, by definition, in every BV-free execution of a sequential solution protocol P in network G with home base $h \in V$, the order in which the nodes are first explored is the same regardless of the system delays. A protocol P is *not-sequential* if it allows simultaneous (i.e., parallel) exploration of multiple nodes in some of its executions.

A solution protocol is *monotone* if no recontamination occurs in any of its executions; that is, once a node is visited by a cleaner, it will not be (re)contaminated by a BV clone.

Now it is the time to prove our previous claim on everywhere optimality. Let us introduce a concept, *residual degree*. At any stage of the exploration, agents always choose the next target with minimum chance of contaminations from the unexplored portion of the network, G_{ux}. The *residual degree* of a node v at time t is defined as the

number of unexplored incident edges (or the number of the unexplored neighbours, which is equal to the degree of the node minus the number of explored neighbours of the node) and denote it as $\rho(v,t)$ $(= E_{ux}(v,t))$. In the following discussion, when no ambiguity arises, the temporal parameter t is omitted, $\rho(v)$. Clearly residual degree is equal to the number of new BV contaminations if a BV resides on this node. Calculating $\rho(v)$ of an unexplored node is possible if agents know the full topology or distance-2 topology of every newly explored node in the graph, and the explored portion of the graph, G_{ex}. Obviously the residual degree of a node is $0 \leq \rho(v) \leq d(v)$. The upper limit is $d(v)-1$ if any one (more) of its neighbours has (have) been visited. Otherwise, the residual degree is the same as the degree of the node. Formally given a starting homebase, x_0, let $\mathcal{P}$ is a set of all solution protocols and P is one solution protocol, and let $\rho^P(x_i)$ be the residual degree of node x_i in protocol P. This *everywhere optimality* tries to achieve $\rho^P(x_i) \leq \rho^{P'}(x_i)$ for all i, where $0 \leq i \leq (n-1)$, i.e., in solution protocol P, the residual degree of every node is less than or equal to that of the same node in any other solution protocol $P' \in \mathcal{P}$.

Theorem 1. *Everywhere optimality is impossible.*

Proof. Assume such an optimal protocol P exists. Then there always exists a node, x_i, such that $d_r^P(x_i) \neq 0$ in the execution of the P on graph $G(V,E)$. However, there always exists another protocol $P' \in \mathcal{P}$, which allows to explore the node, x_i, in the end, i.e., $d_r^{P'}(x_i) = 0$. Therefore no solution protocol can be everywhere optimal. □

3.2 Properties and Bounds

In this section, fundamental properties are established, as well as bound on the costs of solving the BVD problem.

3.2.1 Monotonicity and Structure

The concern is with solution protocols that are efficient and possibly optimal with respect to the *spread* (i.e., number of casualties) and to the *size* (i.e., the total number of cleaning agents).

In this respect, observe that, once the BV is found at a node v, BV clones will move to every neighbour of v. This means that, any previously explored neighbour of v will be contaminated unless an agent is there at that time. An immediate consequence is with regards to the *monotonicity* of the solution.

Property 3.2.1. *Monotonicity is necessary for spread optimality*

Proof. By contradiction, let P be a protocol that solves BVD in G non-monotonically with optimal spread. That is, during the execution of P in G, a BV moves to an unguarded node v which was previously explored by an agent. As a result, v becomes a BV node. Note that the movement of the BV to v must have been triggered by an agent A moving to a BV node u neighbour of v. Consider now a protocol P', whose execution is totally identical to that of P except that, before A moves to u, an agent B moves to v; B is either an agent that, at the time of A moving to u, is not being used by P for protection of an explored neighbour of v, or an additional one. This means that when the movement of A to u causes the BV to move to v, agent B destroys that BV without harm for itself. In other words $spread_{P'}(G) < spread_P(G)$ contradicting the spread optimality of P. □

As a consequence, in our search for spread-optimal solutions, we must restrict ourselves to solution protocols that are monotone.

The values of the *size* and of the *spread* depend on many factors, first and foremost the structure of the graph, but also on the location of the homebase and of the BV. Some basic bounds follow immediately from the definition of the problem. In

particular, regardless of the topology of the network, since at least one agent must survive by definition, we have

Property 3.2.2. *For any network G, $size(G) \geq spread(G) + 1$.*

Let v be an articulation point of G, let $G_{min}(v)$ denote the *smallest* (in terms of number of nodes) connected component formed by removing v and its incident edges from G, and let $\alpha_{min}(v)$ denote its size.

Property 3.2.3. *Let the clones be fertile. If G has an articulation point v, then*

$$spread(G) \geq n - \alpha_{min}(v)$$

Proof. Consider the case when the homebase is a node u in $G_{min}(v)$, and the BV is located at the articulation point v. In this case, regardless of the solution strategy, the BV will spread to all nodes of all connected components formed by removing v from G, with the exception of $G_{min}(v)$. □

Let $e = (u, v)$ be a cut edge of G, let $G_{min}(e)$ denote the *smallest* (in terms of number of nodes) of the two connected components formed by removing e from G, and let $\alpha_{min}(e)$ denote its size; without loss of generality let u be in $G_{min}(e)$.

Property 3.2.4. *Let the clones be fertile. If G has a cut edge e, then*

$$spread(G) \geq n - \alpha_{min}(e)$$

Proof. Let $e = (u, v)$ be a cut edge; without loss of generality let u be in $G_{min}(e)$. Consider the case when the homebase is a node in $G_{min}(e)$, and the BV is located at v. In this case, regardless of the solution strategy, the BV will spread to all nodes of $G \setminus G_{min}(e)$. □

As a corollary of Property 3.2.4, it follows that there are graphs where the *size* and the *spread* are $\Omega(n)$:

Property 3.2.5. *Let the clones be fertile. If G contain a node v of degree $|E(v)| = 1$, then*

$$spread(G) \leq n - 1$$

Notice the the class of graphs covered by Property 3.2.5 includes acyclic networks, i.e., *trees*. In the case of trees, the bounds are tight:

Lemma 1. *Let the clones be fertile. If T is a tree on n nodes, then*

$$size(T) = spread(T) + 1 = n$$

Proof. Necessity is by Properties 3.2.2 and 3.2.5. To see that n agents always suffice in a tree T, consider a depth-first traversal where, starting from the homebase, a single agent moves forward; once the outcome is determined (either the agent returns or a BV arrives), all remaining agents move to that node; when backtracking, all remaining agents move back. It is easy to see that this strategy decontaminates the tree with at most $n - 1$ casualties. □

In other words, there are graphs (e.g., trees) where the *size* and the *spread* are $\Theta(n)$. On the other hand, there are networks where the *spread* and the *size* are $\Theta(1)$ regardless of the number of nodes. To see this, let us concentrate on *regular* graphs.

Lemma 2. *Let G be a Δ-regular graph. Then $spread(G) \geq \Delta$.*

Proof. Since the very first node v explored by the agents can be a BV, the number of casualties is at least one for each neighbour of v (other than the homebase) plus the one incurred at v. □

In most regular graphs, the relationship between *size* and *spread* can be much higher than that expressed by Property 3.2.2; in fact

Lemma 3. *Let G be a triangle-free Δ-regular graph. Then $size(G) \geq 2\Delta$.*

Proof. Let the first node v explored by the agents be a BV; thus all $\Delta - 1$ neighbours (other that the homebase) of v become BV nodes. Since G is triangle-free, none of these BV nodes are neighbours. This means that each of these BV nodes has now Δ clean neighbours (including v). Let z be the first of these BV nodes to be cleared; at least Δ agents are needed to protect the clean neighbours of z, in addition to the Δ casualties, for a total of at least 2Δ agents. □

Notice that the class of graphs covered by Lemma 3 includes among others *rings*, *tori*, and *hypercubes*. In the case of *rings*, both *size* and *spread* are constant, independent of n, and the bounds are tight:

Lemma 4. *Let R be a ring of n nodes. Then, $size(R) = 4$ and $spread(R) = 2$*

Proof. By Lemma 2 it follows that $spread(R) \geq 2$ and thus, by Lemma 3, $size(R) \geq 4$. Sufficiency of four agents with two casualties regardless of the ring size n is immediate. □

3.2.2 Sequentiality and Residual Degrees

Consider how the exploration of the network is performed when executing a solution protocol. Recall that a solution protocol P is *sequential* if it prescribes the nodes to be explored one a time, and the exploration of a new node to be started only after the exploration of the previous node has been completed.

A very important fact is that, in asynchronous systems, sequentiality is not a handicap for optimality.

Lemma 5. *There exist overall optimal sequential protocols in $\mathcal{P}_{Async}$.*

Proof. Let $\mathcal{O}_{Async} \subseteq \mathcal{P}_{Async}$ be the set of the overall optimal protocols for asynchronous systems. Consider an arbitrary $P \in \mathcal{O}$. If P is sequential, the lemma holds. Let P be not sequential; that is, in some BV-free executions in some graphs from some home bases, P requires agents to move to more than one unexplored node concurrently. Consider now the protocol $\tau(P)$ obtained from P by replacing each concurrent visit of unexplored nodes with a sequential visit of the same nodes in an arbitrary order. Every execution of $\tau(P)$ is sequential, thus, $\tau(P)$ is sequential. Furthermore, every execution of $\tau(P)$ is equivalent to a particular execution of P where, whenever P required more than one agent moving concurrently to more than one unexplored nodes, the delays force the nodes to be reached by those agents precisely in the sequential order required by $\tau(P)$; thus, since $P \in \mathcal{O}_{Async}$, it follows that $\tau(P) \in \mathcal{O}_{Async}$, proving the Lemma. □

Thus, in our search for spread-optimal solutions, in asynchronous systems we can restrict ourselves to protocols that are sequential (i.e., in which the nodes are visited one at a time) and monotone (i.e., all the visited neighbours of the node being explored are be protected with a cleaner). Note that, by definition, in every BV-free execution of a sequential solution protocol P in network G with home base $h \in V$, the order in which the nodes are first explored is the same regardless of the system delays.

Let $\mathcal{MS}$ be the set of all monotone sequential solution protocols. Consider a protocol $P \in \mathcal{MS}$, and a graph $G = (V, E) \in \mathcal{G}$. Since P is sequential, the nodes of G are visited for the first time one at a time, starting from the home base. Let

$$P[G, h] \equiv [x_0, x_1, x_2, ..., x_{n-1}]$$

be the resulting ordered sequence in a BV-free execution of P in G starting from

$h = x_0$.

The *residual degree* of x_i in $P[G, h]$, denoted by $\rho(x_i)$, is the number of neighbours of x_i following it in the sequence $P[G, h]$; i.e., $\rho(x_i) = |\{x_j \in N(x_i) : i < j < n\}|$, where $N(x_i)$ represents all the neighbours of x_i.

Note that the residual degree of the entire sequence $P[G, h]$ is just the largest of the residual degrees of its elements:

$$\rho(P[G, h]) = \max_{0<i<n} \{\rho(x_i)\}$$

Notice that the sum of the residual degrees of all nodes in the sequence is equal to the number of edges: $\Sigma_{0 \leq i < n} \rho(x_i) = |E|$.

A lower bound on the number $spread(P, G, h)$ of casualties created, in the worst case, by the cleaners executing P in $G = (V, E)$ starting from $h \in V$ can be easily derived based on the notion of residual degree.

Lemma 6. $spread(P, G, h) \geq \rho(P[G, h]) + 1$.

Proof. In a monotone protocol, when visiting a new node, the still unexplored nodes are unprotected. Thus, if the BV is at x_i, all the neighbours of x_i still unexplored, i.e., the set $\{x_j \in V(x_i) : i < j < n\}$, will become contaminated. Since one casualty is required to decontaminate each of them, the total number of casualties, including the one occurred at x_i, is precisely $\rho(x_i) + 1$. □ □

Let us now introduce the important notion of feasible permutations. By Lemma 6, it follows that to minimize spread, we need to find a protocol that has minimum residual degree in all graphs and for all choices of the home base. That is, our quest is for a protocol $P \in \mathcal{P}$ such that $\forall G = (V, E), \forall v \in V, \rho(P[G, h]) = spread(G, h)$. To aid in our quest, we recall that the sequence $P[G, h]$ is a permutation of the n network nodes.

Let us call a permutation $[x_0, x_1, x_2, ..., x_{n-1}]$ of the nodes of G *feasible* for x_0 if for all $1 \leq i \leq n-1$ there exists a path in G from x_0 to x_i composed only of nodes whose index is smaller than i; let $\Pi(G, v)$ denote the set of all permutations feasible for node v, and $\Pi(G) = \cup_{v \in V} \Pi(G, v)$ the set of all feasible permutations of the nodes of G. Then, for a given graph G, each sequential solution protocol $P \in \mathcal{P}$ uniquely defines a set $F_P(G)$ of n feasible permutations, one for every possible choice of the home base. Conversely, any set $F \subseteq \Pi(G)$ of n feasible permutations, each for a different node of G, corresponds to the BV-free execution sequences of some sequential solution protocol $P_F \in \mathcal{P}$ in G. That is, if $\alpha = [x_0, x_1, x_2, ..., x_{n-1}] \in F \subseteq \Pi(G)$ then $\alpha = P[G, x_0]$ for some $P \in \mathcal{P}$.

In other words, to determine a spread-optimal decontamination strategy for G, it suffices to determine for each h a feasible permutation α for h such that $\rho(\alpha) = \rho(G, h)$.

Before proceeding let us establish an obvious but important property of residual degrees in feasible permutations.

Lemma 7. *Given a permutation $\alpha = [z_0, z_1, ..., z_{n-1}]$ feasible for z_0, let also $\alpha_{i,j} = [z_0, z_1, ..., z_{i-1}, z_j, z_i, z_{i+1}, ..., z_{j-1}, z_{j+1}, ..., z_{n-1}]$ be feasible for z_0, where $0 < i < j \leq n-1$. Then $\rho(z_l, \alpha_{i,j}) \leq \rho(z_l, \alpha)$, for all $l \neq j$.*

Proof. Permutation $\alpha_{i,j}$ is the obtained from α by moving z_j immediately before z_i and leaving the rest unchanged. Since $\alpha_{i,j}$ is feasible for z_0, for each z_p ($0 \leq p \leq n-1$ there is a path from z_0 to z_p composed only of predecessors of z_p in $\alpha_{i,j}$. This means that the the number of neighbours of z_p following it in $\alpha_{i,j}$ (i.e., its residual degree in $\alpha_{i,j}$) is the same in both α and $\alpha_{i,j}$ for $0 \leq p \leq i-1$ and for $j+1 \leq p \leq n-1$. Furthermore, since z_j appears before z_i in $\alpha_{i,j}$, for $i \leq p \leq j-1$ the residual degree of z_p in $\alpha_{i,j}$ is the is either one less than or equal to its residual degree in α, depending on whether or not $(z_i, z_p) \in E$ (i.e., they are neighbours). □ □

3.3 Summary

In this chapter, the basic terminology and the model for the Black Virus Decontamination problem are introduced. Fundamental properties are established, as well as bound on the costs of solving the BVD problem. Monotone property is the necessary condition for spread optimality, and the basic bounds on BV spreads and the team size of agents are also formalized.

Chapter 4

General Strategy and Tools

Our objective is to decontaminate the network from the BV. The primary goal is to do so optimally, that is with minimum number of casualties (*spread optimality*). We would like to achieve this goal using as few agents as possible (*size optimality*). We now present a general strategy that leads to the design of BVD solutions protocols that are both spread and size optimal.

4.1 The overall Strategy

The strategy decomposes the BVD process into two separate phases: *Shadowed Exploration* and *Surrounding and Elimination*. The task of the first phase is to locate the BV, and the task of the second phase is to decontaminate/clean the BV and its clones. Agents discover the unknown location of the BV by exploring the network with the constraint of minimizing infections.

The order of nodes to be explored, called *Search Sequence (SS)*, can be determined offline (i.e., before exploration starts) or online (i.e., computing while exploring) depending on agents' knowledge on the map/topology of the graph.

In the case that agents have the full map of the graph available, they could clearly compute the best possible search sequence beforehand, simply by comparing all possible sequences of exploration (not necessarily sequential) and choosing the strategy that achieves optimality. On the other hand, if the agents have absolutely no knowledge of the topology except for the node in which they reside and the explored area,

it is easy to see that it is impossible for them to select online an optimal sequence. Indeed, the very first node explored from the home base, might be the one that causes the highest possible residual degree and there would be no way for the agents to know how to avoid that choice. In the following we assume that the full map is not given, but that nodes have knowledge of their neighbours and their degree: i.e., node x knows its own degree and the degree of each $y \in N(x)$.

In this situation, the agents have to calculate the SS while exploring the graph. We remind that G_{ex} (V_{ex}) and G_{ux} (V_{ux}) indicate the explored and unexplored portions (nodes) of the graph at any stage of the exploration. A target (i.e., a candidate node to be added to SS and explored) is chosen from the nodes in G_{ux} which are direct neighbours of G_{ex}, called *frontiers or frontier nodes*.

The states of nodes during the exploring and cleaning process are defined to facilitate the design and description for the solution protocols in the following chapters. The states of a node include "unvisited", "visited" (i.e., "cleaned"), "shadowed", and "contaminated". Figure 4.1 shows the node state transitions.

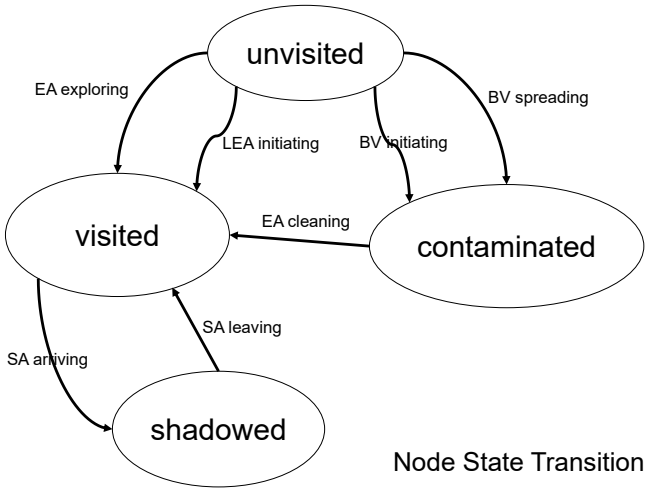

Figure 4.1: Node State Diagram

If a graph contains only one BV, once the BV node is triggered, all locations

of exposed BVs are known to agents. Agents need to clean all exposed BV nodes sequentially. However, if there are more than one BV in the graph and their locations are not known, agents have to continue to explore all the BVs until all nodes of the graph are explored and all BV clones are cleaned.

In the following we describe the main techniques and modules employed during both phases.

4.2 Shadowed Exploration

First note that, to minimize the number of agents loss, the exploration of a target node, v, from a previously explored/visited and proved safe node u, is performed using *safe exploration*, also called *cautious walk*. This technique is executed by at least two agents, the *Explorer Agent* (EA), and the *Leader Explorer Agent* (LEA) both residing at u. The exploring agent EA goes to v to explore it. If EA returns both agents proceed to v. If instead v was a BV node, LEA becomes aware of that because a BV arrives through that link instead of EA; in this case, different actions will be taken instead of continue exploring based on previously computed search sequence (route).

To insure monotonicity (thus to minimize the spread of the virus), before exploring a new target node, the previously explored nodes must be protected from the escaping BVs due to triggering a BV in a target node. Protecting the front line which separates the explored and unexplored graphs would require too many agents. To utilize the minimum number of agents to achieve monotonicity, a dynamic protection mechanism is adopted, i.e., only protecting those nodes which need protection when exploring a specific target. In other words, before the exploration by EA, some agents are employed to guard the previously visited neighbours of the node currently under exploration. This technique is called *shadowing*. In the following, we use the term *Shadow Agent* (SA) to indicate an agent who plays the shadowing role to prevent

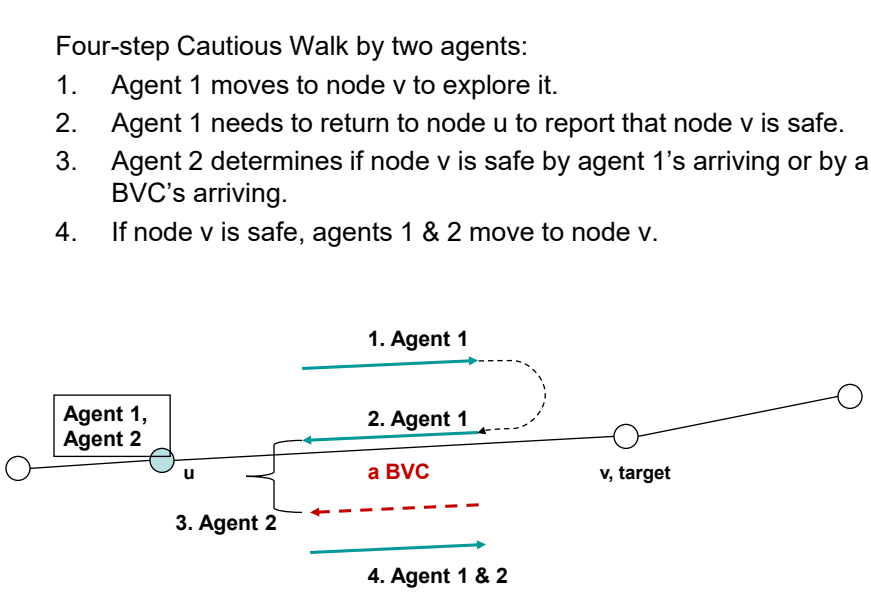

Figure 4.2: Cautious Walk

re-contamination of the explored nodes. Note that LEA, in the exploration traversal, also plays the role of a shadow agent. The exploration performed by EA, LEA, and SA's is thus called *Shadowed Exploration*.

More precisely, in all proposed algorithms, regardless of the network topology, the execution, until the BV is found, consists of a sequence of *exploration steps*. Each step comprises of an ordered sequence of operations:

1. At the beginning of each step, EA and LEA are at a safe node c_u, i.e., current node.

2. All agents determine the new node v to be explored among the neighbours of G_{ex} as well as the set $N_{ex}(v) = \{v_0, v_1, ..., v_q\}$ of the previously explored neighbours of v, and let $u = v_0$.

3. Let $\{S_1, S_2, ..., S_p\}$ be the shadow agents; note that, by construction, in each of

the examined classes of networks, $p \geq q$. Each $v_i \in N_{ex}(c_u)$ is assigned to a SA, S_i, $1 \leq i \leq q$; $u = v_0$ is assigned to LEA and to $S_{q+1}, ..., S_p$. EA moves with LEA to u as well and other agents move to the assigned nodes.

4. EA moves to v.

Note that the assignment of nodes to SAs can be calculated according to optimization criteria (i.e., v_i is vertex in $N_{ex}(v)$ closest to S_i, where ties are broken using identities) so to minimize the number of movements. How the proper sequencing of these operations is ensured depends on whether or not the system is synchronous.

Synchronous Agents

It takes one time unit for each movement (by agent or BV); computing and processing is negligible (instantaneous).

Let t_j denote the time when the j-th step (i.e., exploring j-th node) starts, and let $\hat{t}_j$ denote the time when EA moves to v in that step. Initially (i.e., at time t_1) all agents are at the homebase, and know the position of every other agent; all the algorithms maintain the invariant that, at time t_j, all agents know the position of every other agent and that the j-th step is starting.

Let d_i be the distance between the current position of S_i at time t_j and the assigned node v_i, and let $d = Max\{d_i\}$. This means that, by the time $t_j + d$ all nodes have reached their position for this step. Hence EA moves to $v = next(c_u)$ at time $\hat{t}_j = t_j + d$, when all previously explored neighbours of v are protected by agents. By time $\hat{t}_j + 2 = t_j + d + 2$ all agents know whether the BV has been detected: a BV clone will arrive at the locations of the agents at that time if and only if the BV was at v. If the BV is not detected, step $j + 1$ can start at time $t_{j+1} = \hat{t}_j + 3 = t_j + d + 3$ with LEA moving to v. At this time, v is named as c_u and v will be used for next target

node to be explored. Notice that all agents know both t_{j+1} and the position of every other agent at that time. If the BV is detected, the second phase of the protocols is started.

Asynchronous Agents

It takes a finite but unpredictable amount of time to perform any operation. Communication occurs when two agents meet.

In an *asynchronous* system, the LEA will act as overall coordinator, responsible for the communication with and coordination of EA and the shadow agents.

Let t_j denote the time when the j-th step starts. Initially (i.e., at time t_1) all agents are at the homebase; all the algorithms maintain the invariant that, at time t_j, LEA knows the positions of all the other agents and that the j-th step is starting.

Starting at time t_j, LEA moves sequentially to notify all SAs of the new step. To notify S_i $(1 \leq i \leq p)$ LEA moves to the position currently occupied by S_i; it communicates to S_i its assigned destination node v_i (where $v_l = v$ for $l > q$); both S_i and LEA move (asynchronously and independently) to v_i; when LEA arrives at v_i, it waits until S_i arrives. Once all SA have been notified and have moved to the assigned positions, LEA moves to u, and it instructs EA to move to v.

If BV is not at v, upon arriving at v, EA returns at u and notifies LEA that a new step must be performed; both LEA and EA move to v; the time when they both arrive is called t_{j+1}, v is named as c_u, and the $(j+1)$-th step starts.

If BV is at v, when EA arrives there, it is destroyed and it causes BV to move to all neighbours of v, including u where LEA is waiting. When this occurs, the second phase of the protocol starts. Also in that phase, LEA will coordinate the movements of the agents.

Based on the Property of monotonicity 3.2.1 and by the shadowed exploration

module just described, we immediately have that $G_{ex}(t)$ is continuous. BV nodes and unexplored nodes never cut $G_{ex}(t)$ into isolated pieces.

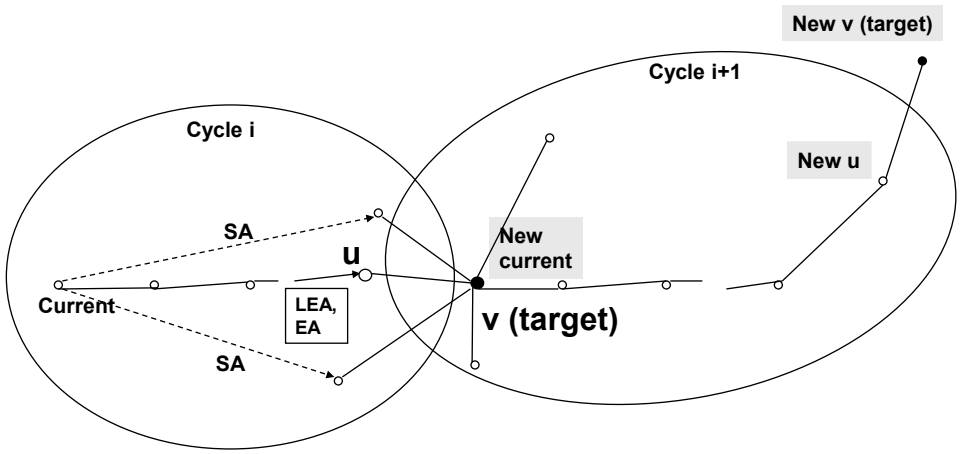

Figure 4.3: General Exploration Strategy

4.3 Surrounding and Elimination

Once the BV node is detected/triggered, the protocols enter this second phase. Again LEA coordinates other agents to sequentially clean the newly created BVs. A BV can be destroyed if there is already an agent at a node when the BV arrives. Thus, the only way to eliminate a BV from the system is to surround it completely and let an agent attack the BV by moving to the BV node. An agent is deployed to each safe neighbour of a BV node to surround it by instructed by LEA or by programmed in advance if the graph structure is simple enough to do so.

Once the BV has been detected and its clones have moved to the unexplored neighbours of its original location, in all proposed algorithms the surviving agents

start the second phase. In this phase, all BV nodes are surrounded and the permanent elimination of the BVs is performed.

Let v be the original location of BV, and let $N_{un}(v) = \{z_1, ..., z_k\}$ denote the set of the unexplored neighbours of v when EA enters u during the exploration. The second phase consists of a sequence of k *elimination steps* to permanently remove the BV from $N_{un}(v)$. In the j-th step, the following operations are performed:

1. Let $M(z_j) = N(z_j) \setminus N_{un}(v) = \{v_1, ..., v_{q_j}\}$ be the set of non-BV neighbours of z_j, and let $\{A_1, A_2, ..., A_{p_j}\}$ be the surviving agents at the beginning of stage j. To each agent A_i $(1 \leq i \leq p_j)$, is assigned node $v_i \in M(z_j)$, where $v_l = v_1$ for $l > q_j$. Note that by construction, $p_j > q_j$; hence more than one agent is assigned to v_1.

2. The agents move to the assigned nodes.

3. A_1 moves to z_j to permanently remove the BV from there.

Note that the assignment of nodes to agents can be calculated according to optimization criteria so to minimize the number of movements; e.g., v_i is vertex in $M(z_j)$ closest to A_i, where ties are broken using identities $(1 \leq i \leq q_j)$.

Also note that, in some of the protocols, the number of agents required for the first phase is greater than what is needed in the second phase. It is intended that these extra agents travel with *LEA* during the first phase; thus, they are co-located with *LEA* when the second phase starts.

How the proper timing of these operations is ensured depends on whether or not the system is synchronous.

Synchronous Agents

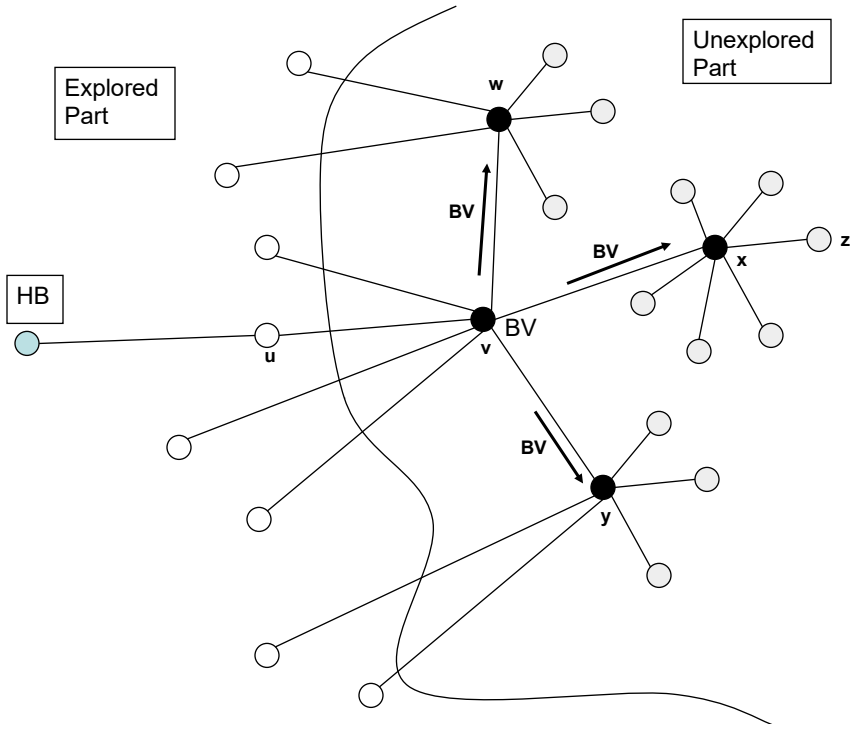

Figure 4.4: Exploration by Agents

Let t_1 the time when the BV is detected; notice that, by construction, all agents (except EA that is destryed by BV) will detect it at the same time. At that time the first step (which means cleaning one of the contaminated neighbours of v) of the second phase starts. Let t_j denote the time when the j-th step starts ($1 \leq j \leq k$), and let $\hat{t}_j$ denote the time when A_1 moves to z_j in that step; all the algorithms maintain the invariant that, at time t_j, all agents know the position of every other agent and that the j-th step is starting.

Let d_i be the distance between the current position of A_i at time t_j and the assigned node v_i, and let $d = Max\{d_i\}$. This means that, at time $t_j + d$ all nodes have reached their position for this step.

Agent A_1 moves to z_j at time $\hat{t}_j = t_j + d$, when all non-BV neighbours of z_j are protected by agents. At time $\hat{t}_j + 2 = t_j + d + 2$ a BV clone will arrive at the locations of the agents. Step $j + 1$ can start at time $t_{j+1} = \hat{t}_j + 3 = t_j + d + 3$; notice that all

agents know both t_{j+1} and the position of every other agent at that time, preserving the invariance.

Asynchronous Agents

In an *asynchronous* system, in addition to the synchronization and coordination of the agents, there is the additional difficulty of dealing with the unpredictable delay before a clone BV arrives at a node protected by an agent. This problem occurs immediately upon the arrival of EA to the original location u of the BV. Let W_j denote the set of nodes where there are agents at the beginning of step j. By definition, $W_j = M(z_{j-1})$ for $j > 1$; at the beginning of the first step, W_1 is instead the set of the locations of all agents when EA moves to v.

The coordinator role is played by LEA (called A_{p_j} in step j). The j-th step starts at time t_j when the BV clone arrives at the node where LEA is waiting, detecting the end of step $j-1$ (if $j > 1$) or of the first phase ($j = 1$). When this occurs LEA does as following.

First LEA moves sequentially to all nodes in W_j. In each of this nodes, it waits until the BV clone has arrived; it then communicates the assigned destination node(s) to the agent(s) resident there.

Then LEA moves sequentially to the nodes in $M(z_j)$ (in reversed order, ending in v_1). In each of this nodes, it waits until all the agents assigned to that node have arrived.

Finally, LEA notifies A_1 to move to z_j. At this point, LEA waits until the BV clone arrives, determining the end of the j-th step.

4.4 Summary

In this chapter, we described a general strategy based on two phases to achieve BV decontamination. The specific details of the strategy depend on the particular setting under consideration. In the next chapter, the focus is on the investigations on the BVD problem in three special classes of graphs: multi-dimensional *grid*, *tori*, and *hypercube*.

Chapter 5

Black Virus Decontamination in Special Graphs

5.1 Introduction

In this chapter, we concentrate on the specific case of multi-dimensional *grids*, *tori*, and *hypercubes*. Optimal protocols are developed for all these classes in presence of a single BV. For each specific special graph, there are some special structure properties, for example the degree of edge nodes are the same and the degree of internal nodes are the same, which allow us to adopt some special exploration routes to achieve minimum contamination during exploration. More details will be given when we deal with each individual graphs.

The protocols consist of two separate phases. In the first phase, exploration, the network is traversed until the initial location of the BV is determined. When this occurs, that location is cleared but all its unprotected neighbors have become BV nodes. In the second phase, surrounding and elimination, these BV nodes are permanently removed one by one ensuring that none of their safe neighbors are infected. The reason that there is a clear separation between two phases is there is only one BV in system. Once its location is located, all the locations of newly infected BV nodes are known and all other unexplored nodes are known to be safe.

5.2 BV Decontamination of Grids

5.2.1 Base Case: 2-Dimensional Grid

The general algorithm for BV decontamination of multi-dimensional grids makes use of the basic algorithm for the 2-dimensional case presented and analyzed in this section. Let G be a 2-dimensional grid of size $d_1 \times d_2$; without loss of generality, let the nodes of G be denoted by their column and row coordinates (x, y), $1 \leq x \leq d_1, 1 \leq y \leq d_2$.

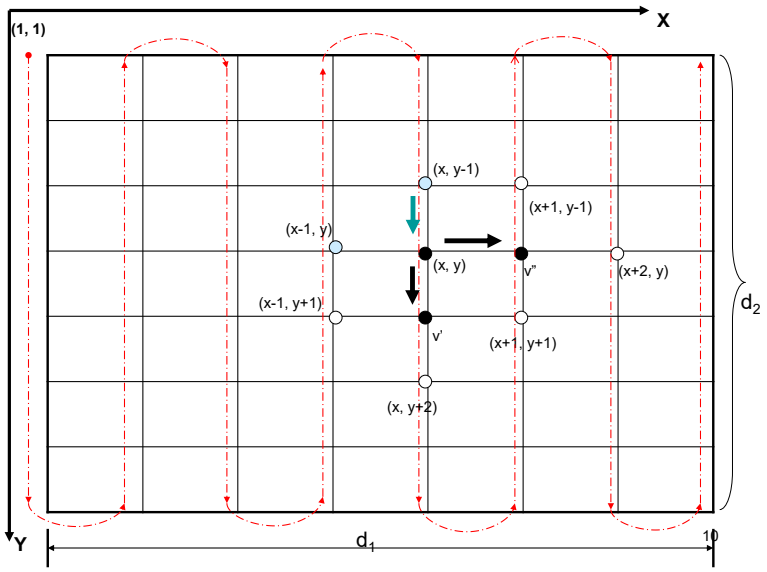

Figure 5.1: BV exploration in 2D

Agents LEA and EA start from the corner node $(1, 1)$ to perform a safe exploration of G with shadowing to locate the BV while protecting the nodes already visited. Once triggered, the BV spreads to its unprotected neighbours, and the elimination phase starts where agents are deployed to surround each newly created BV and to eliminate them.

The safe exploration proceeds along the columns of the grid in a snake-like fashion (see the dotted route in Figure 5.1), where odd columns are traversed downwards and

even columns are traversed upwards. EA leads the exploration, while LEA stays one node behind to detect the presence of the Black Virus, should it be triggered by EA. While the safe exploration proceeds, an appropriate number of shadowing agents are deployed to cover the already explored nodes that are neighbours of the node under exploration by EA so that when the BV is triggered, those nodes are protected and will not be recontaminated.

Let $v = (x, y)$ be the node under exploration, with $1 \leq x \leq d_1$ and $1 \leq y \leq d_2$. Let x^+ and x^- be defined as follows: $x^+ = x + 1$ (if $x \neq d_1$), $x^+ = x$ otherwise. Similarly: $x^- = x - 1$ (if $x \neq 1$), $x^- = x$ otherwise. An analogous definition holds for y^+ and y^-.

We can now define the set of explored and unexplored neighbours of node $v = (x, y)$:

$$N_{ex}(v) = \begin{cases} \{(x^-, y), (x, y^-)\} & \text{if } x \text{ is odd} \\ \{(x^-, y), (x, y^+)\} & \text{if } x \text{ is even} \end{cases}$$

$$N_{un}(v) = \begin{cases} \{(x^+, y), (x, y^+)\} & \text{if } x \text{ is odd} \\ \{(x^+, y), (x, y^-)\} & \text{if } x \text{ is even} \end{cases}$$

In a step of the shadowed exploration, the actions performed are as follows:

ALGORITHM *BVD-2G*: SHADOWED EXPLORATION

Agents EA and LEA are at safe node (x, y).

1. Agents compute $Next(x, y)$:

   If x is odd and $x \neq d_1$, $y \neq d_2$: $Next(x, y) = (x, y^+)$

   If x is even and $x \neq d_1$, $y \neq 1$: $Next(x, y) = (x, y^-)$

   If x is odd and $y = d_2$: $Next(x, y) = (x^+, y)$

   If x is even and $y = 1$: $Next(x, y) = (x^+, y)$

2. SAs move to occupy the nodes $N_{ex}(Next(x, y)) \setminus (x, y)$

3. EA moves to $Next(x, y)$.

4. If unarmed, EA returns to (x, y) and both EA and LEA move to $Next(x, y)$.

The shadowed exploration continues until EA meets the BV, at which time EA dies, LEA detects the presence of the BV (i.e., it receives a new virus from the node v just explored by EA), and the elimination phase starts.

At this point LEA, together with the shadowing agents fully cover $N_{ex}(v)$, and new BVs have spread to $N_{un}(v)$.

ALGORITHM *BVD-2G*: SURROUNDING AND ELIMINATION

BV originally in v now spread to $N_{un}(v)$

LEA and SAs are at the nodes $N_{ex}(v)$

1. LEA moves to v.

2. For each $u \in N_{un}(v)$:

   2.1 Agents move to each unoccupied $z \in \{N(u)\}$

   2.2 An agent moves to u to permanently remove that BV.

Note that while the two exploring agents traverse the first column (i.e., $x = 1$), no other shadowing is necessary; when they traverse subsequent columns (i.e., $x > 1$), a shadowing agent moves to the already explored neighbour (x^-, y) so to protect it, should the node (x, y) to be explored next contain a BV.

Theorem 2. *Protocol* BVD-2G*, performs a BV decontamination of a 2-dimensional Grid using $k = 7$ agents and 3 casualties.*

Proof. Let $v = (x, y)$ be the node containing the BV. When EA moves to v, it will be destroyed and the BV will move to all neighbours of v. When this happens, according to the algorithm, LEA is protecting the neighbour from which EA moved to v. If $x > 1$, the neighbour $(x - 1, y)$ is protected by a shadowing agent; if $x = 1$, then we are still in the first column and this neighbour does not exist. So, when the BV moves to the neighbours of v, the explored ones are protected and will not be infected by the BV; this means that the BV can safely move only to the unexplored neighbours of v, of which there are at most two. In other words, after v is explored, at most two BV nodes are formed; furthermore, since the grid is a triangle-free graph, they are not neighbours. This means that the BV nodes can be sequentially and separately surrounded and destroyed using at most six agents (including SA and LEA): one to enter a BV node and four to protect the neighbours. Hence, in addition to EA, at most two more agents die, and the total number of employed agents is seven. □

It is not difficult to verify that both the spread and the size of Protocol *BVD-2G* are optimal.

Theorem 3. *Let G be a 2-dimensional grid. Then, regardless of the number of nodes, $spread(G) = 3$ and $size(G) = 7$.*

Proof. Since the very first node visited by any algorithm could be a BV, at least two more BV will be generated; that is, for any solution protocol P, $spread_P(G) \geq 3$.

Of these two generated BVs at least one has four neighbours; furthermore, since G is triangle-free, these neighbours have no edges in common, and none of them is the other BV node. This means that at least four agents are needed in addition to the three casualties; that is, for any solution protocol P, $size_P(G) \geq 7$. □

Let us now consider the number of movements.

Theorem 4. *Protocol* BVD-2G*, performs a BV decontamination of a 2-dimensional grid of size n with at most $9n + O(1)$ movements in time at most $3n$.*

Proof. Let $v = (x, y)$ be the BV node, and let the size of the grid be $n = d_1 \times d_2$.

Let us first consider the number of movements performed during the shadowed exploration. The travelling distance from the home base to v is $d = (x-1)d_1 + x + y$. The shadowing agent follow the same path, except when EA and LEA traverse the first column (that does not need shadowing), so the number of movements for the shadowing agent is bounded by $d-1$. The number of movements is then $3(d-1)+1$ for EA, $(d-1)$ for LEA, and at most $(d-1)$ for the SA. The other four agents (needed for the surrounding and elimination) travel with LEA incurring in the same cost for a total of at most $4(d-1)$. We then have an overall cost of at most $9(d-1)+1$ movements for this phase.

Consider now the number of movements performed for Surrounding and Elimination. The new BV nodes v' and v'' are not connected, so they can be surrounded and cleaned sequentially. Each surrounding and cleaning requires a constant number of movements by 5 agents (4 to surround and 1 to clean the BV). Hence $O(1)$ movements are performed in this phase.

In total we then have that the number of movements is at most $9(d-1) + O(1) \leq 9n + O(1)$.

As for the time complexity. The time required for the exploration phase is equal

to the number of movements of EA, which is $3(d-1)+1$; the time required for the surrounding and elimination phase is constant. □

5.2.2 Multi–Dimensional Grid

Let M be a q-dimensional grid of size $d_1 \times \ldots \times d_q$ and let each node of M be denoted by its coordinates $(x_1, \ldots, x_q)$, $1 \leq x_i \leq d_i$. The algorithm, called *BVD-qG*, follows a general strategy similar to the one described in Section 5.2.1: a safe exploration with shadowing, followed by a surrounding and elimination phase.

We first describe the path followed for the safe exploration, and then the details of the surrounding and elimination phase.

Informally, the multi-dimensional grid is partitioned into $d_1 \times \ldots \times d_{q-2}$ 2-dimensional grids of size $d_{q-1} \times d_q$. Each of these grids is explored using the shadowed traversal technique of Protocol *BVD-2G* described in Section 5.2.1, in a snake-like fashion column by column, and returning to the starting point; in the following we refer to this exploration as *Traverse2*. From that starting point, the exploration proceeds to another grid, with a neighbouring starting point.

More precisely, given a sequence of $k < q$ integers $i_1, \ldots i_k$ with $1 \leq i_j \leq d_j$ (where $1 \leq j \leq k$), let $M[i_1 \ldots i_k]$ denote the $(q-k)$-dimensional sub-grid of M formed by the elements $\{(i_1, i_2, \ldots, i_k, x_{k+1}, \ldots, x_q)\}$ where $1 \leq x_l \leq d_l$ (where $k+1 \leq l \leq q$). The exploration traversal is achieved using the procedure TRAVERSE(V,K) described below, where V is a sequence of k integers. The symbol $V \circ j$ denotes the operation of adding integer j at the end of sequence V, resulting in V containing now $k+1$ integers. Initially, the procedure is invoked with $V = \emptyset$ and $k = 0$; at each end of the recursion, V is a 2-dimensional grid of size $d_{q-1} \times d_q$.

Figure 5.3 visualizes the traversal on a 3-dimensional Grid M with size $d_1 \times d_2 \times d_3$. M is explored by sequentially traversing the 2-dimensional grids $M[1], M[2], \ldots, M[d_1]$.

```
TRAVERSE(V,K)

if (q − k) = 2 then Traverse2(M[V])
else
  for 1 ≤ j ≤ d_{k+1}
      Traverse(V ∘ j, k + 1)
```

Figure 5.2: Traverse q-D grid

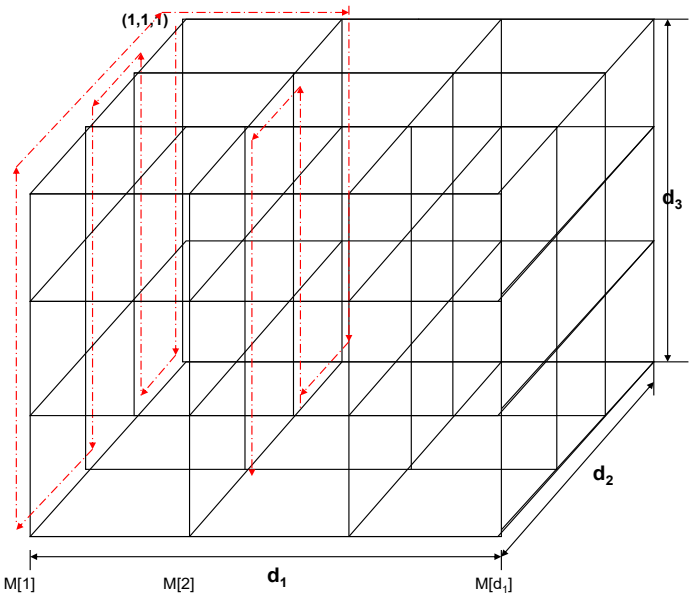

Figure 5.3: BV exploration in 3D

During the traversal, before exploring a node $v = (x_1, \ldots, x_q)$ $(1 \leq x_i \leq d_i)$ from a node u, the SAs move to the already explored neighbours of v to protect them; note that u is already protected by LEA. The set of these nodes is $N_{ex}(v) =$

$\{(x_1^-, x_2, ..., x_q), (x_1, x_2^-, ..., x_q), ..., (x_1, x_2, ..., x_q^-)\} \setminus \{v\}$, where

$$x_i^- = \begin{cases} x_i - 1 & \text{if } i \leq q-1 \text{ and } x_i > 1 \\ x_i - 1 & \text{if } i = q \text{ and } x_i \text{ is odd and } x_i > 1 \\ x_i + 1 & \text{if } i = q \text{ and } x_i \text{ is even and } x_i < d_i \\ x_i & \text{otherwise} \end{cases}$$

Once EA visits the BV node (and is destroyed there), the LEA and the SAs become aware of the location of the new BV nodes. These are precisely the unexplored neighbours of the original BV node v and, because of the structure of the traversal, their location is precisely determined knowing the coordinates of v; in fact, the set of unexplored neighbours of $v = (x_1, x_2, ..., x_q)$ is $N_{un}(v) = \{(x_1^+, x_2, ..., x_q), (x_1, x_2^+, ..., x_q), ..., (x_1, x_2, ...$ $\{v\}$, where

$$x_i^+ = \begin{cases} x_i + 1 & \text{if } i \leq q-1 \text{ and } x_i < d_i \\ x_i + 1 & \text{if } i = q \text{ and } x_i \text{ is odd and } x_i < d_i \\ x_i - 1 & \text{if } i = q \text{ and } x_i \text{ is even and } x_i > 1 \\ x_i & \text{otherwise} \end{cases}$$

Thus, once v is identified, sequentially, $2q$ agents surround each node $u \in N_{un}(v)$ and an additional agent enters it destroying the BV resident there and the instances that it generates.

Theorem 5. *Protocol* BVD-qG*, performs a BV decontamination of a q-dimensional Grid using* $3q + 1$ *agents and at most* $q + 1$ *casualties.*

Proof. Let $v = (x_1, \ldots, x_q)$ $(1 \leq x_i \leq d_i)$ be the node containing the BV. When EA moves to v, it will be destroyed and the BV will move to all neighbours of v. When

this happens, according to the algorithm, all the neighbours in $N_{ex}(v)$ are protected (either by LEA or by a SA) and will not be infected by the BV; this means that the BV can safely move only to the neighbours in $N_{un}(v)$. Since $|N_{un}(v)| \leq q$, there will be at most q new BVs. Since the q-grid is a triangle-free graph, these nodes have no edges in common. This means that these BV nodes can be (sequentially and separately) surrounded and destroyed using at most $3q$ agents (including SAs and LEA): one to enter each BV node, and $2q$ to protect the neighbours. Hence, in addition to EA, at most q agents die, and the total number of employed agents is $3q + 1$. □

It is not difficult to verify that the spread and the size of Protocol *BVD-qG* are both *optimal*.

Theorem 6. *Let M be a q-dimensional grid. Then, regardless of the number of nodes, $spread(M) = q + 1$ and $size(M) = 3q + 1$.*

Proof. Since the very first node visited by any algorithm could be a BV, at least q more BV will be generated; that is, for any solution protocol P, $spread_P(M) \geq q+1$. Of these generated BVs at least one has $2q$ neighbours; furthermore, since M is triangle-free, these neighbours have no edges in common, and none of them is a BV node. This means that at least $2q$ agents are needed in addition to the $q+1$ casualties; that is, for any solution protocol P, $size_P(G) \geq 3q + 1$. □

Let us now consider the number of movements performed by the agents.

Theorem 7. *A q-dimensional Grid of size $d_1 \times d_2 \ldots \times d_q$ can be decontaminated with at most $O(qn) = O(m)$ movements and $\Theta(n)$ time.*

Proof. Since the length of the traversed path until the BV is found is $O(n)$ in the worst case, the number of movements by LEA, EA and each SA is $O(n)$. Since there

are at most q shadowing agents, the total number of movements until the BV is found is $O(qn)$ in the worst case. At that point, there are at most q BV nodes, which are surrounded and eliminated sequentially. Each such node is surrounded by at most $q+1$ agents, each performing $O(1)$ movements, which gives $O(q^2)$ movements in total for surrounding and elimination. Since $q < n$, the bound follows. The total time is obviously $O(n)$ for the first phase, and constant in the second. □

5.3 BV Decontamination of Tori

A Torus is a regular graph obtained from a grid by adding wrap-around links to the "border" nodes. In a $q-$dimensional torus every node has $2q$ neighbours.

The algorithm to decontaminate the BV in a q-dimensional torus, called $BVD\text{-}qT$, follows a strategy very similar to the one used for the q- dimensional Grid described in Section 5.2.2.

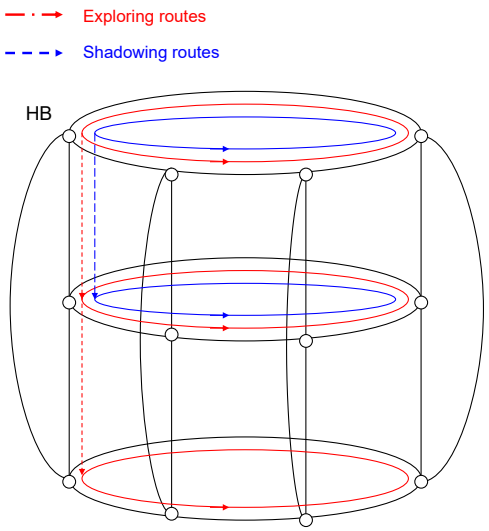

Figure 5.4: BV exploration in Tori

Let T be a q-dimensional torus of size $d_1 \times \ldots \times d_q$ and let each node of T be denoted by its coordinates $(x_1, \ldots, x_q)$, $1 \leq x_i \leq d_i$. Without loss of generality, let

$(1, 1, \ldots, 1)$ be the homebase. Informally, the multi-dimensional torus is partitioned into $d_1 \times \ldots \times d_{q-2} \times d_{q-1}$ *rings* of size d_q. The exploration procedure traverses a ring and, when back to the starting point, proceeds to another ring, with a neighbouring starting point. In Figure 5.4 the traversal is shown for $q = 2$.

Note that, because of the lack of borders, the spread of the BV might be higher in Tori than in Grids. For example, if one of the nodes visited during the first traversed ring is the BV, the BV will reach all its neighbours except the one occupied by LEA, inducing a casualty count of $4q$.

The Shadowing procedure is identical to the one described for the Grid: when node v is to be explored, all nodes in $N_{ex}(v)$ are occupied by a SA to protect them. Once the BV (say node v) is visited by EA, each node of $N_{un}(v)$ is surrounded sequentially and then eliminated like it is done for the Grid.

Theorem 8. *Protocol* BVD-qT*, performs a BV decontamination of a q-dimensional Grid using* $4q$ *agents with* $2q$ *casualties with at most* $O(qn)$ *movements and* $\Theta(n)$ *time.*

Proof. Let $N_{ex}(v)$ (resp. $N_{un}(v)$) denote the set of explored (resp. unexplored) neighbours of node v. The number of casualties is equal to $1 + |N_{un}(v)| \leq 2q$. To surround a BV node it always takes $2q$ agents, while for the shadowing $|N_{ex}(v)| = 2q - |N_{un}(v)|$ are deployed. Thus, in addition to the agents that die in the BVs, at most $\max\{2q, 2q - |N_{un}|(v)\} = 2q$ agents are used for shadowing and surrounding. The total number of agents employed by the algorithm is then: $1 + |N_{un}| + 2q \leq 4q$.

It is easy to see that the asymptotic number of movements as well as the time are the same as the ones determined for the q-Grid. □

It is not difficult to verify that both the spread and the size of Protocol $BVD\text{-}qT$ are optimal.

Theorem 9. *Let T be a q-dimensional torus. Then, regardless of the number of nodes, $spread(T) = 2q$ and $size(T) = 4q$.*

Proof. Since a q-dimensional torus is a triangle-free $2q$-regular graph, the optimality follows directly from Lemmas 2 and 3. □

5.4 BV Decontamination of Hypercubes

5.4.1 The Hypercube and its Properties

The hypercube is a classical topology for interconnection networks. A 1-dimensional hypercube is simply composed by two connected nodes, one labeled 0 and the other labeled 1. A q-dimensional hypercube can be constructed recursively by two copies of the (q-1)-dimensional hypercubes with links between each pair of corresponding nodes (e.g. nodes with the same labels) in the two sub-cubes. The name of each node in one of the sub-cubes is changed by prefixing it with a bit 0, and the name of the corresponding node in the other is changed by prefixing it with bit 1. A q-dimensional hypercube (q-Hypercube) has 2^q nodes, and $m = n \cdot \frac{q}{2} = \frac{1}{2} n \log_2 n$ links. Associating a q-bit string to each node as described above, an edge between two nodes u and v exists if and only if u and v differ in exactly one bit position (which is called the "dimension" of the edge).

The hypercube has several interesting properties that make it a desirable topology in many applications. The hypercube clearly admits an Hamiltonian tour. A particular Hamiltonian tour is given by following the so called *Gray code*, and it is described below.

The Gray code (also called reflected binary code) is a binary numeral system where two successive values differ in only one bit. The binary-reflected Gray code list G_q for q bits can be constructed recursively from list G_{q-1} by reflecting the list (i.e.

listing the entries in reverse order), concatenating the original list with the reversed list, prefixing the entries in the original list with a binary 0, and then prefixing the entries in the reflected list with a binary 1. Let $0G_i$ indicate list G_i where every element is prefixed by 0 (resp. 1), let $\overline{G_i}$ indicate list G_i reversed, and let $\circ$ indicate concatenation of lists. We then have: $G_q = 0G_{q-1} \circ 1\overline{G_{q-1}}$. For example $G_1 = [0, 1]$, $G_2 = [00, 01, 11, 10]$, $G_3 = [000, 001, 011, 010, 110, 111, 101, 100]$, and so on.

The Gray code provides a way to traverse a hypercube starting from node $(00 \ldots 0)$: each and every node is visited once and a sub-cube is fully visited before proceeding to the next.

The properties of the Gray code allows one to determine easily the position $Pos(u)$ of a node $u = (d_1 \ldots d_q)$ in the traversal of the hypercube following the Gray code. In fact, we have that $Pos(u) = (d'_1 \ldots d'_q)$ where $d'_1 = d_1$ and, for $1 < i \leq q$:

$$d'_i = \begin{cases} d_i & \text{if } d_{i-1} = 0 \\ 1 - d_i & \text{if } d_{i-1} = 1 \end{cases}$$

In a similar way, given a binary number $b = (b_1, \ldots b_q)$, the the b-th node $G(b)$ in the traversal of a q-hypercube following the Gray code (starting from 0) is $G(b) = (b'_1, \ldots b'_q)$, where $b'_1 = b_1$ and, for $1 < i \leq q$:

$$b'_i = \begin{cases} b_i & \text{if } b_{i-1} = 0 \\ 1 - b_i & \text{if } b_{i-1} = 1 \end{cases}$$

Particularly useful are the functions $Pred(u)$ and $Succ(u)$ that, given a node $u = (d_1, \ldots d_q)$ determine all its successors and all its precedessors in the Gray code traversal of the hypercube. Those functions are defined below, and they can clearly be computed locally by LEA: $Pred(u) = \{G_q(i) : i < Pos(u)\}$ and $Succ(u) = \{G_q(i) :$

$i > Pos(u)\}$.

Another useful function which is locally computable is $Next(u)$ which, given a node u, returns the next node in the Gray code traversal: $Next(u) = G(Pos(u) + 1)$.

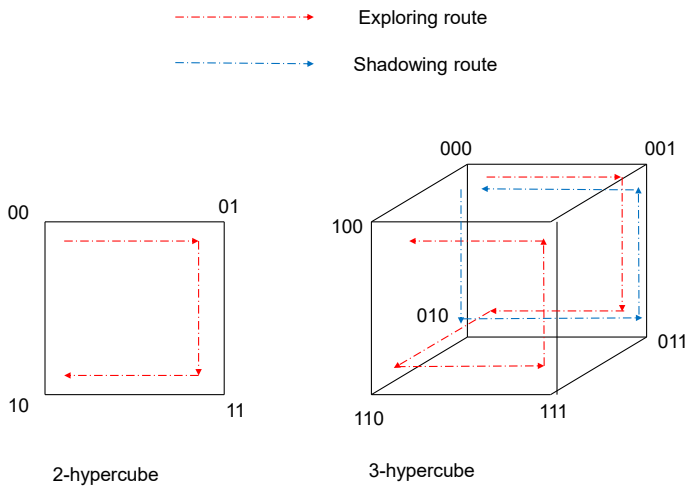

Figure 5.5: BV exploration in 2 and 3-Hypercube

5.4.2 Decontaminating a BV in q-Hypercube

Also in the case of the hypercube, the agents first perform a shadowed exploration, and then proceed to the elimination once the BV is detected and triggered. In the shadowed exploration, the two exploring agents EA and LEA traverse the hypercube following the Gray code, with the shadowing agents protecting at each step the nodes in the explored area adjacent to the node under exploration. EA goes ahead first, leaving LEA in the previously explored node, and if the BV arrives instead of EA, LEA realizes that the BV has been hit. In the particular case of a q-hypercube, hitting the BV triggers its propagation to the q neighbours, some of which are already explored (and protected by the SAs), while some are not explored yet. Those neighbours protected by the SAs will not be infected by the arrival of the new BV, the

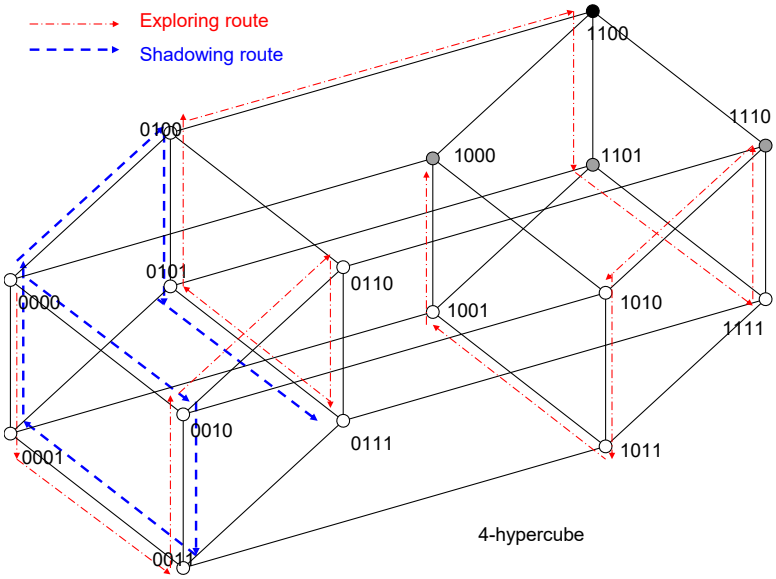

Figure 5.6: BV exploration in 4-Hypercube

others will become BV nodes; they will then be surrounded and eliminated (procedure Elimination and Surrounding).

Example: 3-Hypercube. As an example of shadowed exploration, let us consider in detail the case of the 3-Hypercube. The exploring agents follow the Gray code traversal, which in this case consists of the sequence of nodes: $000, 001, 011, 010, 110, 111, 101, 100$ and is depicted in Figure 5.5. Two shadowing agents (a and b) are needed overall, and they also perform a traversal of a subcube starting from 000. More precisely, no shadow is required while EA is visiting 001 and 011; when EA explores node 010, both a and b are still in node 000 to protect it, no shadow is needed when EA is exploring 110 (because both neighbours different from the predecessor are unexplored), the shadowing agents then proceed to 010 when EA visits 110, then one of the shadowing agents (say a) stays here, and b moves to 011 to protect it while EA visits 111, b then moves to 001 when EA is exploring 101, and finally b moves to 000 and a moves to 110 when EA is exploring 100. Note that synchronicity of the system allows the shadowing agents to correctly synchronize their movements with

the exploration agents. Also note that one of the shadowing agents (a in the example) initially follow the other shadowing agents without being needed ("inactive"). Shadowing agent a becomes needed ("active") when the exploration reaches node 100. In other words, in the 3-Hypercube one shadowing agent performs the reverse of the first half of the gray code, while the second shadow agent moves to the node connecting the two 2-Hypercubes to protect from the exploration of the last node.

In general, in the case of a q-Hypercube, one step in the shadowed exploration, with EA and LEA both at a safe node u, is described by the following algorithm:

ALGORITHM *BVD-qH*: SHADOWED EXPLORATION

Agents EA and LEA are at safe node u.

1. Agents compute $Next(u)$:
   $Next(u) = G(Pos(u) + 1)$. Let $u' = Next(u)$
2. Shadowing agents compute $N_{ex}(u') = N(u') \cap Pred(u')$
   Shadowing agents move to occupy $N_{ex}(u') \setminus u$
3. EA moves to $Next(u)$.
4. If unarmed, EA returns to u and both EA and LEA move to $Next(u)$.

When moving to occupy their computed positions, the paths followed by the $q-1$ shadowing agents can be intuitively described recursively as follows:

Shadow(Q, q)

- 1. For $q = 3$, the paths of the 2 shadowing agents are as described in the example above.
- 2. For $q > 3$: the q-Hypercube Q is decomposed along dimension q in the two $(q-1)$-subcubes Q^1 and Q^2

- 2.1. During the exploration of Q^1, $q-2$ shadowing agents follow the paths indicated by **Shadow**$(Q^1, q-1)$, while one is inactive.
- 2.2. During the exploration of Q^2, the $q-2$ shadowing agents already employed for Q^1 now follow the paths indicated by **Shadow**$(Q^2, q-1)$. The extra shadowing agent covers the path in Q^1 that corresponds to the exploration path of EA in Q^2.

As an example, the shadowed exploration routes in the 4-Hypercube by the 3 shadowing agents are shown in Figure 5.6. Indicated are also the timing constraints: label t on a SA movement has to be completed before the movement labeled t by EA takes place.

Notice that the starting point in each subcube depends on the exploration path followed by LEA, that is on the Gray code.

The shadowed exploration continues until EA meets the BV, at which time EA dies, LEA receives a new virus from the node v just explored by EA detecting the virus, and the elimination phase starts. At this point LEA, together with the shadowing agents SAs fully cover $N_{ex}(v)$, and new BVs have spread to $N_{un}(v)$.

ALGORITHM *BVD-qH*: SURROUNDING AND ELIMINATION

BV originally in v now spread to $N_{un}(v) = N(v) \cap Succ(v)$
LEA and SAs are at the nodes $N_{ex}(v)$

1. LEA moves to v.
2. For each $u \in N_{un}(v)$:
   2.1 Agents move to each unoccupied $z \in \{N(u)\}$
   2.2 An agent moves to u to permanently remove that BV.

5.4.3 Complexity analysis

The overall number of agents needed for the whole process depends on the position of the BV with respect to the traversal.

Theorem 10. *Protocol* BVD-qH*, performs a BV decontamination of a q-Hypercube using* $2q$ *agents and* q *casualties.*

Proof. One agent dies in the BV. Let v be the node containing the BV. $|N_{ex}(v)|$ agents are first employed to shadowing and then reused for surrounding and eliminating. Since the neighbours of the BV are disjoint, nodes in $N_{un}(v)$ are surrounded and eliminated sequentially. There are $q - |N_{ex}(v)|$ such nodes and for each of them, q agents are needed for surrounding and 1 for the elimination. Overall we then need $q - |N_{ex}(v)| + 1$ agents for elimination (these agents die and cannot be reused), and q for surrounding sequentially each of the nodes in $N_{un}(v)$. The total number of agents employed is then $2q + 1 - |N_{ex}(v)|$. Since $|N_{ex}(v)| \geq 1$, the bound follows. □

It is not difficult to verify that both the spread and the size of Protocol *BVD-qH* are optimal.

Theorem 11. *Let* H *be a q-Hypercube, we have that* $spread(H) = q$ *and* $size(H) = 2q$*.*

Proof. Since a q-Hypercube is a triangle free q-regular graph, the optimality follows directly from Lemmas 2 and 3. □

Let us now consider the number of movements.

Theorem 12. *Protocol* BVD-qH*, performs a BV decontamination of a q-Hypercube with at most* $O(n \log n)$ *movements and time* $\Theta(n)$*.*

Proof. First note that the number of movements performed by the exploring agents is $4(Pos(v)-1)+1$, where v is the node containing the BV. So we have a total of at most $4n$ movements for the exploring agents. The number of movements performed by each shadowing agent is bounded by n and the total number of shadowing agents employed is $q-1=\log n-1$. We then have that the total number of movements by the shadowing agents is $O(n\log n)$.

Finally, let us now consider the number of movements performed for surrounding and eliminating the new BVs. There are q such new BVs, to be treated sequentially. For each of them there are q neighbours to be occupied by the q agents. To reach any of these assigned nodes, an agent performs a constant number of movements. So, each new BV is surrounded and eliminated in at most $O(q)$ movements and all the BVs are eliminated in at most $O(q^2)=O(\log^2 n)$ movements. Adding all these costs, the movement complexity is then $O(n\log n)$.

As for the time complexity. The time required for the exploration phase is equal to the number of movements of EA, which is $O(n)$; the time required for the surrounding and elimination phase is constant. □

5.5 Summary

In this chapter, the BVD problems have been examined for three classes of interconnection networks: grids, tori, and hypercubes. The decontamination protocols for three classes (for all dimensions) have been presented, and their complexity in terms of spread of the virus and total number of agents is analyzed. It has been shown that the protocols are optimal in both measures. Although described for a synchronous setting, the solutions easily adapt to asynchronous ones (requiring only coordination among the shadows and the exploring agents), all the results hold also in this case, and the extra cost consists of an additional $O(n)$ movements in total for coordinating

the activities. An advantage of our solutions is that the agents use only local information to execute the protocol. A summary of the results is shown in the table below, where q denotes the dimension of the network, and n and m denote the numbers of nodes and of links, respectively. The results of this chapter appeared in [18, 19].

Table 5.1: Summary of Results in Special Graphs

Network	Spread	Size	Movements
q-Grid	$q+1$	$3q+1$	$O(m)$
q-Torus	$2q$	$4q$	$O(m)$
Hypercube	$\log n$	$2\log n$	$O(n \log n)$

In next chapter, the investigation of the BVD problem is extended to arbitrary graph. Solution protocols and complexity analysis will be preformed.

Chapter 6

Black Virus Decontamination in Arbitrary Graphs

In this chapter, we investigate the problem of BVD for arbitrary graphs in asynchronous environments in presence of a single BV. As in the previous chapter, the solution consists of performing a Shadowed Exploration phase to locate the BV, followed by a Surrounding and Elimination phase to clear the triggered BVs. We propose two different exploration protocols, both spread optimal and we analyze their complexity. In the end, an interesting connection is established between the solutions of the BVD problem and the problem of determining rooted acyclic orientations of unoriented graphs with minimum outdegrees. An algorithm is developed and its complexity have been analyzed. Some of the results of this chapter appeared in [20, 21].

6.1 Introduction

The general exploring strategy is based on the one described in Chapter 4. In fact, the solution still consists of performing a *Shadowed Exploration* phase to locate the BV, followed by a *Cleaning* phase to eliminate the cloned BVs.

With regard to the knowledge of the graph topology, in this chapter we assume 2-hop visibility, i.e., node v can "see" its neighbours $N^2(v)$. As mentioned earlier, full topological knowledge would imply the possibility of choosing the best possible exploration strategy offline in a centralized fashion, while we are interested here in an online distributed solution. In the following we consider asynchronous settings and both BV modes, fertile and sterile. In the case of fertile BV, the cloned BVs

are as powerful as the original (i.e., they can be further triggered to produce more BVs), in the case of sterile BV instead, the cloned BVs are weaker and cannot clone new BVs). The exploration strategy is the same for sterile and fertile BVs; only the cleaning phase is different and in the case of sterile BVs it simply consists in having the agents move directly to the newly triggered BVs.

6.2 The Solution Protocols

In asynchronous environment, there are no global clocks, and the duration of any activities (e.g., processing, communication, and moving) by agents, and the BV and its clones are finite but unpredictable. It is not possible to force concurrent exploration in asynchronous setting, so our proposed solutionst are sequential.

The main ingredient of our algorithms is the efficient determination, by the agents, of a minimum residual degree sequences $\pi =< x_0, x_1, ..., x_{n-1} >$. The construction of this sequence is done online, while exploring the graph in the shadowed exploration phase. In fact, in the shadowed exploration phase, starting from the homebase h, the agents explore sequentially the nodes of the graph, using agents (the *shadows*) to protect the already explored neighbours of the node to be visited, and thus ensure monotonicity of the solution. Once the BV node has been found, causing BV clones to spread to the unprotected neighbouring nodes, the cleaning phase start by finally and safely remove all the clones. The agents do not have a priori knowledge of the graph (e.g., a map); on the contrary, they have only local knowledge restricted to neighbours at distance at most two of the nodes where they have visited. The agents discover the network (and construct a map) as they move and explore. Indeed the shadow exploration phase is a distributed exploration and a map-construction process at the same time.

In the following we present and analyze two different exploration protocols, the

first is based on a simple "greedy" strategy, the other is employing a "threshold" strategy. We prove that both protocols achieve overall optimality; furthermore, the total number of agents used is asymptotically optimal.

6.3 Greedy Exploration

6.3.1 General Description

At each step of the shadowed exploration, the agents select a node, a *target*, in the map of the graph constructed so far. The map includes V_{ex} plus the distance-2 area in G_{ux}. The target is chosen among the unexplored neighbours of the already explored nodes (referred as *frontier or frontier nodes*), according to a greedy criterion: the selected node is the one with minimum residual degree; should there be multiple candidates, the one with the shortest distance[1] from the last target is chosen as the new target.

Once the target has been selected, agents (the *shadows*) move to occupy the explored neighbours of the target, so to protect them from BV clones. Once this operation is concluded, an agent (the *explorer*) moves to the target.

If the target was not a BV node, the current map of the network is updated so to include the information acquired from visiting the target, and a new step of the exploration process takes place. If instead the target was a BV node, the explorer dies, the BV is removed from the target, and BV clones move to all its neighbours. The clones arriving at the neighbours still unexplored (and thus unprotected by shadows), transform them into BV nodes; the clones arriving at the explored neighbours of the target are destroyed by the shadows located there.

Once this operation is completed, the cleaning phase starts. Note that at this time, the map contains all the exposed BV nodes and their neighbours. In the cleaning

[1] symmetry broken arbitrarily.

GREEDY EXPLORATION

(* Initialization *)
All agents initially at home-base h.
$M = (V_M, E_M) := N^2(h)$; (* initial Map is the 2-neighborhood of home-base *)
$V_{ex} := \{h\}$; (* only the home base is explored *)
$V_{un} := V_M \setminus \{h\}$; (* initial unexplored nodes *)
$B := h$; ((* only the home base borders the unexplored nodes*)
$Fr := N(h)$; (*unexplored frontier*)
$\pi := [h]$; (* the home base is the first in the search sequence *)
Found:= FALSE;

(* Iteration *)
while Found=FALSE
    Forall $v \in Fr$ **do** $r(v) := |\{u \in V_{un} : (u, v) \in E_M\}|$; (* residual degree of frontier *)
    Choose $v \in Fr$ such that $r(v)$ is minimum; (* selection of target v *)
    $N_{ex}(v) := \{u \in V_{ex} : (u, v) \in E_M\}$; (* explored neighbours of target *)
    Locate a shadow agent at each $u \in N_{ex}(v)$;
    Move an exploring agent to v;
    if $(v \neq BV)$ **then** (* update Map and variables *)
        $M := M \cup N^2(v)$;
        $V_{ex} := V_{ex} \cup \{v\}$;
        $V_{un} := V_M \setminus V_{ex}$;
        $B := \{x \in V_{ex} : \exists y \in V_{un}(x, y) \in E_M\}$;
        $Fr := \{x \in V_{un} : \exists y \in V_{ex}(x, y) \in E_M\}$;
        $\pi := \pi * [v]$; (*v is next in the search sequence*)
    else
        Found := TRUE
endwhile
Start ELIMINATION

Figure 6.1: GREEDY Exploration

phase, each BV is decontaminated by an exploring agent moving to BV's node after all its non-BV neighbours have been occupied by shadow agents.

Coordination and Synchronization. The algorithm is described in Figure 6.1. Missing from the descriptions are the details about how the coordination among the agents and the synchronization required among the different operations (e.g., all the shadow agents must be in place *before* the explorer can move to the target node) are achieved. Time-out cannot be employed since the system is asynchronous and time delays unpredictable and unbounded. The solution is however simple. One of the agents is used to perform the role of coordinator and synchronizer; we shall call this agent the *leader*. It is the leader's task to maintain a map of the explored network and its distance-two boundaries, with the location of all agents. Another agent, the *explorer*, will be the one visiting the unexplored nodes until it finds the BV (and is destroyed).

At the beginning of each step, the leader and the explorer are at the same node. The leader starts the step by selecting the next target, v, and determining its already-explored neighbours $N_{ex}(v) = \{z_1, ...z_p\}$ based on the information on the map. The explorer also computes the target v and the neighbour z_p; it then moves to z_p and waits there for instructions from the leader. The leader meanwhile sequentially goes to the current locations[2] $\{y_1, ...y_{p-1}\}$ of the shadow agents needed for this step. When at location y_i, the leader tells the agent there to become a shadow and to go to node $z_i \in N_{ex}(v)$ to protect it; the leader itself goes to z_i, and waits there until the shadow agent arrives; it then proceeds to the location y_{i+1} of the next needed shadow agent. To reduce the total number of agents used by the protocol, the leader will act as the shadow of the last node $z_p \in N_{ex}(v)$; thus, after the leader determines that a shadow

[2]Initially, all agents are in the home-base.

agent has arrived at z_{p-1}, it moves to z_p and acts as the shadow agent for that node. All movements of all the agents during this process is through the already explored part of the graph, and thus safe. Once both the leader and the explorer are at z_p, the leader gives the order and the explorer moves to the target node.

At this point two outcomes are possible: either the explorer returns to z_p or a clone arrives there. In the first case, the next step of the shadow exploration takes place, with the leader updating the map with the information reported by the explorer and repeating the process just described.

In the second case, the leader disables the arriving clone and starts the coordination of the cleaning phase. The leader starts the cleaning phase by updating its map to include the locations of the new BV nodes (the unexplored neighbours of the last target v). It then starts the process by going sequentially to the current locations of the agents required to clean the BV nodes, notifying them of the current map, and assigning a BV node to each of them to clean. Each of those cleaners independently reaches the assigned BV node (by first going to v and then to its destination) without any further need of coordination.

6.3.2 Complexity Analysis

Optimality. We now show that the BV-free exploration sequence created by protocol Greedy Exploration has minimal residual degree.

Theorem 13. *Let $\pi =< x_0, x_1, ..., x_{n-1} >$ be the BV-free exploration sequence created by procedure* Greedy Exploration *with $x_0 = h$. Then $\rho(\pi) = \rho[h]$.*

Proof. Let $\alpha =< h, y_1, ..., y_{n-1} >$ be an optimal exploration sequence, i.e., a feasible permutation with minimal residual degree $\rho[h]$. If $\alpha = \pi$, the theorem holds. Thus let us consider the case $\alpha \neq \pi$.

Let $i \geq 1$ be the smallest index such that $x_i \neq y_i$, and let $x_i = y_j$. Consider now the sequence $\alpha_{i,j} = [y_0, y_1, ..., y_{i-1}, y_j, y_i, y_{i+1}, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}]$

$= [x_0, x_1, ..., x_{i-1}, x_i, y_i, y_{i+1}, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}]$, where $y_0 = x_0 = h$. The feasibility of $\alpha_{i,j}$ follows from the feasibility of α and that, by construction, $y_j = x_i$ is a neighbour of some x_l with $l < i$. Thus, by Lemma 7 we have that $\rho(y_l, \alpha_{i,j}) \leq \rho(y_l, \alpha)$, for all $l \neq j$. That is, in $\alpha_{i,j}$ the residual degree of every node, except possibly for y_j, is not more than in α.

Consider now y_j in $\alpha_{i,j}$; and recall $\rho(y_j, \alpha_{i,j}) = \rho(x_i, \pi)$.

Consider the step of the algorithm in which x_i is chosen; obviously at that time x_i belongs to the frontier. Since y_i is a neighbour of some x_l with $l < i$, in that step also y_i and/or x_l belong to the frontier. By definition, $\rho(y_i, \alpha) \leq \rho(\alpha)$ is a node with residual degree not exceeding the current threshold and thus $\rho(x_l, \alpha) = \rho(y_l, \alpha) \leq \rho(\alpha)$; this implies that the algorithm would not have chosen x_i in that step if $\rho(x_i, \pi) > \rho(\alpha)$; hence, $\rho(y_j, \alpha) \leq \rho(\alpha)$.

Summarizing, in $\alpha_{i,j}$ the residual degree of every node, including y_j, is not more than in α; that is, also $\alpha_{i,j}$ has minimal residual degree $\rho(\alpha)$ and is thus an optimal exploration sequence.

In other words, if an optimal exploration sequence (e.g., α) coincides with π in the first i elements, then there exists an optimal exploration sequence (e.g., $\alpha_{i,j}$) that coincides with π in the first $i+1$ elements. By repeating this argument, the optimality of π follows; i.e., $\rho(\pi) = \rho[h]$. □

Spread and Size. Let us consider now the spreads (i.e., casualties) and the team size of agents required by the protocol.

Theorem 14. *Protocol,* Greedy Exploration*, causes* $\rho[h] < \Delta$ *spreads.*

Proof. Agents have to explore the BV node from one of the neighbours of the BV

node. No more spreads are generated during cleaning phase, so the total spreads are less than Δ. □

Theorem 15. *Protocol,* Greedy Exploration*, employs* $\Delta+1$ *agents in the* sterile model*, and* $\rho[h] + \Delta + 1 \leq 2\Delta$ *agents in the* fertile BV model*.*

Proof. During the shadow exploration phase, one exploring agent is used to explore the target, and one shadow is used to protect each of the already explored neighbours of the target; hence, since the leader is also a shadow, at most $\Delta+1$ agents are needed in this phase. Agent sizes for cleaning phase are different depending on weather the clones are sterile or fertile. If the clones are sterile, the total number of agents needed is at most $\rho[h]$ cleaners; since $\rho[h] < \Delta$, the claim holds; In case the clones are fertile, each new virus needs to be surrounded using at most Δ agents and eliminated by one exploring agent; since this phase is done sequentially, the same Δ agents (employed as shadows in the shadow exploration phase) can be used for all the surroundings, and exactly $\rho[h]$ additional agents are used for the elimination. Thus, in total $\rho[h]+\Delta+1 \leq 2\Delta$ agents suffice. □

Theorem 16. *Let* G *be a* d*-regular graph. Then any solution protocol needs at least* $d+1$ *agents if clones are sterile, and* $2d$ *if clones are fertile.*

Proof. In a d-regular graph G, $\rho(P[G,h]) = d-1$ for every home-base h. By Lemma 6, $spread(P,G,h) \geq \rho(P[G,h]) + 1$. At least one agent must survive. If clones are sterile, one agent is needed to clean each exposed BV; if clones are fertile, d agents are needed to surround the exposed BV. Therefore the theorem follows. □

Thus, by Theorem 15 and 16, it follows that Greedy Exploration is worst-case optimal with respect to the team size.

Agent Movements. In both the fertile and the sterile model, the shadowed exploration phase is exactly the same.

Theorem 17. *In Protocol,* GREEDY EXPLORATION*, the total numbers of agents movements is* $O(\Delta n^2)$.

Proof. Consider a step of the shadow exploration phase. Let c the current node from which agents explore the next target v. Based on algorithm, agents are all assembled to c before exploring next target. Let $N_{ex}(v) = \{z_1, ...z_p\}$ be v's already-explored neighbours; p shadows, SAs (one of them is the leader), and an explorer will be needed. LEA instructs $(p-1)$ SAs and one (1) EA to $|N_{ex}(v)|$ explored neighbours (called destinations). Assuming the distance (within G_{ex}) from current node c to one of the destinations, z_i, is $dis_{ex}(c, z_i)$, the total number of movements is $\sum_{i=1}^{p} dis_{ex}(c, z_i)$. LEA needs to follow all SAs to their destinations and make sure a SA arrives at every destination. The total number of movements by LEA is $dis_{ex}(c, z_1) + \sum_{j=1}^{p-1} dis_{ex}(z_j, z_{j+1})$.

Thus, in total, it performs $\sum_{i=1}^{p} dis_{ex}(c, z_i) + dis_{ex}(c, z_1) + \sum_{j=1}^{p-1} dis_{ex}(z_j, z_{j+1})$ movements. Let n_{ex} denote the number of nodes already explored at the start of this step; obviously for any explored nodes a, b, $dis_{ex}(a, b) < n_{ex}$. Hence the number of movements by the leader is less than pn_{ex}, while those by the other agents is at most pn_{ex}, for a total of no more than $2pn_{ex}$ movements. Since $p < \Delta$, $n_{ex} < n$, and the number of steps performed by the shadow exploration is less than n, the total movements during shadow exploration is $O(\Delta n^2)$.

Once the BV is detected, with sterile clones, LEA directly sends agents to exposed BVs to eliminate them. The movements are less than Δ. In case of fertile clones, *surround and eliminate* phase starts. In OnDec setting, LEA does not know the map of the G_{ux}, but the exposed BVs' locations are known, so LEA can do a safe traversal to locate all the neighbours of exposed BVs. The total cost of safe traversal is at

most $O(m = \Delta n)$. To surround and eliminate one BV, at most $O(n)$ movements are performed per agent to reach their shadow destination, and there are at most $(\Delta - 1)$ destinations, so a total of $O(\Delta n)$ movements are required. There are at most $(\Delta - 1)$ exposed BVs, so the total cost of surrounding and cleaning phase is $O(\Delta^2 n)$.

Adding movement costs for shadow exploration and cleaning phases, the total number of movements during shadowed exploration phase are the same for fertile and sterile models, the claim holds. □

6.4 Threshold Exploration

In this section we describe and analyze a simple variant protocol, called *Threshold Exploration*. This algorithm is also sequential and it chooses a single target at each step of the shadow exploration phase. The coordination and synchronization is exactly as in the previous algorithm. The main idea of the algorithm is to select as a target, among the nodes of the frontier, not the one with smallest residual degree (like in the "greedy" protocol), but rather one with residual degree not greater than a *threshold*.

Initially the threshold is set to the smallest residual degrees of the neighbours of the home-base. In subsequent steps, should all frontier nodes have residual degree above the threshold, the threshold is increased to the smallest of those residual degrees. When more than one frontier node is within the threshold, the one closest to the last explored node is chosen.

The algorithm is described in Figure 6.2, where the coordination and synchronization details are omitted.

6.4.1 Complexity Analysis

Optimality. We now show that π has minimal residual degree $\rho[h]$.

THRESHOLD EXPLORATION

(* Initialization *)
All agents initially at home-base h.
$M = (V_M, E_M) := N^2(h)$; (* initial Map is the 2-neighborhood of home-base *)
$V_{ex} := \{h\}$; (* only the home base is explored *)
$V_{un} := V_M \setminus \{h\}$; (* initial unexplored nodes *)
$Fr := N(h)$; (*unexplored frontier*)
$\pi := [h]$; (* the home base is the first in the search sequence *)
$\tau := Min\{|\{u \in V_{un} : (u,v) \in E_M\}| : v \in Fr\}$ (* initial threshold *)
Current := h; Found:= FALSE;

(* Iteration *)
while Found=FALSE
  Forall $v \in Fr$ **do** $r(v) := |\{u \in V_{un} : (u,v) \in E_M\}|$; (* residual degree of frontier *)
  $\tau := Max\{\tau, Min\{r(v) : v \in Fr\}\}$; (* update threshold *)
  Choose $v \in Fr$ closest to Current with $r(v) \leq \tau$ (* selection of target v *)
  $N_{ex}(v) := \{u \in V_{ex} : (u,v) \in E_M\}$; (* explored neighbours of target *)
  Locate a shadow agent at each $u \in N_{ex}(v)$;
  Move an exploring agent to v;
  if $(v \neq BV)$ **then** (* update map and variables *)
    $M := M \cup N^2(v)$; (* update map*)
    $V_{ex} := V_{ex} \cup \{v\}$; (* update explored nodes*)
    $V_{un} := V_M \setminus V_{ex}$; (* update known unexplored nodes*)
    $Fr := \{x \in V_{un} : \exists y \in V_{ex}, (x,y) \in E_M\}$; (* update frontier*)
    Current:= v; $\pi := \pi * [v]$;
  else
    Found := TRUE
endwhile
Start ELIMINATION

Figure 6.2: THRESHOLD Exploration

Theorem 18. *Let* $\pi =< x_0, x_1, ..., x_{n-1} >$ *be the exploration sequence created by protocol* THRESHOLD EXPLORATION *from home-base* $h = x_0$. *Then* $\rho(\pi) = \rho[h]$.

Proof. Let $\tau(i)$ be the value of the threshold τ when x_i was explored in the execution of THRESHOLD EXPLORATION generating π; thus, by construction, $\tau(j) \leq \tau(j+1)$ $(0 \leq j < n-1)$ and $\rho(\pi) = \tau(n)$

Let $\alpha =< y_0, y_1, ..., y_{n-1} >$ be an optimal exploration sequence, i.e., a feasible permutation for $h = y_0$ with minimal residual degree $\rho[h]$. If $\alpha = \pi$, the theorem holds. Thus let us consider the case $\alpha \neq \pi$. Let $i \geq 1$ be the smallest index such that $x_i \neq y_i$, and let $y_j = x_i$. Consider now the sequence $\alpha_{i,j}$ obtained by moving y_j before y_i; that is, $\alpha_{i,j} = [y_0, y_1, ..., y_{i-1}, y_j, y_i, y_{i+1}, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}] = [x_0, x_1, ..., x_{i-1}, x_i, y_i, y_{i+1}, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}]$, where $y_0 = x_0 = h$.

The feasibility of $\alpha_{i,j}$ follows from the feasibility of α and that, by construction, x_i is a neighbour of some x_l with $l < i$. Thus, by Lemma 7, we have that $\rho(y_l, \alpha_{i,j}) \leq \rho(y_l, \alpha)$, for all $l \neq j$. That is, in $\alpha_{i,j}$ the residual degree of every node, except possibly for y_j (i.e., x_i), is not more than in α and (due to the optimality of α) no more than $\rho[h]$. This also means that the first $i-1$ thresholds are all no more than $\rho[h]$; in particular, $\tau(i-1) \leq \rho[h]$.

Let now prove that $\rho(x_i, \alpha_{i,j}) \leq \rho[h]$. By definition, $\rho[h] \geq \rho(x_{i-1}, \alpha) = \rho(x_{i-1}, \pi) = \tau(i-1)$. Consider $\tau(i)$, i.e., the threshold when x_i is chosen constructing π; by construction $\tau(i-1) \leq \tau(i)$. If $\tau(i-1) = \tau(i)$, then trivially $\rho(x_i, \alpha_{i,j}) \leq \rho[h]$. Thus consider the case $\tau(i-1) < \tau(i)$. This case occurs only if the residual degree of all the nodes in frontier, after the exploration of x_{i-1}, is greater than $\tau(i-1)$; then $\tau(i)$ is set to the smallest among the residual degrees of the nodes in the frontier; in particular, $\tau(i) = \rho(x_i, \pi)$.

Since α and π concide for the first i elements, the number of unexplored neighbours of y_i after $x_{i-1} = y_{i-1}$ has been explored is precisely $\rho(y_i, \alpha)$ which is no more that $\rho[h]$,

since α is optimal. Observe that, when x_i is selected as the next target in π, also y_i belongs to the frontier (it follows from feasibility of α); thus, the residual degree of y_i at that time is at least $\tau(i)$. In other words $\rho(x_i, \alpha_{i,j}) = \rho(x_i, \pi) = \tau(i) < \rho(y_i, \alpha) \leq \rho[h]$.

Summarizing, in $\alpha_{i,j}$ the residual degree of every node, including y_j, is not more than in α; that is, also $\alpha_{i,j}$ has minimal residual degree $\rho(\alpha)$ and is thus an optimal exploration sequence.

In other words, if an optimal exploration sequence (e.g., α) coincides with π in the first i elements, then there exists an optimal exploration sequence (e.g., $\alpha_{i,j}$) that coincides with π in the first $i+1$ elements. By repeating this argument, the optimality of π follows; i.e., $\rho(\pi) = \rho[h]$. □

Spread and Size. The spread of BVs and the size of agents are the same as Protocol Greedy Exploration.

Theorem 19. *Protocol* Threshold Exploration*, causes $\rho[h] < \Delta$ spreads.*

Theorem 20. *Protocol* Threshold Exploration*, employs $\Delta + 1$ agents in the* sterile BV model*, and $\rho[h] + \Delta + 1 \leq 2\Delta$ agents in the* fertile BV model*.*

Agent Movements.

Theorem 21. *In Protocol* Threshold Exploration*, the total number of agent movements is $O(\Delta^2 n)$ in both fertile and sterile models.*

Proof. Consider a step of the shadow exploration phase in Threshold Exploration. All the candidate nodes under the same threshold can be explored. This relaxes the requirement for agents to move in G_{ex}, which requires shadow exploration be performed from one node with minimum residual degree to next node with minimum residual degree. This is equivalent to sequentially visiting multiple (unexplored) nodes

by traveling in G_{ex}, and the cost is $O(m) = O(\Delta n)$ (including sending SAs to their destinations). There are at most Δ thresholds, so total shadowed exploring cost is $O(\Delta^2 n)$.

If the target v is not BV node, the above shadow exploration continue by changing v to new c, assembling all SAs at their destinations to new c, and repeating the above process.

The total movements during shadow exploration phase are the same for the fertile and the sterile models. Once the BV is triggered, in the sterile model, LEA directly sends agents to exposed BVs to eliminate them. The movements are less than Δ. In the fertile model, the *cleaning* phase starts. There are at most $\Delta - 1$ spreads, and $O(n)$ movements are performed per agent to reach their target, so the total of movements are $O(\Delta^2 n)$.

Combining the moving costs at shadow exploration and cleaning stages, we conclude that agent movements is $O(\Delta^2 n)$. □

6.5 Rooted Acyclic Orientation with Minimum Outdegree

In this section, we establish an interesting connection between solutions of the BVD problem and the problem of determining rooted acyclic orientations of unoriented graphs with minimum outdegrees. As a consequence, our protocols provide a distributed optimal solution to this graph optimization problem.

6.5.1 Definitions and Properties

Given an undirected graph $G = (V, E)$, an *orientation* λ of G is an assignment of direction to each edge. Every orientation λ transforms G into a directed graph $\vec{G}_\lambda = (V, \vec{E}_\lambda)$. Let $\mathcal{D}(G)$ be the set of all directed graphs generated by acyclic orientations of G. An acyclic orientation λ of G is said to be *rooted* if $\vec{G}_\lambda$ has a single source (i.e.,

exactly one node of zero in-degree). Let $\mathcal{R}(G, v)$ be the set of all directed graphs generated by acyclic orientations of G rooted in v, and let $\mathcal{R}(G) = \cup_{v \in V}\{\mathcal{D}(G, v)\}$.

Given a directed acyclic graph $\vec{G}$, let $d^+(u, \vec{G})$ be the out-degree of u in $\vec{G}$; and let $d^+(\vec{G}) = Max_{u \neq v}\{d^+(u, \vec{G}_v)\}$ be the maximum out-degree among the nodes. An acyclic orientation λ of G is said to be *optimal* if $d^+(\vec{G}_\lambda) \leq d^+(\vec{G}')$ for all $\vec{G}' \in \mathcal{D}(G)$; similarly an acyclic orientation λ of G rooted in v *optimal* if $d^+(\vec{G}_\lambda) \leq d^+(\vec{G}')$ for all $\vec{G}' \in \mathcal{R}(G, v)$.

The interesting connection between BV-decontamination and optimal rooted acyclic orientations is provided by the property discussed next.

As well known, to any directed acyclic graph $\vec{G}$ corresponds a *partial order* "$\preceq_{\vec{G}}$" on the nodes of the graph, where $x \preceq_{\vec{G}} y$ if and only if there is a directed path from x to y. A *linear extension* of "$\preceq_{\vec{G}}$" is any total order "$<$" on the nodes consistent with $\preceq_{\vec{G}}$; that is, if $x \preceq_{\vec{G}} y$ then $x < y$. The sequence of the nodes ordered according to $<$ defines a unique permutation $X_< = [x_0, x_1, x_2, ..., x_{n-1}]$ of the nodes. Let $\Gamma(\vec{G})$ denote the set of all permutations defined by the linear extensions of the partial order $\preceq_{\vec{G}}$.

Theorem 22. *Let $\vec{G} \in \mathcal{R}(G, v)$ be a directed acyclic orientation of $G = (V, E)$ rooted in $v \in V$.*

1) $\Gamma(\vec{G}, v) \subseteq \Pi(G, v)$

2) $\forall\ X \in \Gamma(\vec{G}, v),\ \rho(X) = d^+(\vec{G})$.

Proof. Let $\vec{G} \in \mathcal{R}(G, v)$ and let $\preceq$ be the partial order it defines. Let $X =< x_0, x_1, ..., x_{n-1} >$ be the permutation defined by a linear extension $<$ of $\preceq$, where $x_0 = v$. First observe that X is feasible. In fact, since $\vec{G} \in \mathcal{R}(G, v)$, then every node is reachable by its only source $x_0 = v$; furthermore, since a linear extension is by definition consistent with the partial order it extends, every node z reachable by x in $\vec{G}$ (i.e., such that $x \preceq z$) appears after x in X. Hence, for each x_i there exists a path

in G from x_0 to x_i composed only of nodes whose index is smaller than i, which is the definition of feasibility. That is, $\Gamma(\vec{G}, v) \subseteq \Pi(G, v)$.

Furthermore, since all nodes reachable by x_i in $\vec{G}$ have an index higher than i in π, the residual degree r_i of x_i is precisely its outdegree in $\vec{G}$, i.e., $r_i = d^+(x_i, \vec{G})$. But this implies that the residual degree of a node y is the same in *every* permutation corresponding to a linear extension of $\preceq$. Thus, all permutations corresponding to the linear extensions of $\preceq$ have the same maximal residual degree, which is equal to the maximum out degree $d^+(\vec{G})$. □

In other words, in a directed acyclic graph $\vec{G} \in \mathcal{R}(G, v)$ rooted in v, every linear extension of $\preceq_{\vec{G}}$ defines a feasible permutation; additionally, all these permutations have the same residual degrees, which coincides with the maximum out degree in $\vec{G}$. As a consequence

Theorem 23. *Let $\vec{G} \in \mathcal{R}(G, v)$ be such that $\forall \vec{G'} \in \mathcal{R}(G, v), d^+(\vec{G}) \leq d^+(\vec{G'})$. Then $\forall\ X \in \Gamma(\vec{G}, v)$, $\rho(X) = \rho(G, v)$*

6.5.2 Algorithm Rooted Orientation

That is, the problem of finding an acyclic orientation of G with v as its only source and with the minimum out-degree possible is equivalent to the problem of determining an optimal feasible permutation for v.

In the previous sections we have seen two protocols that determine an optimal feasible permutation in a decentralized way. By exploiting the result of Theorem 23, we can use them to construct an optimal rooted acyclic orientation in a distributed way, using a single agent.

In our protocols, a BV-free exploration sequence $\pi =< x_0, x_1, ..., x_{n-1} >$ is created (different depending on the protocol) stating from the home-base x_0. When at x_i the

explorer has enough information to determine what the next target (i.e., x_{i+1}) is; the explorer moves sequentially from x_i to x_{i+1}. Consider now single agent protocol Rooted Orientation, described in Figure 6.3. In this protocol, the single agent performs exactly the same operations as performed by the explorer in the BV decontamination protocol being used. The only difference is that now the agent, when visiting a node for the first time (starting from the home-base), orients as outgoing all the edges connecting that node to its still unexplored neighbours.

The selection of x_{i+1} when at x_i will be different depending on whether we follow the greedy strategy of protocol OnDec Greedy Exploration (Protocol Greedy Rooted Orientation) with or the threshold strategy of protocol OnDec Threshold Exploration (Protocol Threshold Rooted Orientation). Regardless of the strategy, the result is an optimal rooted orientation.

Theorem 24. *Both* Greedy Rooted Orientation *and* Threshold Rooted Orientation *produce an optimal acyclic orientation rooted in the home-base.*

Proof. It follows from Theorems 13, 18 and 23. □

In a single agent computation, the important cost measure is the number of *movements* performed by the agent.

Let $\pi =< x_0, x_1, ..., x_{n-1} >$ be the sequence obtained by the strategy employed, and let $dis_{ex}(x_i, x_{i+1})$ be the shortest distance between x_i and x_{i+1} in the explored part of the graph when x_i was visited for the first time. Then the number of movements the explores performs is precisely

$$\sum_{i=0}^{n-1}(dis_{ex}(x_i, x_{i+1}))$$

Since $dis_{ex}(x_i, x_{i+1}) < i + 1$, it follows that the total number of movements for both protocols is less than $\frac{1}{2}n^2$. This upper-bound is however quite coarse.

ROOTED ORIENTATION

(* Initialization *)
Explorer initially at home-base x_0.
$M = (V_M, E_M) := N^2(x_0)$;
(* initial Map is the 2-neighborhood of home-base *)
$\vec{E} = \emptyset$.
$V_{ex} := \{x_0\}$; (* only the home base is explored *)
$V_{un} := V_M \setminus \{x_0\}$; (* initial unexplored nodes *)
(* Iteration *)
while $V_{un} \neq \emptyset$
    determine x_i
    move to x_i
    ORIENT(x_i)
    $M := M \cup N^2(x_i)$; (* update map*)
    $V_{ex} := V_{ex} \cup \{x_i\}$; (* update explored nodes*)
    $V_{un} := V_M \setminus V_{ex}$; (* update known unexplored nodes*)
endwhile

ORIENT(x)
    forall $y \in N(x) \cap V_{un}$ **do** $\vec{E} := \vec{E} \cup \{(\overrightarrow{x, y})\}$

Figure 6.3: Rooted Orientation

A more precise bound can be easily established for the THRESHOLD ROOTED ORIENTATION protocol as follows.

Theorem 25. *The total number of movements of* THRESHOLD ROOTED ORIENTATION *is less than* $2n\Delta$.

Proof. Once the threshold τ is set, the explorer performs a depth-first traversal visiting all nodes whose residual degree is within that threshold. This is what is implied by the requirement that x_{i+1} be the frontier node (satisfying the threshold) closest to x_i. In other words, the traversal stop only all the frontier nodes have a residual degree exceeding the threshold. At this point the threshold is increased (to the smallest among the residual degrees of the frontier nodes) and a new traversal is started

by the explorer. Clearly a spanning-tree of the already explored nodes is used at all times; hence the traversal with threshold τ requires at most $2n-1$ movements. Since the threshold is increased less than Δ times, the claim follows.

□

6.6 Summary

This chapter deals with the BVD problem for arbitrary graphs. The main challenge for solving the BVD problem in arbitrary graph is discussed. Exploring protocols are proposed based on residual degree to minimize the damage of the BVs during exploration. The solution protocols have been proved to be spread optimal in asynchronous environments. Complexity analyses in terms of agent size, and movements are performed. Finally, an interesting connection is established between the solutions of the BVD problem and the problem of determining rooted acyclic orientations of unoriented graphs with minimum outdegrees. An algorithm is developed to provide a distributed solution to this graph optimization problem. The complexity, i.e., agent movement, is analyzed and obtained.

Chapter 7

Multiple Black Virus Decontamination, MBVD

The problem of decontaminating a graph containing an unknown number of BVs at unknown locations is clearly difficult to solve efficiently. In this chapter we examine this problem in meshes, tori, hypercubes, and arbitrary graphs; we establish bounds on the number of casualties; we show how the problem can be solved without re-contamination, and sometimes optimally.

Since we do not know how many BV there are and where they are located in the network, every single node must be explored by the agents, regardess of the strategy used. In other words, any solution strategy must be a complete exploration strategy.

We first investigate the MBVD problem in arbitrary graph to obtain general solutions and results, then we apply them to special graphs by considering the properties of these individual graphs. Since we are dealing with *asynchronous* systems and since Lemma 5 holds regardless of the number of BVs, we can and will limit our focus to *sequential* solution protocols.

We consider both BV clone modes, sterile and fertile. In the case of fertile BV clones, the cloned BVs are as powerful as the original (i.e., they can be further triggered to produce more BVs); in the case of sterile BV clones, the cloned BVs are weaker (i.e., they cannot produce new clones).

7.1 Arbitrary Graphs: Sterile Clones

7.1.1 Observations

Consider online distributed solutions to the MBVD problem in arbitrary graphs when the BV clones are sterile. We first observe the following upperbound:

Theorem 26. *Let G be an arbitrary graph with n_{BV} black viruses producing sterile clones. Any monotone decontamination protocol G incurs* at most $\Delta(G)\ n_{BV}$ *casualties.*

Proof. By monotonicity of the protocol, a BV will not contaminate an explored node. Since a BV only contaminates its non-explored neighbours, then each BV contaminate with clones at most $\Delta(G) - 1$ nodes; since these clones are sterile, each will cause only one casualty; by adding the casualty done for the BV node itself, we obtain the claim. □

This upperbound can actually be tight.

Theorem 27. *There is an infinite number of graphs G with n_{BV} black viruses producing sterile clones where any monotone decontamination protocol incurs* at least $\Delta(G)\ n_{BV}$ *casualties.*

To prove this theorem, consider the following infinite family of of regular graphs (i.e., where all nodes have the same degree $\Delta(G)$). A graph $G(d, k)$, $d \geq 2$ and $k \geq 0$, in this family is composed of $2k$ complete graphs of dimension $d-1$, $G_1, G_2, ..., G_{2k}$ plus two single nodes x and y; there is a matching between the nodes of G_i and those of G_{i+1} for $i < 2k$; there is an edge between x and y; there is an edge between x and all nodes of G_1, and an edge between y and all nodes of G_{2k}; see Figure 7.1 for an example. It is to verify that $G(d, k)$ so defined has $n = 2k(d-1) + 2$ nodes and is d-regular.

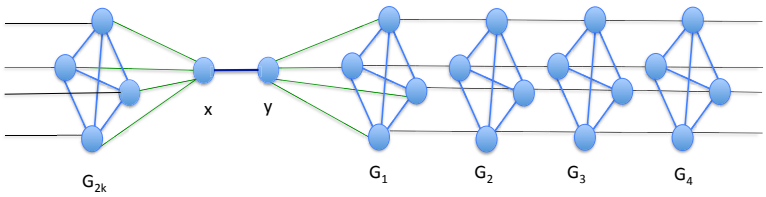

Figure 7.1: Graph $G(5, k)$

Theorem 28. *To decontaminate $G(k, d)$ from $n_{BV} \leq k$ black viruses producing sterile clones, the number of casualties required in the worst case is*

$$spread(G(k, d)) = d\ n_{BV}$$

Proof. Consider any monotone sequential decontamination protocol P starting from y as the homebase; there is a single BV in each of the clusters G_1, G_3,..., $G_{n_{BV}-1}$. In each of those clusters, the choice of which node is a BV is made by the adversary according to the following rule: when a cluster is visited for the first time, that node is declared a BV. This means that if cluster G_{2i+1} is visited for the first time the entire cluster will be infected; Furthermore, if the exploration came from G_{2i} (or y if $i = 0$), then also the neighboring node in G_{2i+2} (or x if $i = k-1$) will be contaminated; similarly, if the exploration came from G_{2i+2} (or x if $i = k-1$), then also the neighboring node in G_{2i} (or y if $i = 0$) will be contaminated. In other words, each BV will contaminate

$d-1$ nodes, and there is no overlap between these contamination; thus in total, there will be $n_{BV}(d-1)$ contaminations by sterile clones. Adding the n_{BV} casualties needed to clear the BVs, the theorem follow. □

Notice that, in this proof, the adversary was able to obtain the maximum possible number of casualties by placing the BVs separate from each other. Indeed the distance between the BVs can play a role on the total number of casualties.

The relationship of two or more BVs can be categorized as one of three cases with regard to the distance between them: 1) the BVs are direct neighbours (i.e., $distance = 1$); 2) the BVs share common neighbours (i.e., $distance = 2$); 3) the BVs don't have common neighbours (i.e., $distance > 2$). Let us call the neighborhood of a BV node its *Impacting Zone* (IZ). If we say *IZs* of two BV nodes do not overlap, it means the distance between the two BVs are greater than two.

By *Locality of Casualty (LC)*, we mean that the spread of a BV and the casualties it causes are not influenced by those of another BVs. Intuitively, LC is a condition enhanced by the distance between BVs, and thus it depends on IZ. Perhaps counter-intuitively, presence of LC may have an adverse effect on the number of casualties. Infact, in *absence* of LC there are savings: upon triggering a BV node, the clone sent to a BV neighbour will not increase the number of created casualties; similarly if two BV neighbours at distance two are visited, the common neighbour will be infected only once (because of monotonicity) even though two clones will be sent to it, and thus it will cost only one casualty and not two.

To see this more clearly consider a line graph L with the homebase at one end. If there is LC, then the exploration cost will be precisely $2n_{BV}$; on the other hand, if the distance between successive BV is one, then the exploration cost will be only $n_{BV} + 1$.

The negative effect of the presence of LC might however be mitigated by the

topology of the graph.

7.1.2 Decontamination without Map

We are interested in this section in online distributed solutions. With regard to the knowledge of the graph topology, we do not need full topological knowledge and again assume 2-hop visibility, i.e., node v can "see" its neighbours $N^2(v)$.

The strategy we use with multiple BVs when clones are sterile is to visit the graph sequentially, following the same minimum residual degree sequence $\pi =< x_0, x_1, ..., x_{n-1} >$ as in the greedy BV-free exploration of Section 6.2. The algorithm will actually be the greedy shadowed exploration with three basic modifications.

The first obvious modification is that we do not stop the exploration once we find a BV node. What we do instead is to record in our map where all the clones of that BV have moved to; we then proceed with the shadowed exploration, determining the next target in the sequence.

The second (and major) difference is when the next target u is a node where a BV clone has moved to (as just mentioned, we keep track of these sites). In a single-BV system, u is cleaned by having an agent a move from a safe neighbour v to u; in this way, it will disactivate the clone, and be destroyed in the process. In the multiple-BV system it is however possible that u is a BV node; in this case, the agent a will clean up u but BV clones will be sent to all neighbours of u, including v. This means that, to avoid recontamination of v, an agent b must be there when the clone arrives. Since we do not know whether u contains just a clone or a BV, this means that an agent b must be at v while a goes to u. The question then becomes: how long must b stay at v ? Notice that if u is a BV node, a will be killed, a clone will arrive at v and b will detect it and kill it. However, if u has only a BV clone, a will be killed, and nobody will go to v. Since the system is asynchronous, the problem of b determining after

time t whether or not a BV clone will be arriving is *undecidable*.

To overcome this difficulty, we exploit a simple mechanism of doubly-cautious walk when visiting a still unexplored node u where we know a BV clone moved there. The mechanism consists of using in addition to LEA (agent b) and EA (agent a) an additional agent (agent c), called verifying agent or VA, whose function is to contribute to verify whether or not u is a BV node. The way the mechanism works is to have first EA and then VA move to u while LEA stays at v. Observe that a, upon its arrival, will destroy the BV clone or original resident there, and will be destroyed; hence when c arrives, it will find no trace of other entities (BV, clone, EA); it then just returns to v. If u is a BV node, the arrival of a will send a BV clone to all of u's neighbours, including v; should this happen, the clone will arrive at v before c returns. In other words, once c has departed for u, within finite time, either a BV clone will arrive followed by c, or just c will return. Thus, LEA can detect whether or not u was a BV node and, if so, update the information of which unexplored nodes contain a clone if not a BV.

The third and final modification is in the calculation of the residual degrees of the nodes in the fronteer, needed to choose the target node to explore. In fact, if there is an edge leading from a fronteer node x to an unexplored node y which is in the clone list (i.e., $y \in V_{un}^{-}$) y is not considered in the calculation; this is because, if x is chosen as the target and it turns out to be a BV, then the clone it sends to y is irrelevant (i.e., it does not increases the number of nodes to be decontaminated) since y contains already a BV (clone or original). We denote by $\hat{\rho}$ such a modified residual degree.

The resulting protocol, MBV_Decontamination_Sterile, or MS for short, is shown in Figure 7.2. The number of casualties incurred by the execution of the algorithm depends on the number and location of the BVs in the graph. We can

MULTIPLE-BV DECONTAMINATION - STERILE

(* Initialization *)
All agents initially at home-base h.
$M = (V_M, E_M) := N^2(h)$; (* initial Map is the 2-neighborhood of home-base *)
$V_{ex} := \{h\}$; (* only the home base is explored *)
$V_{un} := V_M \setminus \{h\}$; (* initial unexplored nodes *)
$V_{un}^- := \emptyset$; (* initial unexplored nodes containing a BV or a BV clone *)
$Fr := N(h)$; (* unexplored frontier *)

(* Iteration *)
while $V_{un} \neq \emptyset$
    forall $v \in Fr$ **do**
        $r(v) := |\{u \in V_{un} \setminus V_{un}^- : (u, v) \in E_M\}|$; (* residual degree of frontier *)
    endfor
    Choose $v \in Fr$ such that $r(v)$ is minimum; (* selection of target v *)
    $N_{ex}(v) := \{u \in V_{ex} : (u, v) \in E_M\}$; (* explored neighbours of target *)
    Position a shadow agent at each $u \in N_{ex}(v)$;
    if $(v \in V_{un}^-)$ **then**
        Move an exploring agent followed by a verifying agent to v
    else
        Move an exploring agent to v;
    endif
    $M := M \cup N^2(v)$; (* update map*)
    $V_{ex} := V_{ex} \cup \{v\}$; (* update explored nodes*)
    $V_{un} := V_M \setminus V_{ex}$; (* update unexplored nodes*)
    if (v was a BV) **then**
        $V_{un}^- := (V_{un}^- \setminus \{v\}) \cup (N(v) \setminus V_{ex})$; (* update clone list *)
    endif
    $Fr := \{x \in V_{un} : \exists y \in V_{ex}, (x, y) \in E_M\}$; (* update frontier*)
endwhile

Figure 7.2: Multiple-BV Decontamination - Sterile

calculate them precisely.

Let $\pi =< x_0, x_1, ..., x_{n-1} >$, $x_0 = h$, be the minimum residual degree sequence of the algorithm, where the residual degree is computed with the third modification described above; and let $\xi : V \to \{0, 1\}$ be such that $\xi(x_i) = 1 \iff x_i$ is a BV. Then, by construction we have:

Theorem 29. *The number of casualties of Protocol* MS *is*

$$spread(MS, G) = \sum_{0<i<n} (\hat{\rho}(x_i, \pi) + 1)\xi(x_i)$$

From this, it immediately follows that:

Theorem 30. *Let G be an arbitrary graph contaminated by n_{BV} black viruses which produce sterile clones. Decontamination of G starting from h can be performed with*

$$spread(G) \leq (\rho(G, h) + 1)\ n_{BV}$$

Proof. Clearly $\hat{\rho}(G, h) \leq \rho(G, h)$. Since π is a minimal residual degree sequence starting from h, then, by definition, $\hat{\rho}(x_i, \pi) \leq \rho(G, h)$ for $0 < i < n$, and by Theorem 29, the claim holds. □

That is, each BV will cause a number of additional casualties which is at most the minimal residual degree of the graph.

7.1.3 Decontamination with Map and n_{BV}

If both a map of the network is available and the number n_{BV} of BVs is known to the agents, then it is simple, albeit computationally expensive, to derive an optimal algorithm for decontaminating the network from the BVs regardless of their unknown location. In this section we describe how.

Recall that any sequential protocol explores the nodes of the graph sequentially, generating a sequence of the nodes in the order in which they are explored the first time. As we already discussed, any feasible permutation $\pi =< x_0, x_1, ..., x_{n-1} >$ of the nodes, with $h = x_0$, describes a BV-free exploration sequence starting from homebase. Consider now the MBV decontamination protocol $P(\pi)$ where the target sequence is as specified by the feasible permutation π and where the execution is performed using the three modifications described in the previous section.

To calculate the cost in terms of casuaties incurred by this protocol in the worst case, we must consider the cost incurred if the BV were in a specified set of locations.

Let $Y_i =< x_{i_1}, x_{i_2}, ..., x_{i_{n_{BV}}} >$ be the ordered sub-sequence of π containing the initial locations of the BVs. Let

$$succ(x_{i_j}) = \{x_l \in N(x_{i_j}) \setminus Y_i) : l > i_j\} \cup \{x_{i_j}\}$$

denote the set composed of x_{i_j} and of its non-BV neighbours following it in the sequence π. Then the number of casualties incurred by protocol $P(\pi)$ if the BVs are in the location specified by Y_i is precisely

$$\Xi(\pi, Y_i) = |\bigcup_{1 \leq j \leq n_{BV}} succ(x_{i_j})|$$

The location of the BVs is however unknown. Any of the possible ordered n_{BV}-subsets of π are possible; let $\mathcal{Y}(\pi)$ be the set of all such subsets. This means that the number of casualties encurred by protocol $P(\pi)$ in the worst case is

$$\Xi(\pi) = \max\{\Xi(\pi, Y_i) : Y_i \in \mathcal{Y}(\pi)\}$$

In other words, given a feasible permutation $\hat{\pi}$, the decontamination protocol $P(\hat{\pi})$ is

an *optimal* solution if and only if $\Xi(\hat{\pi}) \leq \Xi(\pi)$ for any feasible permutation π.

At this point, to determine an optimal decontamination protocol, it is sufficient to compute $\Xi(\pi)$ for all feasible permutations π, and to determine one, say $\hat{\pi}$, for which this quantity is minimum: protocol $P(\hat{\pi})$ is then the desired solution.

COMPUTE_OPTIMAL_SOLUTION

(* Map of the network G and number n_{BV} of BVs are known *)
$\Pi :=$ set of all feasible permutation for G starting from h;
forall $\pi \in \Pi$ **do**
    $Y(\pi) :=$ set of all ordered n_{BV}-subsets of π;
    forall $Y \in Y(\pi)$
        compute $\Xi(\pi, Y)$;
    $\Xi(\pi) := \max\{\Xi(\pi, Y) : Y \in Y(\pi)\}$;
$\hat{\pi} := \pi \in \Pi : \forall \pi \in \Pi, \Xi(\hat{\pi}) \leq \Xi(\pi)$;
Protocol $P(\hat{\pi})$ is an optimal solution protocol;

Figure 7.3: Compute Optimal Solution

7.2 Arbitrary Graphs: Fertile Clones

When there are multiple BV and their clones are fertile, that is they are complete copies of the BV from which they originate, the decontamination problem becomes drastically more complex, especially in the number of casualties that can be incurred.

7.2.1 General Observations and Bounds

The first thing to observe is that the initial location of the BVs can have catastrophic consequences for the cost of decontamination.

Let V_{BV} be the set of BV nodes. The removal of V_{BV} and their incident edges from G will create a set of disjoint connected subgraphs $G_0, G_1, ..., G_{k-1}$; let G_0 be the subgraph containing the homebase h (see Figure 7.4). Each isolated component

is eventually contaminated.

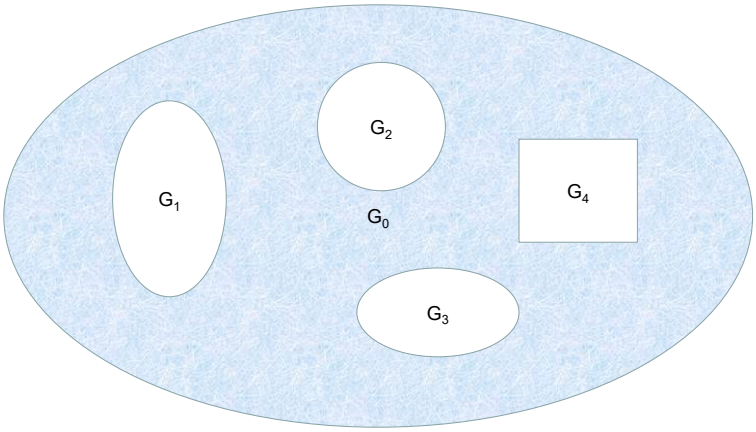

Figure 7.4: An example of separation

Theorem 31. *Decontamination of G requires at least*

$$|V_{BV}| + \sum_{0<i<k-1} |V_i|$$

casualties, where V_i is the set of vertices of G_i.

Proof. By definition, each G_i, $i > 0$, is completely separated from the rest by BVs. This means that, regardless of the protocol, to enter G_i, a BV node is first encountered. All the unexplored neighbours of this node in G_i become BVs. Thus, the boundary formed by BV nodes isolating G_i from the rest is pushed forward. In other words, every node of G_i will be eventually contaminated and need to be cleaned. Since all G_i are disjoint, and the original BVs need also to be cleaned, the claim holds. □

Notice that the above is a generalization of Properties 3.2.3 and 3.2.4 in case of mutiple BVs.

We call the condition when $k > 1$ that of a *separation*. An interesting separation is when $k = 2$, that is the graph can be partitioned into two connected subgraphs, G' containing the homebase and G'' containing all the BVs, and all edges between the two subgraphs have a BV as an endpoint (see Figure 7.5). We call this condition *single separation*; note that in this case, by Theorem 31, all nodes of G'' will become contaminated, regardless of the decontamination protocol.

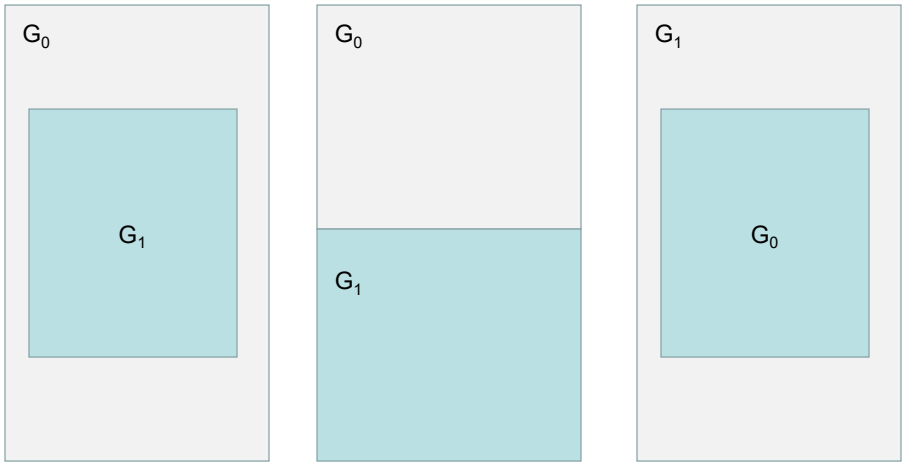

Figure 7.5: Examples of single separation (the dark blue subgraphs are BVs)

Even if initially there is no separation (i.e. $k = 1$), the graph can break into separate components *during the execution* of a decontamination protocol. Consider a sequential monotone decontaminination protocol P and consider its execution in G at time t. Let $V_{BV}(t)$ be the set of BV nodes at time t; the removal of $V_{BV}(t)$ and their incident edges from G will create a set of disjoint connected subgraphs $G_0(t), G_1(t), ..., G_{k-1}(t)$; let $G_0(t)$ be the subgraph containing the homebase; due to monotonicity, $G_0(t)$ is the explored portion of the graph at this time. If $k > 1$, we

call this condition one of *dynamic separation*. Then, with a reasoning similar to that of the proof of Theorem 31, we have the following.

Theorem 32. *The number spread(P, G) of casualties of protocol P in G is at least*

$$spread(P, G) = spread(P, G, t) + |V_{BV}(t)| + \sum_{0<i<k-1} |V_i(t)|$$

where $V_i(t)$ is the set of vertices of $G_i(t)$, and spread(P, G, t) is the number of casualties incurred by P up to time t.

Notice that agents might not be able to detect if a condition of dynamic separation exists at a given time because they do not know the location of the still unexplored original BVs. On the other hand, they can detect if a dynamic separation has been caused by the clones.

As mentioned, separation at any time (initial or dynamic) has a negative effect on the number of casualties because all the non-BV nodes inside those components will become infected.

Another important observation is that, even if there is no initial separation and the execution might not cause dynamic separations, there are networks where no protocol can avoid $O(n)$ casualties, even if the number of BVs is small.

Consider the graph R where the homebase is connected to a ring of $2n_{BV}$ nodes, each connected to a (possibly distinct) node of a connected graph G' of $n-(2n_{BV}+1)$ nodes; see Figure 7.6 where $n_{BV} = 5$. The sample graph shows five appropriately placed BVs can contaminate all nodes but h.

Theorem 33. *spread$(R) = n - 1$*

Proof. Consider any monotone sequential decontamination protocol P, and consider the following game between P and an adversary that will choose n_{BV} out of the $2n_{BV}$

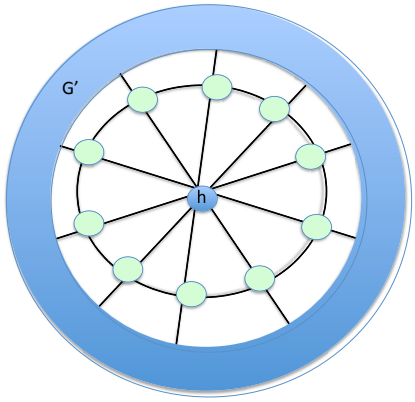

Figure 7.6: Sample graph with $O(n)$ spread

ring nodes to be BVs. Let us call "free" a ring node where a decision has not been made yet by the adversary; initially all ring nodes are free. When the protocol chooses as a target a ring node z, if z is free, the adversary will designate it as a BV node; then both is neighbours on the ring are no longer free and become "contaminated". If z is "contaminated", it will also contaminate its two ring neighbours. Notice that in this process the adversary will designate as BV nodes at most n_{BV} ring nodes. It is easy to see that, whatever is the decontamination sequence of P, any selected target is a BV, either because so designated by the adversary, or because contaminated by a previous target. □

Let us make a final important observation. Consider the two adverse situations discussed above: (initial or dynamic) separation, and the situation expressed by Theorem 33; in order to occur, they require that the the impact zones of BVs overlap. In

fact, for separation to occur, a circle (or a chain) of BVs must surround each component, and on this circle/chain each BV at distance 1 from the next. Similarly, in the proof of Theorem 33, the allocation of the BVs that yields the catastrophic bound is along the ring; that is, each BV is at distance at most 2 from the next.

Indeed, in any graph G, if the distances between the BVs at a given time are greater than 2, these adverse conditions do not occur at that time. This means that *Locality of Casualty*, which we discussed in Section 7.1.1, can have a positive effect in the case of fertile clones. This is particularly interesting because, as observed in Section 7.1.1, it instead has a negative effect in the case of sterile clones.

7.2.2 Decontamination Protocol

We now present a decontamination protocol in an arbitrary graph with multiple BVs whose clones are fertile. The number and location of the BVs is unknown, and so is the topology of the graph. We call the BVs which exist in the system before BVD process starts, as *original BVs*; and we call the BVs, which are generated by triggering the original BVs during BVD process, as *exposed BVs*. The reason we name the newly generated ones as exposed BVs is that their locations is uncovered (i.e., exposed) by agents' exploration.

The general strategy of our protocol MBVD-Fertile, shown in Figure 7.7, is to sequentially explore the network from the home base, using shadows to ensure monotonicity, and keeping track of the exposed BVs.

Based on the latest knowledge on the graph (i.e., which nodes have been explored, and which nodes are infected by BVs), the node to be explored next is chosen from the unexplored nodes of the fronteer, initially excluding the exposed BV nodes.

To minimize contamination, the target is selected as the candidate with the minimum number of neighbouring unexplored nodes that are not exposed BV; the reason

not to consider a neighbouring exposed BV (say node z) when computing the residual degree of a candidate x, is because if x is an original BV and is chosen as a target, its clone moving to z will merge with the BV already there; thus it will not add to the number of infected nodes; in other words, x can be explored without worrying about infecting node z because z has been infected already.

Observe that, in this selection process, the target is never an exposed BV; it could however be an original BV. If this is the case, its unexplored neighbours become exposed BVs and are are recorded properly. After each new exploration, the agents update the residual degrees of all unexplored nodes including the exposed BVs; again, when computing the residual degree of a candidate its neighbouring exposed BVs are not considered.

Once the update of the residual degrees completes, the agents check if there is/are exposed BV node(s) whose residual degree reach(es) zero. If yes, the agents first clean these exposed BV nodes sequentially, shadowing their explored neighbours. Note that to eliminate one zero-residual-degree (zero-RD) exposed node does not require any surrounding, and removes that node without creating any new BV node (procedure CLEAN_0-DEGREE_EXPOSED_BV). After all zero-RD exposed nodes are cleaned, a new target is then selected and the exploration process continues

The only deviance from this process occurs if the agents determine from the current map that they can surround and clean some exposed BVs safely, i.e., without contaminating any non-BV node. In this case, the agents perform the cleaning (Procedure CLEAN_SAFE_BV) and then resume the exploration process.

The exploration process terminates when either there are no more unexplored nodes or the fronteer is composed solely of exposed BVs, each of residual degree greater than zero. In the first case, the decontamination process is terminated. In the second case, one more step, called *Secondary Step*, is needed. Let $V_{BV}(t)$ be the

set of exposed BV nodes at this time; the removal of $V_{BV}(t)$ and their incident edges from G will create a set of disjoint connected subgraphs $G_0(t), G_1(t), ..., G_{k-1}(t)$ where the subgraph $G_0(t) = G_{ex}$ is the currently explored graph and contains the homebase, while all other $G_i(t)$ are still unexplored.

We know, by Theorem 32 that all nodes of those unexplored G_i will become necessarily contaminated, regardless of what protocol is now used. Thus, what we do now is just simply proceed to sequentially explore the unexplored nodes, protecting with shadows their explored neighbours. The agents clean up all regions completely separated/circled by exposed BVs, causing all unexplored nodes to become contaminated, then cleaned up. This process continues until there are no more unexplored nodes.

The proposed algorithm, MBV Decontamination-Fertile, is shown in Figure 7.7; the pseudo-code of procedures CLEAN_0-DEGREE_EXPOSED_BV and CLEAN_SAFE_BV is straightforward and not shown.

7.2.3 Costs of Protocol

The actual cost of the proposed protocol MBV Decontamination - Fertile, MF for short, depends on many factors; in particular, it depends on (1) the topology of the network and the fact that it is unknown and chosen by an adversary; and (2) the location and number of BV, unknown to the agents and under the choice of an adversary. We can however establish some general bounds on the number of casualties.

Theorem 34. *Let the decontamination of G by* MF *do not require executing* Secondary Step. *Then*

$$spread(MF, G) \leq n_{BV} \cdot \Delta(G)$$

MBV Decontamination - Fertile

(* Initialization *)
All agents initially at home-base h.
$M = (V_M, E_M) := N^2(h)$; (* initial Map is the 2-neighborhood of home-base *)
$V_{ex} := \{h\}$; (* only the home base is explored *)
$V_{un} := V+_{un}; = V_M \setminus \{h\}$; (* initial unexplored nodes *)
$V_{un}^- := \emptyset$; (* initial unexplored nodes containing an exposed BV *)
$Fr := N(h)$; (* unexplored frontier *)
$Finished :=$ **false;**

(* Iteration *)
while $V_{un}^+ \neq \emptyset$ (* *Primary Step* *)
    $Fr^+ := Fr \setminus V_{un}^-$;
    forall $v \in Fr$ **do**
        $r(v) := |\{u \in V_{un}^+ : (u, v) \in E_M\}|$; (* residual degree of frontier *)
    Choose $v \in Fr^+$ such that $r(v)$ is minimum; (* selection of target v *)
    $N_{ex}(v) := \{u \in V_{ex} : (u, v) \in E_M\}$; (* explored neighbours of v *)
    Position a shadow agent at each $u \in N_{ex}(v)$;
    Move an exploring agent to v;
    $M := M \cup N^2(v)$; (* update map*)
    $V_{ex} := V_{ex} \cup \{v\}$; (* update explored nodes*)
    $V_{un} := V_M \setminus V_{ex}$; (* update unexplored nodes*)
    $Fr := \{x \in V_{un} : \exists y \in V_{ex}, (x, y) \in E_M\}$; (* update frontier*)
    if (v was a BV) **then**
        $V_{un}^- := (V_{un}^- \setminus \{v\}) \cup (N(v) \setminus V_{ex})$; (* update exposed BV list *)
    CLEAN_0-DEGREE_EXPOSED_BV;
    CLEAN_SAFE_BV;
    $V_{un}^+ := V_{un} \setminus V_{un}^-$;
endwhile
while $V_{un} \neq \emptyset$ (* *Secondary Step* *)
    Choose $v \in Fr$; (* selection of target v *)
    $N_{ex}(v) := \{u \in V_{ex} : (u, v) \in E_M\}$; (* explored neighbours of v *)
    Position a shadow agent at each $u \in N_{ex}(v)$;
    Move an exploring agent to v;
    $M := M \cup N^2(v)$; (* update map*)
    $V_{ex} := V_{ex} \cup \{v\}$; (* update explored nodes*)
    $V_{un} := V_M \setminus V_{ex}$; (* update unexplored nodes*)
    $Fr := \{x \in V_{un} : \exists y \in V_{ex}, (x, y) \in E_M\}$; (* update frontier*)
endwhile

Figure 7.7: MBV Decontamination - Fertile

Proof. If the *Secondary Step* is not executed, then all the exposed clones are cleaned in the *Primary Step*; this is done by the procedures CLEAN_0-DEGREE_EXPOSED_BV and CLEAN_SAFE_BV, without contaminating non-BV nodes. In other words, only the original BVs have contaminated non-BV nodes; since a BV can only contaminate its neighbours, then each BV will contribute at most $\Delta(G) - 1$ casualties needed to clean the non-BV nodes contaminated by its clones, plus one casualty for itself; thus the claim follows. □

In other words, if the *Secondary Step* is not required, the number of casualties is quite limited; indeed, it is asymptotically optimal in bounded-degree graphs.

Theorem 35. *Let the decontamination of G by* MF *require executing* Secondary Step*, starting at time t. Then*

$$spread(MF, G) \leq n_{BV}(t) \cdot \Delta(G_{ex}(t)) + |V_{un}(t)|$$

where $G_{ex}(t)$ is the explored graph at time t, $V_{un}(t)$ are the still unexplored nodes at time t, and $n_{BV}(t)$ is the number of original BVs in $G_{ex}(t)$

Proof. Let $spread(MF, G, t)$ be the number of casualties incurred by MF up to time t. By Theorem 32, $spread(MF, G) = spread(MF, G, t) + |V_{un}(t)|$. But, by Theorem 34, $spread(MF, G, t) = spread(MF, G_{ex}(t)) \leq n_{BV}(t) \cdot \Delta(G_{ex}(t))$, and the claim follows. □

In case the *Secondary Step* is required, Theorem 35 states that the number of casualties depends on the size of the still unexplored area $V_{un}(t)$ at that time, which could be very high. However, the total number of casualties can be very high not just for the proposed protocol. Indeed, the adversary can always force *every* decontamination protocol to incur $O(n)$ casualties, even with a small number of BVs; see for

example Theorem 33.

7.3 MBVD in Special Graphs

The special graphs we studied in Chapter 5 in presence of a single BV are Meshes, Tori, and Hypercubes. We have seen that the fact of whether the BV clones were fertile or sterile made no difference for the exploration. In presence of multiple BVs, the situation is quite different whether the BV triggers sterile or fertile clones.

With multiple BVs with *sterile* clones, in Meshes, Tori or Hypercubes, the agents calculate the exploring sequence as in a single-BV system (e.g., Protocols BVD-qT, BVD-qH and ... described in Chapter 5) and proceed on the predetermined exploring sequences. Whenever encountering a BV node, the agents follow the general strategy for multiple sterile BV systems described in Section 7.1 where, instead of choosing greedily the next node to explore, the agents select the "next" in the predefined exploration sequences, calculated as in Chapter 5. In fact, of the three modifications indicated in the general case only the first two are needed: 1) continuing the exploration until all the graph has been visited, keeping track of the clones' positions; 2) use doubly cautious walk when moving on a node that contains a clone. The third one (update of residual degrees) is not necessary because the exploring sequence is predefined and depends solely on the topology.

In the case of *fertile* clones, the algorithm follows the lines of the one for general graphs described in Section 7.2. As for general graphs, there is the unavoidable risk that the BVs and their clones disconnect the explored portion of the graph from the unexplored one. We also modify the protocol under certain conditions on the distances between the BVs. In fact, when the BVs are guaranteed to be "far enough", the agents are made to safely surround and clean the clones of a BV as soon as they trigger it, and before proceeding in their exploration sequence.

7.3.1 Sterile Clones in Grids, Tori, and Hypercubes

MBVD in qD Grid

Let us first consider the simpler case of 2D grids, and let us to refer to Figure[1] 7.8. Protocol BVD-2G follows the route by going along a snake-like path starting from the border. In multiple-BV 2D Grids, there are n_{BV} BVs. Protocol MBVD-2G follows the same exploration sequence in the same way as in one-BV 2D Grid. Whenever encountering a BV node, the agents mark the new BV nodes, and proceed on the same exploration path moving to one of the new clone BVs and cleaning it. As described in Section 7.1, when moving to a node containing a clone, the agents perform a double cautious walk to take into account the possibility that the node contains a BV.

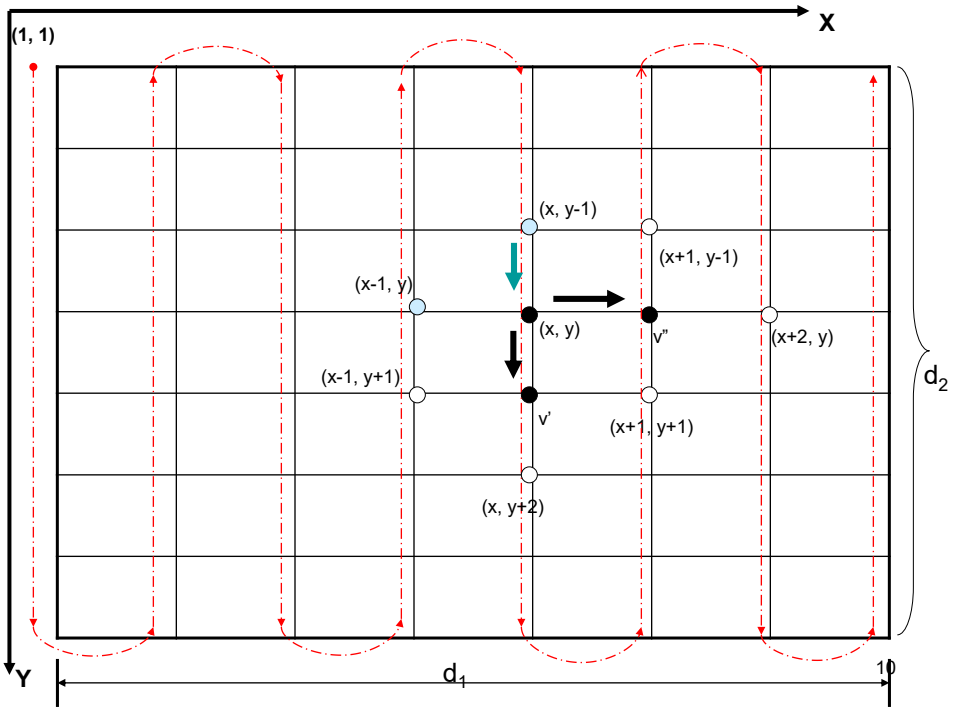

Figure 7.8: Exploration in 2D

Before proceeding, we remind some notation. Let $v = (x, y)$ be the node under

[1]Notice that this is the same as Figure 5.1, reprinted here for the convenience of the reader.

exploration, with $1 \leq x \leq d_1$ and $1 \leq y \leq d_2$. Let x^+ and x^- be defined as follows: $x^+ = x + 1$ (if $x \neq d_1$), $x^+ = x$ otherwise. Similarly: $x^- = x - 1$ (if $x \neq 1$), $x^- = x$ otherwise. Analogous definition holds for y^+ and y^-. The sets of explored and unexplored neighbors of node $v = (x, y)$ are defined as in Chapter 5:

$$N_{ex}(v) = \begin{cases} \{(x^-, y), (x, y^-)\} & \text{if } x \text{ is odd} \\ \{(x^-, y), (x, y^+)\} & \text{if } x \text{ is even} \end{cases}$$

$$N_{un}(v) = \begin{cases} \{(x^+, y), (x, y^+)\} & \text{if } x \text{ is odd} \\ \{(x^+, y), (x, y^-)\} & \text{if } x \text{ is even} \end{cases}$$

The actions performed in the shadowed exploration are as follows:

ALGORITHM *MBVD-2G*: SHADOW EXPLORATION

Agents at node $(x, y - 1)$.

1. Agents compute $Next(x, y)$:

   If x is odd and $x \neq d_1$, $y \neq d_2$: $Next(x, y) = (x, y^+)$

   If x is even and $x \neq d_1$, $y \neq 1$: $Next(x, y) = (x, y^-)$

   If x is odd and $y = d_2$: $Next(x, y) = (x^+, y)$

   If x is even and $y = 1$: $Next(x, y) = (x^+, y)$

2. Shadow agents move to occupy the nodes $N_{ex}(Next(x, y)) \setminus (x, y)$

3.If $Next(x, y)$ contains a clone:

   move an exploring agent followed by a verifying agent to $Next(x, y)$.

   otherwise: move an exploring agent to $Next(x, y)$.

4. If unarmed, the agent(s) returns to (x, y) and move to $Next(x, y)$.

As in the general case, each BV will cause a number of additional casualties which is at most the minimal residual degree of the graph, which in this case, is always 2. So, we have:

Theorem 36. *Let G be a 2D grid contaminated by n_{BV} black viruses which produce sterile clones. Decontamination of G starting from h can be performed with at most $spread(G) \leq 3n_{BV}$ casualties.*

The case of q-dimensional Grids is similar, and analogously we have:

Theorem 37. *Let G be a qD grid contaminated by n_{BV} black viruses which produce sterile clones. Decontamination of G can be performed with at most $spread(G) \leq (q+1)n_{BV}$ casualties.*

MBVD in qD Tori and q Hypercubes

The case of the multi dimensional tori and hypercubes are treated in the same way. We use the original algorithms described in Chapter 5, while implementing the modification described in this Chapter for the general graph (Section 7.1), thus obtaining:

Theorem 38. *Let G be a qD tori contaminated by n_{BV} black viruses which produce sterile clones. Decontamination of G can be performed with at most $spread(G) \leq 2qn_{BV}$ casualties.*

Theorem 39. *Let G be a qD Hypercubes contaminated by n_{BV} black viruses which produce sterile clones. Decontamination of G can be performed with at most $spread(G) \leq qn_{BV}$ casualties.*

7.3.2 Fertile Clones in Grids, Tori, and Hypercubes

As in the case of arbitrary graphs, with Fertile clones we have to face the problem of disconnection. In this case, the number of casualties could highly exceed Δn_{BV} and this could very well happen also in these specific topologies. The disconnection could either occur because the removal of the BVs generate disconnected components, or could happen during the execution of the algorithm even if this is not the case.

In these special topology, however, we can guarantee that disconnection does not happen slightly modifying the general algorithm, under some conditions on the distance between the BVs. Let us consider the case when the impacting zone of the BVs does not overlap, and actually their distance is greater than some *safe distance* *sd*, which will depend on the topology. Consider the following variation of the general Algorithm MBV Decontamination - Fertile described in Figure 7.7:

- when triggering a BV at node v, the agents immediately perform a surrounding of each newly exposed BV in $N_{un}(v)$ and they clean it, before proceeding with the exploration.

If any node in $N_{un}(v)$ is within distance *sd* from v, the surrounding is guaranteed not to encounter any BV and the agents will be able to safely clean all the clones. Note that nodes where the clone reside do not contain an original BV so the cleaning cannot trigger a new spread.

It turns out that the safe distance between BVs is 4 in each of these classes of topologies.

Lemma 8. *Consider q- dimensional meshes, tori and hypercubes. If the BVs are at distance* ≥ 4 *from each other, an agent can always reach the neighbours of a node containing a clone moving through "safe" nodes.*

Proof. Let the agents explore a node u containing a BV moving there from a neighbouring node v. Let u be denoted by q values corresponding to its dimensions $u = i_1, i_2, \ldots i_q$. Without loss of generality, let node v differ in dimension i_j. so $v = i_1, i_2, \ldots, \overline{i_j}, \ldots, i_q$. The nodes to be reached to surround the ones containing the clones triggered by u belong to $N^2(u)$, which means that they differ with u in at most two dimensions (and thus they do not contain BVs). Consider one of those nodes, say w differing from u in some dimensions i_k and i_h. An agent can move from u to w

by following dimension i_j (thus reaching v that is safe), then traversing dimensions i_k and i_h, finally taking dimension i_j again. In doing so no BV can be encountered because the path taken by the agent passes only through nodes at distance at most 3 from a BV. □

Theorem 40. *Let G be a q dimensional grid contaminated by n_{BV} black viruses which produce fertile clones. If the BVs are at distance ≥ 4 from each other, then decontamination of G can be performed with at most $spread(G) \leq 3n_{BV}$ casualties.*

Theorem 41. *Let G be a q dimensional torus or hypercube contaminated by n_{BV} black viruses which produce fertile clones. If the BVs are at distance ≥ 4 from each other, then decontamination of G can be performed with at most $spread(G) \leq 2qn_{BV}$ casualties.*

Theorem 42. *Let G be a q dimensional grid contaminated by n_{BV} black viruses which produce fertile clones. If the BVs are at distance ≥ 4 from each other, then decontamination of G starting from h can be performed with at most $spread(G) \leq qn_{BV}$ casualties.*

7.4 Summary

This chapter extends the investigation of the BVD problem to systems that contain multiple BVs. The challenges due to the existence of multiple BVs in graphs are discussed. In particular, the difference made by the BV producing sterile vs fertile clones has been highligthed.

In the case of *sterile* clones, a general strategy for arbitrary graphs is presented. The basic idea is to combine shadow exploration and cleaning phases together. Once triggering a BV, all exposed BVs are recorded, and all residual degrees of unexplored nodes (including exposed BV nodes) are updated. Agents continue to perform shadow

exploration based on minimum residual degree. Whenever an exposed BV's residual degree reaches zero, agents explore (i.e., clean) it. The complexities of MBVD in arbitrary graph are studied.

In the case of *fertile* clones, the situation is dramatically different in terms of casualties. In particular, the concepts of *separation* have been introduced. *Dynamic separation* is the situation that, during the execution of a decontamination protocol, the only edges between G_{ex} and G_{ux} are BVs. Notice that G_{ux} could be disconnected. If dynamic separation occurs, all nodes in G_{ux} will become contaminated, regardless of the decontamination protocol, and at least additional $|G_{ux}| + 1$ agents are required to complete BVD task. A general algorithm for arbitrary graphs is presented and analyzed.

After understanding the solution to the MBVD in arbitrary graph, the investigation continue to the MBVD in special graphs previously studied, i.e., regular special graphs (i.e., Tori and Hypercube) and irregular special graphs (i.e., 2D, 3D, and qD grids). The solution in these graphs are presented both for steriele and for fertile clones.

Chapter 8

Experimental Study on Black Virus Decontamination

In this chapter, we experimentally investigate the problem of black virus decontamination by studying the Greedy Exploration Algorithm (GREEDY) described in Chapter 6, and comparing it with Random Exploration (RANDOM).

Among the existing simulators for reactive distributed algorithms in network applications, DisJ combines many advantages and overcomes many shortcomings of existing simulators. We introduce DisJ by discussing its capabilities, graphic interface, modelling methods, execution and debugging, and APIs.

The basic solution protocol for decontaminating an arbitrary network through GREEDY EXPLORATION has been considered. Its behaviour, properties, and performance have been investigated through an extensive number of computer simulation runs. The simulation results not only confirm the existing theoretical results, but also disclose many interesting behaviour/properties of the solution protocol. In particular, they show that the examined protocol outperforms random search. The influence of graph connectivity density and size on complexities (movement, time, and agent size) is clearly depicted. The results of this chapter have been presented in [17].

8.1 Simulation Model

As described in Chapter 3, the environment in which the agents operate is a network whose topology is modeled as a simple arbitrary undirected connected graph. An agent is modeled as an entity with computational or information processing capability.

Agents follow the same protocol, but may have different functions/roles (i.e., states). Communication among agents (e.g., to coordinate, synchronize or update their maps, and etc.) occurs when they meet at the same node. Each agent has a unique id from some totally ordered set. There is a unique BV in the system.

Initially all agents are in *sleep* state; after initialization, an agent is activated as the first and unique system agent called Leader Exploration Agent (LEA). During different stages of operations, LEA can clone itself and change that cloned agent to an Exploration Agent (EA) and Shadowing Agent (SAs). At any time, there is only one LEA and EA. When the EA is killed by exploring or cleaning a BV, LEA will clone one or more EA or convert one SA into EA depending on whether the total number of SAs is sufficient to perform a specific task. Figure 8.1 shows the agents and BV state transition.

The states of a node include *unvisited*, *visited* (i.e., cleaned), *contaminated*, and *shadowed*. The node state diagram is shown in Figure 4.1. When an agent or a BV arrives a node, the interactions among agents or BV can be described in five possible situations depending on the state of the node and whether other agents or BV are located on the node.

8.2 Simulation of BVD

8.2.1 Simulation Platform, Distribution Java (DisJ)

The simulation software used in the work is called DisJ, which is a Java based software project implemented in Eclipse environment as a plug-in. The purpose of the software is to provide users with a generic event based simulation engine, i.e., a simulation environment with basic utilities (nodes, links, events, timer, etc.), so to allows the users to focus on developing their distributed algorithm protocols, which can then be

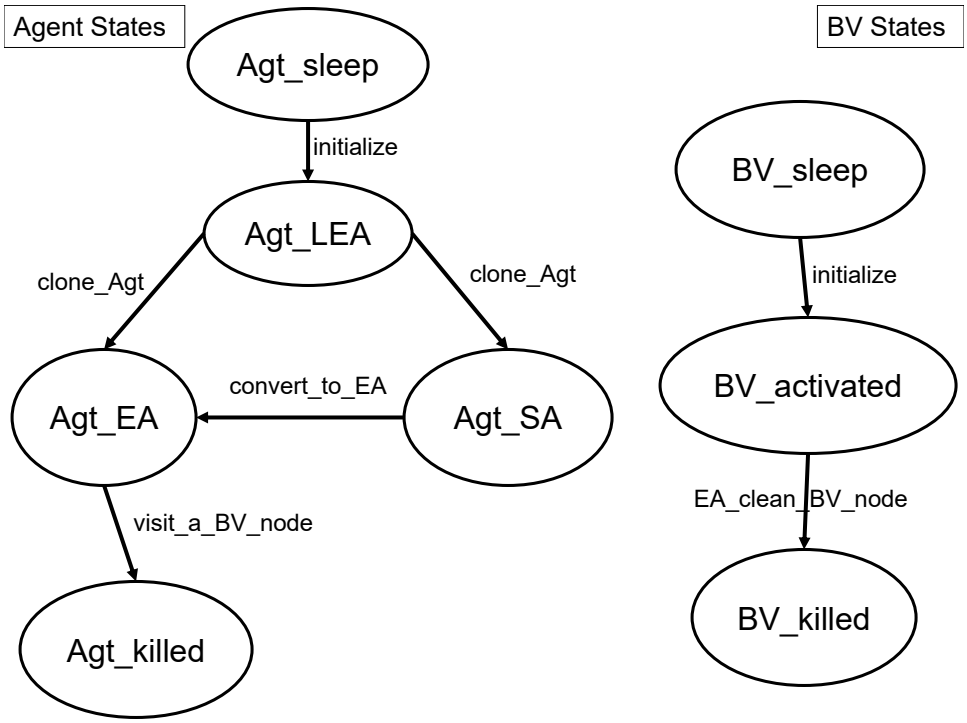

Figure 8.1: Agent and BV State Diagram

simulated in the engine with user defined network topologies. The results from the simulation software help users to understand the behavior of their protocols and to verify the correctness of the protocols.

The design of DisJ started from rigorous requirement specification inherited from its parent project [97]. The simulation engine per se is an event based simulation engine. The core of the simulation engine is driven by events put on an event heap. The simulation runs as long as there are events on the event heap. One of the main requirements is to decouple the users protocol developing activity from defining user network topology and executing the protocol. This means that the intended protocol and the network topology are developed, defined, and built separately from each other and separately from the simulation engine. This also means that a protocol can be run on different network topologies on the simulation engine. From the perspective of a support for writing a protocol, DisJ has basic functions:

- declare and set variables that the protocol will use (unique for each node in the topology);
- define the nodes that will be the initiator(s) (user defined, random, delayed initiator);
- define user specified delays (either bounded, random, or unique per link);
- define possibility of faults (either random or user controlled faults);
- define the type of faults that will occur (Byzantinus, crash, omission);
- perform local calculations;
- send messages to nodes in various ways (broadcast, reply, and along a labeled link);
- change the state of the currently executing node;
- specify a log file to which to log errors and tracing information;
- receive meaningful error messages when unexpected or erroneous events happen.

The simulation engine originally supported only messaging between nodes. To support mobile agents in the system, more functions have been added:

- inject agents on nodes;
- allow agents to move to neighboring nodes;
- support different communication mechanisms among agents on the same node via whiteboard, token, or message;
- allow agents to create new agents.

The simulation engine is able to calculate basic and relevant statistics pertaining to distributed algorithms. These statics include the number of messages/bits sent from a certain node or exchanged overall during the execution of a protocol, total agent movements, total time before termination, and etc. DisJ has strong debugging features which include: breakpoint rules (for example, stop when any node changes state), breakpoint with the mark, insert/enable/disable breakpoints, step (on events, or user defined events), exit at any time, watches on variables, states, and other statistics during simulation, restart, log execution to files, and replay based on records. DisJ allows users to change colors of nodes in different states, and to adjust the speed of simulation run to allow users to better view and follow the execution behavior.

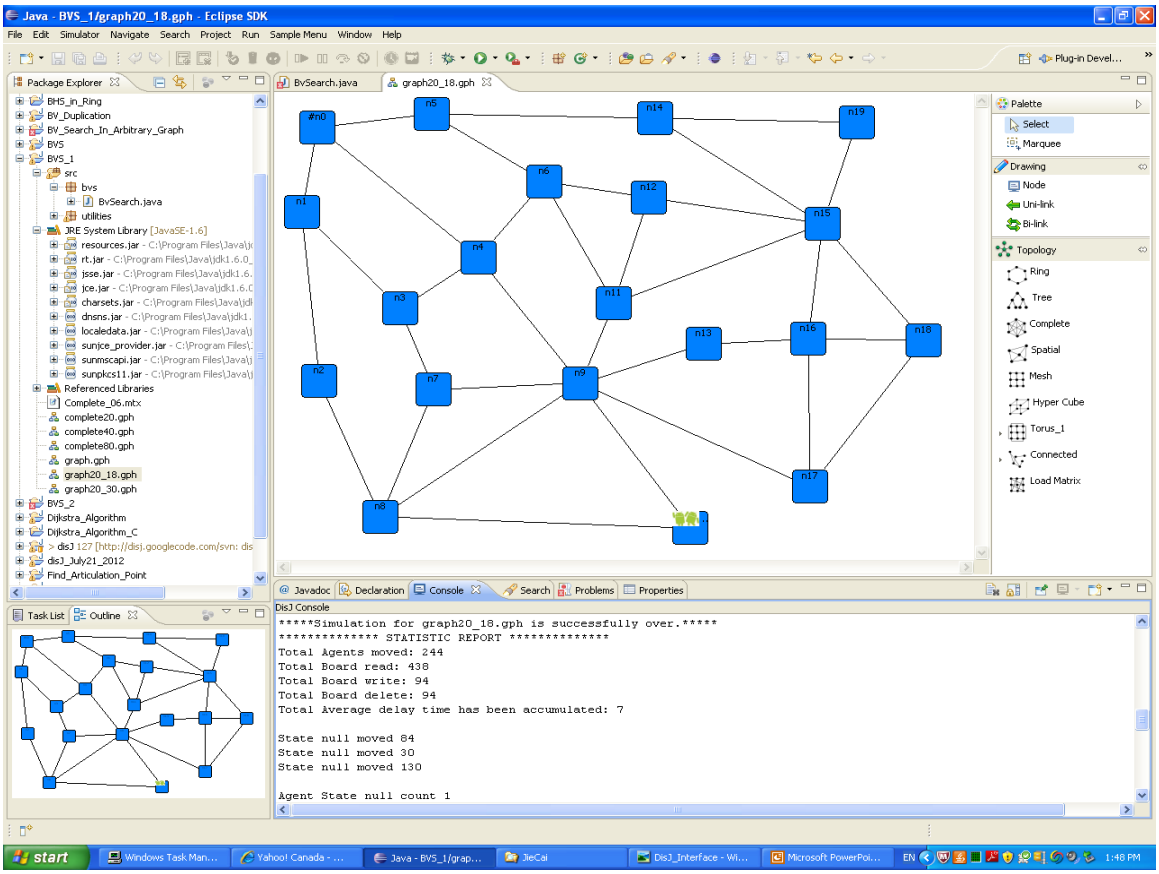

Figure 8.2: DisJ Interface

DisJ also provides nice GUI to allow the user to define their network topologies by automatic generating, drawing, or inputting network topology from user preprepared network matrix files, specify link properties (delay, loss, probabilities, and etc.), load, run and debug users protocols on simulation engine, and display final simulation statistic results. It also allows user to control simulation running speed to be able to carefully watch the execution process. Figure 8.2 shows a sample DisJ Interface.

The APIs for programming a protocol is small and simple. For example, following are essential APIs needed to implement a message passing distributed algorithm:

- public void *init()*: this method is invoked on a node if and only if the node property *Initiator* is set to be True;
- public void *alarmRing()*: this method is invoked when an alarm clock rings. The user can set an alarm clock to activate the event by invoking method *setAlarm(short)*;
- public void *receive()*: this method is invoked when a node receives a message;
- public void *send(), sendTo(), sendtToAll(), and sendToOthers()*: message sending interface family with different types of parameters argument;
- public int *getState()*: this method is for checking a current state of an entity;
- *public void become()*: sets a state of an entity to a given state;
- *public void moveTo()*: this method is invoked to allow an agent to move to a port;
- *public void moveToNode()*: this method is invoked to allow an agent to move to a neighbor node.

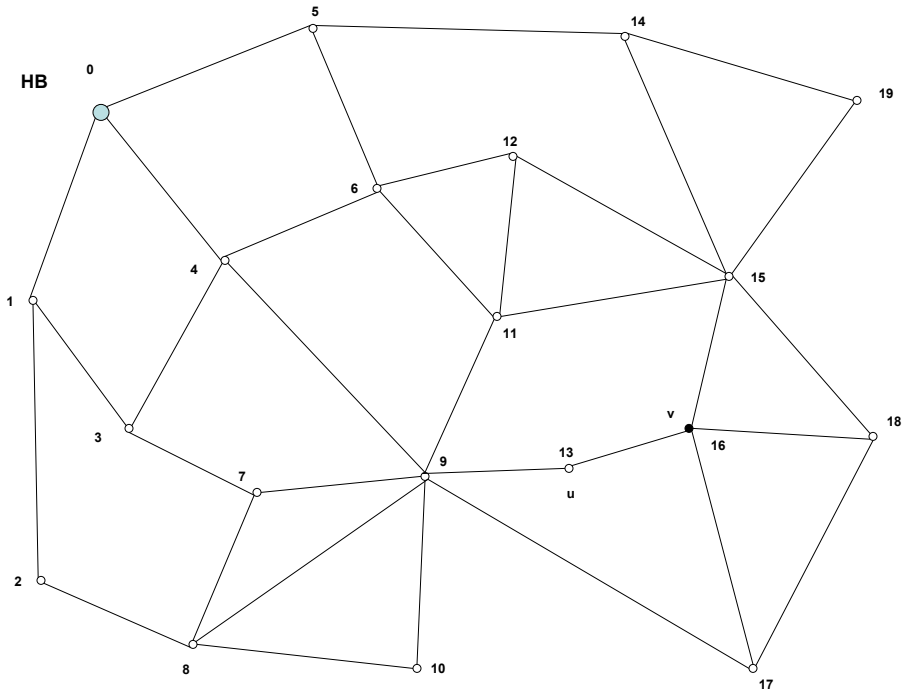

Figure 8.3: A sample graph20_18

Simulation Sample Graphs In order to obtain large amount of data, we prepared and ran simulations on 2255 graphs with different sizes and network connectivity densities. The sizes of the sample graphs are 20 ($graph20$), 40 ($graph40$), 60 ($graph60$), and 80 ($graph80$), and 100 ($graph100$) nodes. The network connectivity density is defined as the ratio of the number of links of a graph to the number of links in complete graph with the same node size, i.e., $\frac{2m}{n(n-1)}$. For each size of graphs, we consider 10 connectivity levels, i.e., 10%, 20%, 30%, ..., 80%, 90%, and 100%. A computer program is used to randomly generate 50 graphs for each size and each connectivity of graphs. Each graph is represented as $graph\#1\_\#2\_\#3$. #1 represents a graph size #2 indicates a connectivity density and #3 shows an instance of 50 graphs. The output from the program is a graph matrix file, which is the input for DisJ to generate and draw a graph for the simulation. The matrix file is also the input for the BV search protocol to generate the adjacency matrix. Two planar graphs are generated

manually, i.e., graph20_18 and graph40_11, with special arrangement of degrees for certain nodes for easy visualization of the behavior of different exploring protocols during simulations. Figure 8.3 shows a sample graph, graph20_18.

8.2.2 Simulation Results

We implemented the GREEDY EXPLORATION Algorithm described in Chapter 6, where the exploration sequence is obtained starting from an arbitrary homebase x_0, greedily selecting node x_{i+1} among the unexplored nodes connected to $x_0, \ldots x_i$ that minimize the residual degree.

We also implemented a RANDOM EXPLORATION Algorithm, where the exploration sequence is obtained starting from an arbitrary homebase x_0, ransoming selecting node x_{i+1} among the unexplored nodes connected to $x_0, \ldots x_i$.

We already know that the Greedy Exploration strategy produces the optimal solution in terms of spread. We are however interested in observing how far the solution is from the random exploration strategy depending on various topological factors, as well as to understand the impact of the choice of the homebase.

Comparison between GREEDY Exploration and RANDOM Exploration

In this Section we compare GREEDY with RANDOM to see how often GREEDY outperform RANDOM when focusing on minimizing the contamination spread.

As a worst case scenario, we actually let the exploration proceed until all nodes are explored without placing any black hole. When a node i is explored, its residual degree, d_{ri}, is recorded. After exploring all nodes of the graph, they are sorted into an array according to their residual degrees in descendant order. If the same d_{ri} in the array appears multiple times, we use a coefficient, i.e., c_i, to record this. The resultant array is called *Array of Residual Degree (ARD)*. Comparing two exploration

Figure 8.4: GREEDY_vs_RANDOM_G40_10

strategies becomes equivalently to lexicographically compare the two ARDs.

We already know that GREEDY is optimal, which means that the maximum d_{ri} for the GREEDY strategy is certainly smaller or equal than the one devised for the RANDOM strategy. What we do not know, however, is how the overall array of residual degree compare with the two strategies.

We do the comparisons of GREEDY and RANDOM for graphs with sizes of 20, 40, and 80 nodes. For each size category, we created 10 connectivity levels, so there are total 30 comparisons.

Simulation results demonstrate GREEDY is never worse than RANDOM for all test scenarios. As shown in Figures 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, there are two curves, one for

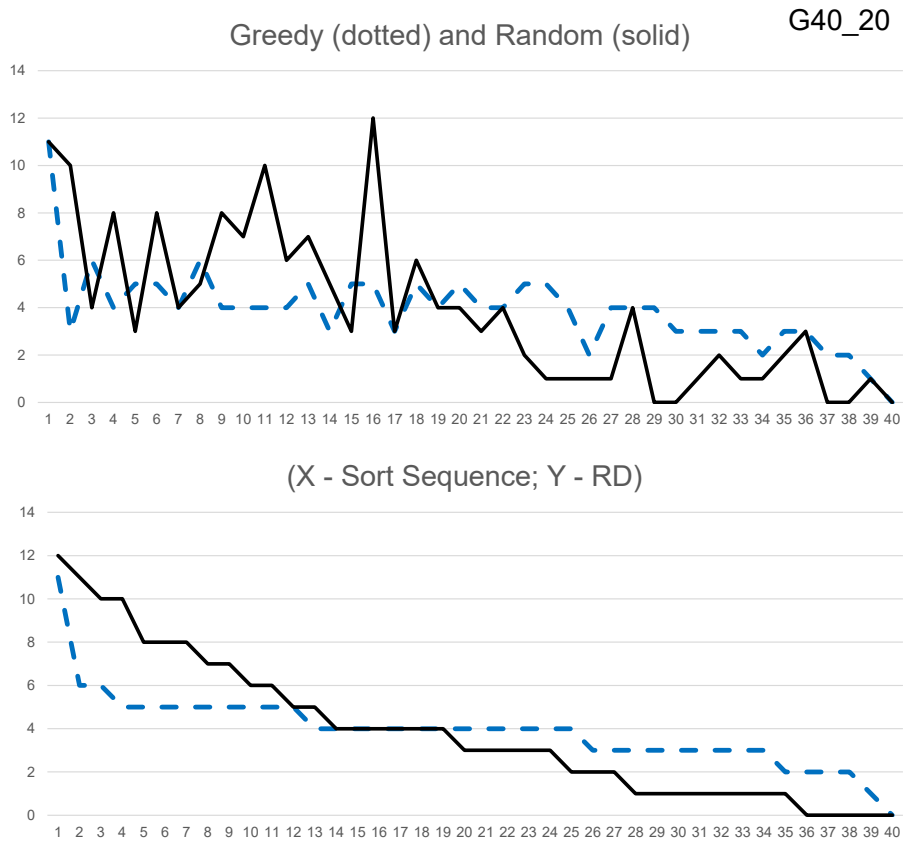

Figure 8.5: GREEDY_vs_RANDOM_G40_20

d_rs of each SS. The figures clearly show GREEDY is better than RANDOM for all graphs at each connectivity level. In RANDOM, d_r changes dramatically, while in GREEDY, d_r changes smoothly. It also demonstrated that, when the connectivity increases, the difference between GREEDY and RANDOM decreases. When connectivity approaches 100%, i.e., the complete graph, GREEDY and RANDOM are the same. This matches the hypothesis that in complete graph, every node have the same degree, so a random choice is effectively the same as a Greedy one.

Exploration Behavior and Properties

With the simulations, we observed some general properties of the algorithm. In particular, running the simulation on sample graph20_18, we observed following facts:

Figure 8.6: GREEDY_vs_RANDOM_G40_50

- *Sensitivity to the home-base location.* By checking the exploration behavior starting from different home bases, it is observed that changing home base has effect on the search sequences locally, but has no impacts on search sequence in far areas. The behavior is obvious when observing the exploration process starting from nodes 0 to node 8 in Figure 8.3. The observation demonstrates that exploration paths are exactly determined by residual degrees of nodes and relative distance from the starting node or the *current* node at which agents reside currently. This is a good property because it provides flexibility for users to decide where to start to explore the graph.

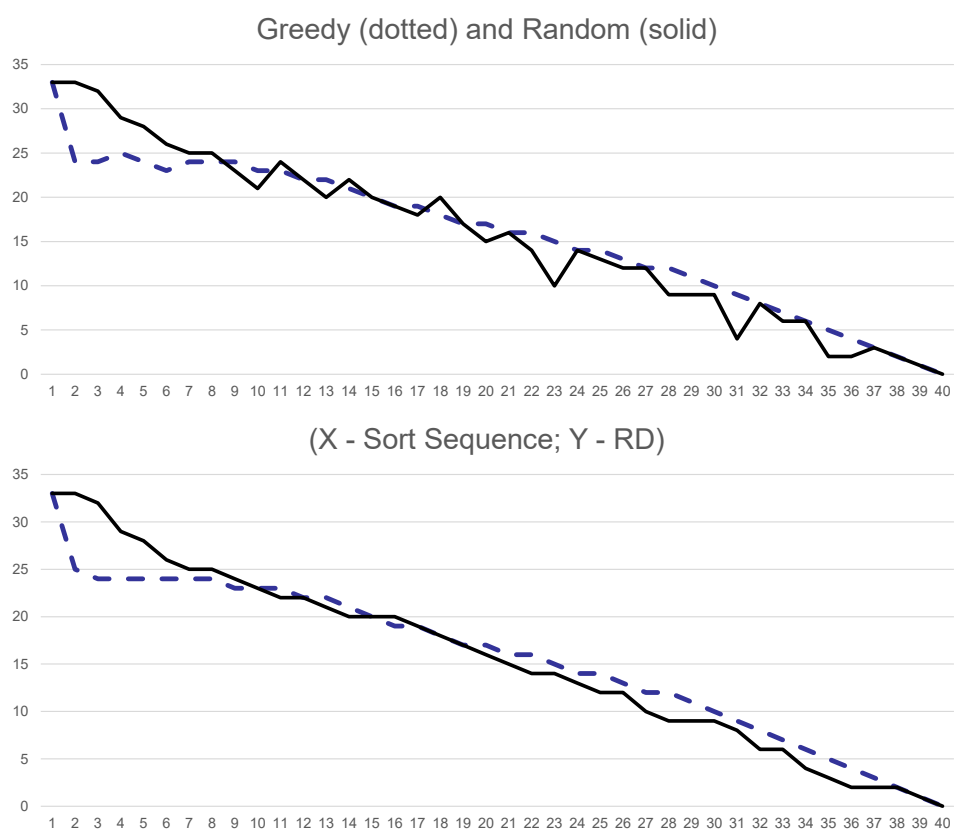

Figure 8.7: GREEDY_vs_RANDOM_G40_80

- *Influence of Graph Structure.* Next we observed that the exploration sequences, is influenced by the graph structures. By checking the search behavior starting from nodes 13, 16, 17, or 18 in Figure 8.3, it is observed that search sequences dramatically changes initially in contrast with the cases starting from node 0 to 8. However all of them terminate the exploration at node 10. This is because initially the search is restricted by nodes 9 and 15, which have high degrees and are the exits for agents to explore other areas of the graph. After breaking through the barrier at node 15 or 9, all search sequences follow the same/similar paths in the rest of the graph. This further demonstrates the first observation.

Figure 8.8: GREEDY_vs_RANDOM_G40_90

- *Home-bases with high degree.* The third observation is that starting from nodes with high degrees could reduce the *Size(G)* and the exploration cost. Let BH be at some node with a large degree, it is observed that the team *Size(G)* to perform the task and exploration cost could be smaller compared with cases where HB is located at other nodes. Taking HB at node 9 as an example. This node has the largest degree of 7 in this graph. Starting from this node means excluding it from BV suspicion. Therefore during exploration, agents do not need to go around this node to explore other nodes at first. Thus exploration cost could be reduced. SA's are reduced because it is the nodes with large degrees that require more SA's to shadow the $N_{ex}(v)$ before exploring node v.

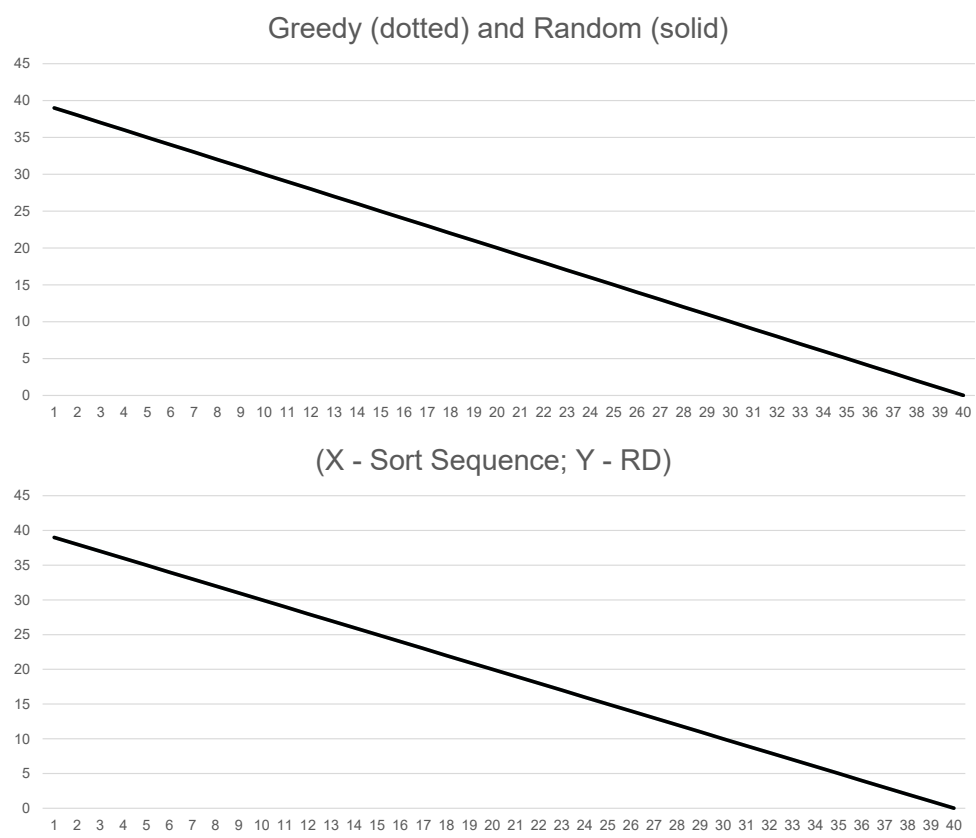

Figure 8.9: GREEDY_vs_RANDOM_G40_C

- *Worst case behaviour.* In the worst case complexity analysis, we mentioned that agents may move in one direction to explore one node, then move to opposite direction to explore a node at other end of a graph. In addition, agents may pass some nodes multiple times. We observed this behavior in the simulation run in graph20_18. When the agents start from node 0, we observe that the exploration first moves in one direction along nodes 1 and 2, follows an opposite direction alone nodes 2, 1, 3, 4, 0, 5, and 6, afterwards travels in opposite direction again alone nodes 4, 3, 7, 8, and 10, then changes direction again to move along nodes 8, 7, 3, 4, 6, 12, 11, 9. Exploring node 9 causes the worst case shadowing effort, 4 SA agents are sent to nodes 4, 7, 8, and 10. The search sequence starting

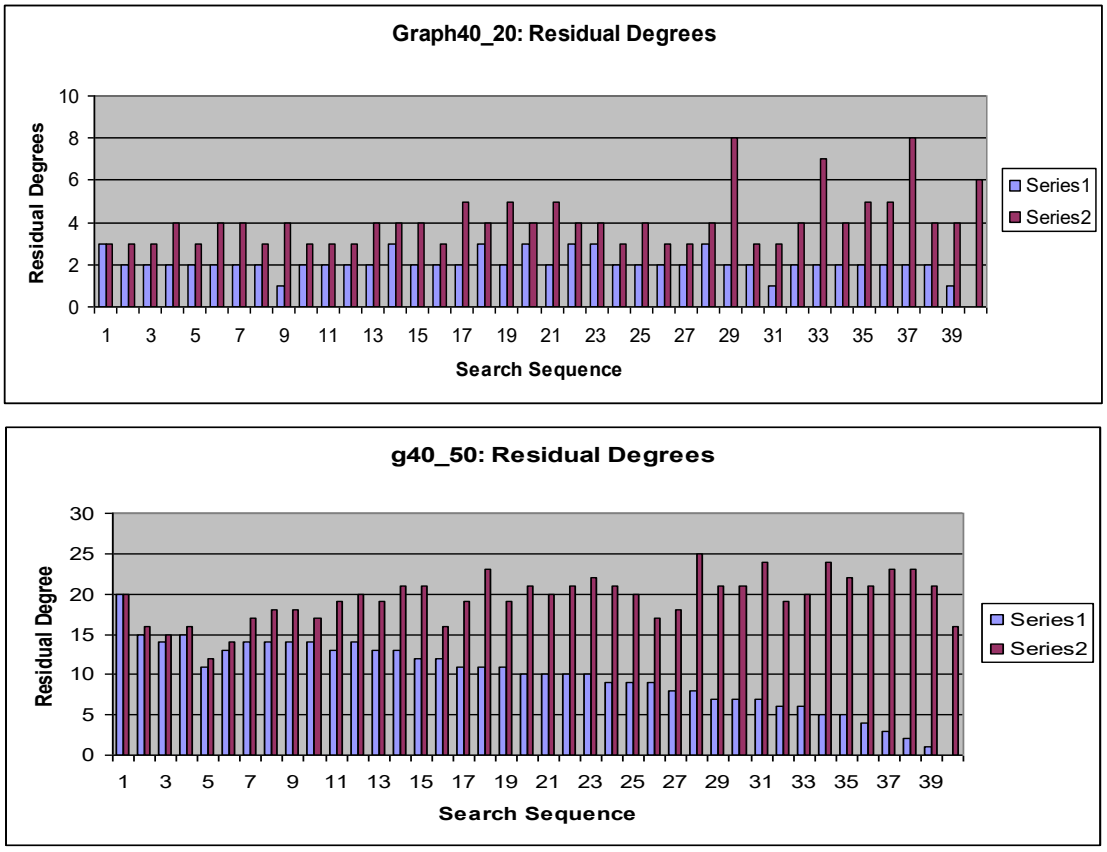

Figure 8.10: graph40_20 and graph40_50 search sequences

from node 0 does demonstrate the relatively worse situation that agents need to move back and forth, and to pass some nodes multiple times.

Degree vs. Residual degree

Figures 8.10 and 8.11 show search sequences, degrees and residual degrees of nodes for graph40_20%, graph40_50%, graph40_80%, and graph40_complete.

Note that, at the early stage of exploration, degrees and residual degrees do not differ too much. However, in a later stage of the exploration, their differences become larger and larger, as expected. Residual degrees gradually reduce and eventually reach zero. This is because the more nodes are explored, the more links are checked from 'the other ends", so the residual degrees of un-explored nodes decrease. Comparing

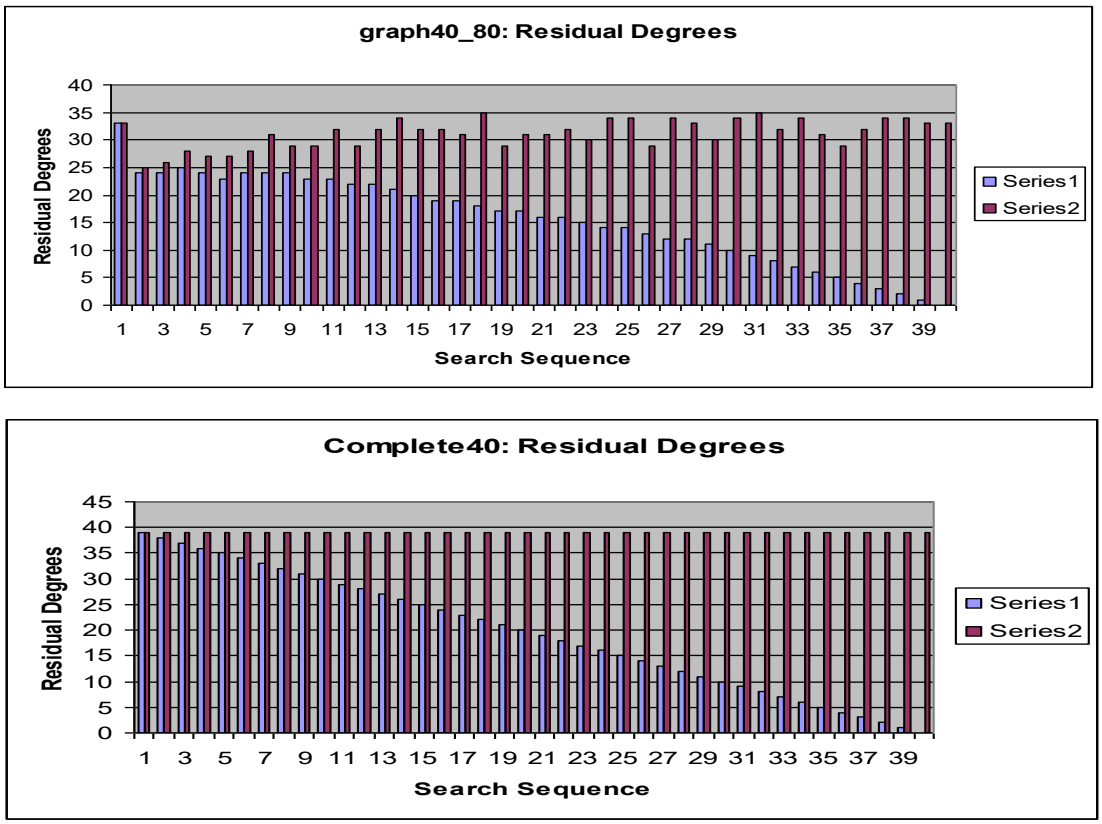

Figure 8.11: graph40_80 and graph40_Complete search sequences

the two strategies for a same graph, the exploration sequences are different. In both exploration strategies, the residual degrees reduces similarly. However in GREEDY, the residual degree decreases more smoothly. In both GREEDY and RANDOM, it is possible that a node could have a higher residual degrees compared with a node explored earlier. It could be that a node with high degree is explored first due to structure constraints. In other words, the residual degree curves in RANDOM show more abrupt jumps than those in GREEDY.

Statistics of Simulation Results

As we mentioned before, for each graph size at each connectivity level, 50 instances of graphs were generated, and simulations are run for each instance of the graph.

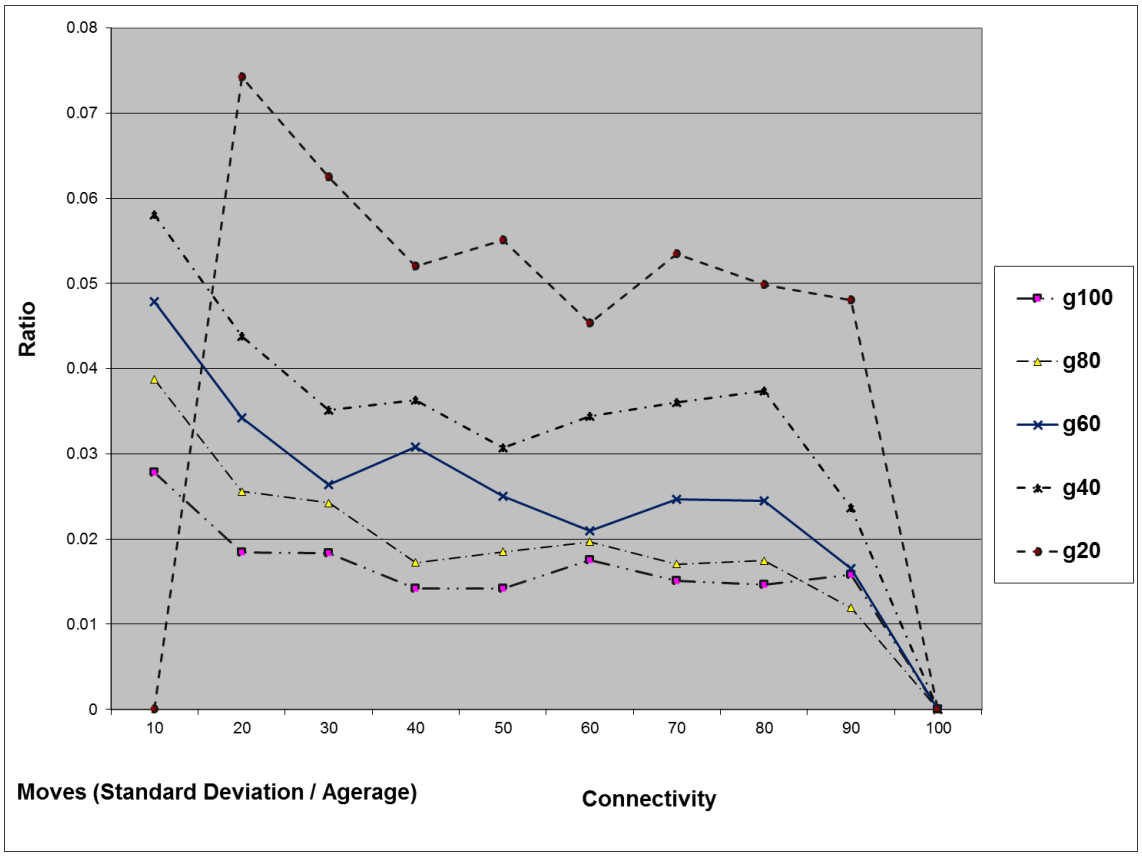

Figure 8.12: Movement SD Ratio vs Connectivity

After each simulation run, we collected movement, agent, and time complexities. Finally, we calculate averages and standard deviations (SD) for each graph size at each connectivity level. The ratio of standard deviation over average for movements, agents, and time are plotted in Figures 8.12, 8.13, and 8.14.

It is observed that in all cases, the standard deviations for movements, agents, and time complexities are all very small compared with the average estimations. From these three figures, we have some simple observations:

a. With regard to the graph sizes, graph20 has the largest ratio of standard deviation over average for movements, agents, and time. The is obviously true because

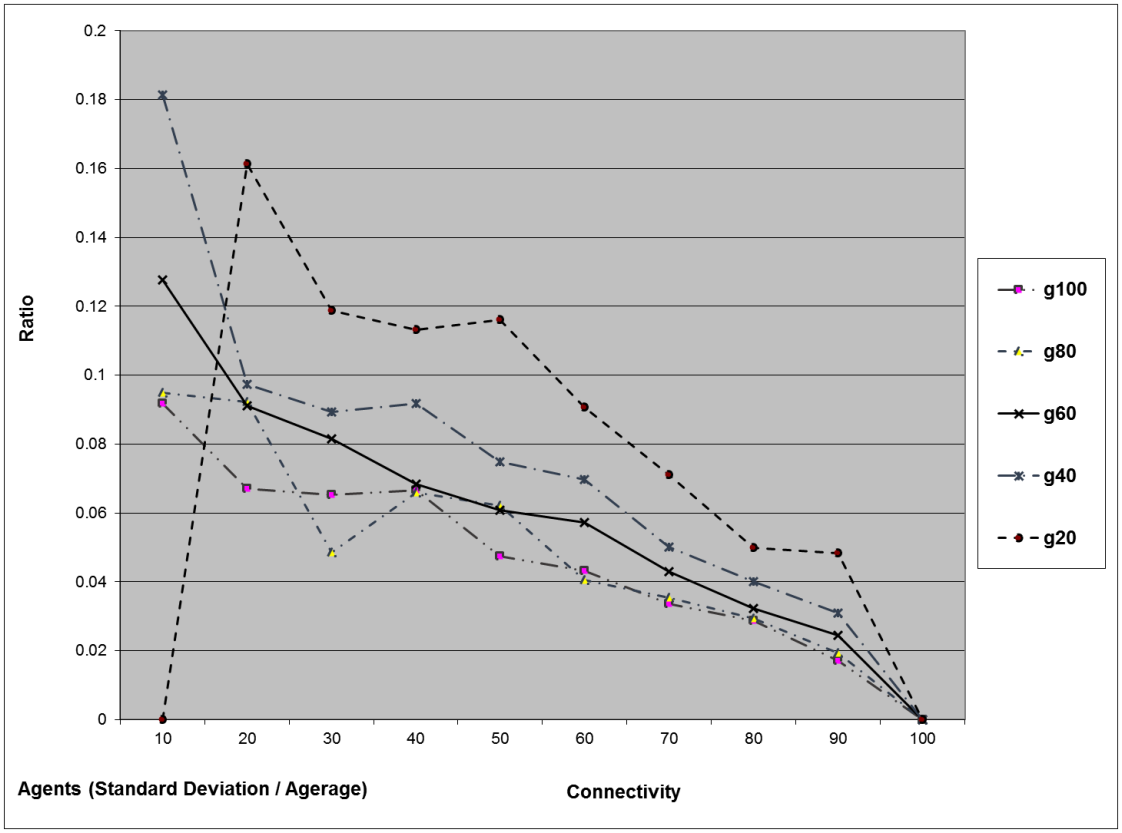

Figure 8.13: Agent SD Ratio vs Connectivity

small size graph does not generate as good statistical results as compared with large size graphs. Large graphs produce small ratio, i.e., better statistical results.

b. With regard to the connectivity levels, 10% connectivity level generates the largest ratio of standard deviation over average for movements, agents, and time. This is obviously true because at connectivity level 10%, the links of the graphs are very small (close to the link connectivity of ring or tree of the same size). The higher the connectivity levels, the smaller the ratio of standard deviation over average. At 100% connectivity level, the ratio reaches zero. In this case, all the nodes in complete graph are the same, so no mater where agents start and which path to take to explore

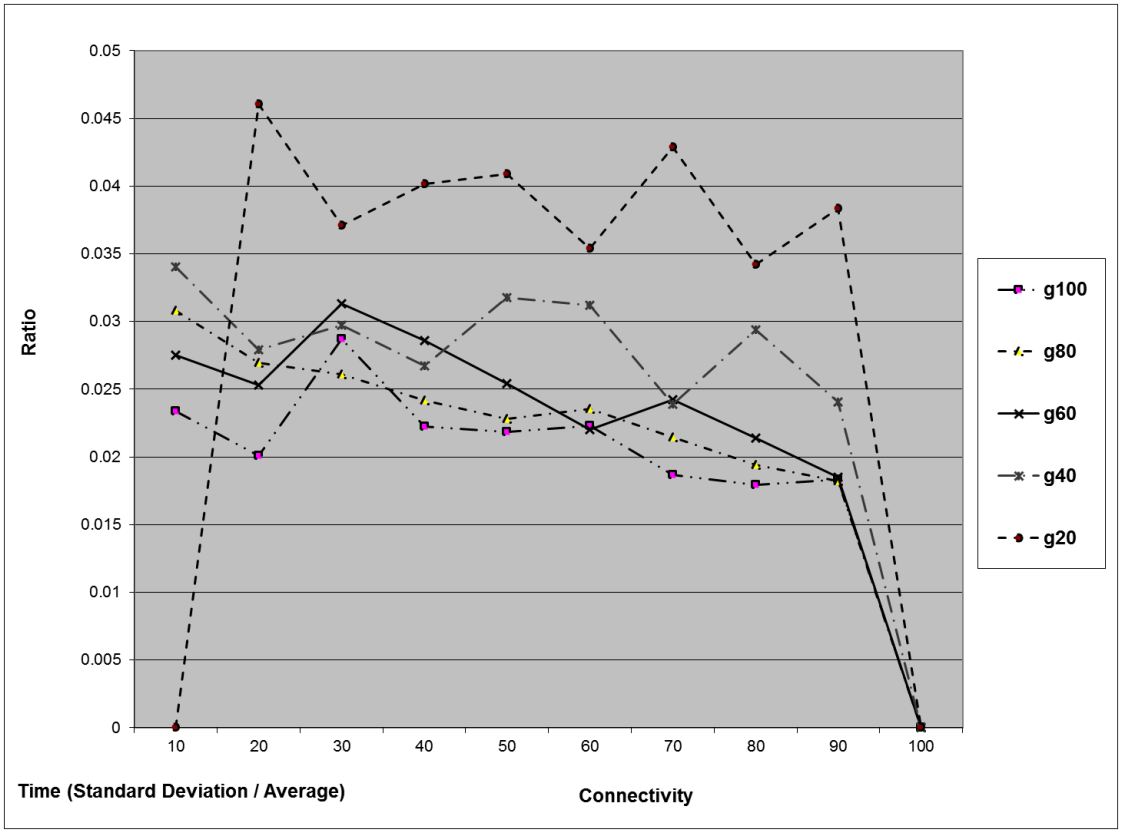

Figure 8.14: Time SD Ratio vs Connectivity

the graph, the simulation results are the same.

c. The ratios of standard deviation over average for graph20 at 10% connectivity are all zero (or very small), as can be seen in Figures 8.12, 8.13, and 8.14 . The reason is that for such a small graph with very low connectivity, the graphs generated are barely connected to meet the 2-connected condition, so the variations among these graphs are very small. Therefore the statistic results among these graphs deviate very little. Specifically, to meet the 2-connected condition, 20 links are required among 20 nodes. 10% connectivity among 20 nodes only needs 19 links, so there is no way to generate random graphs with this connectivity level.

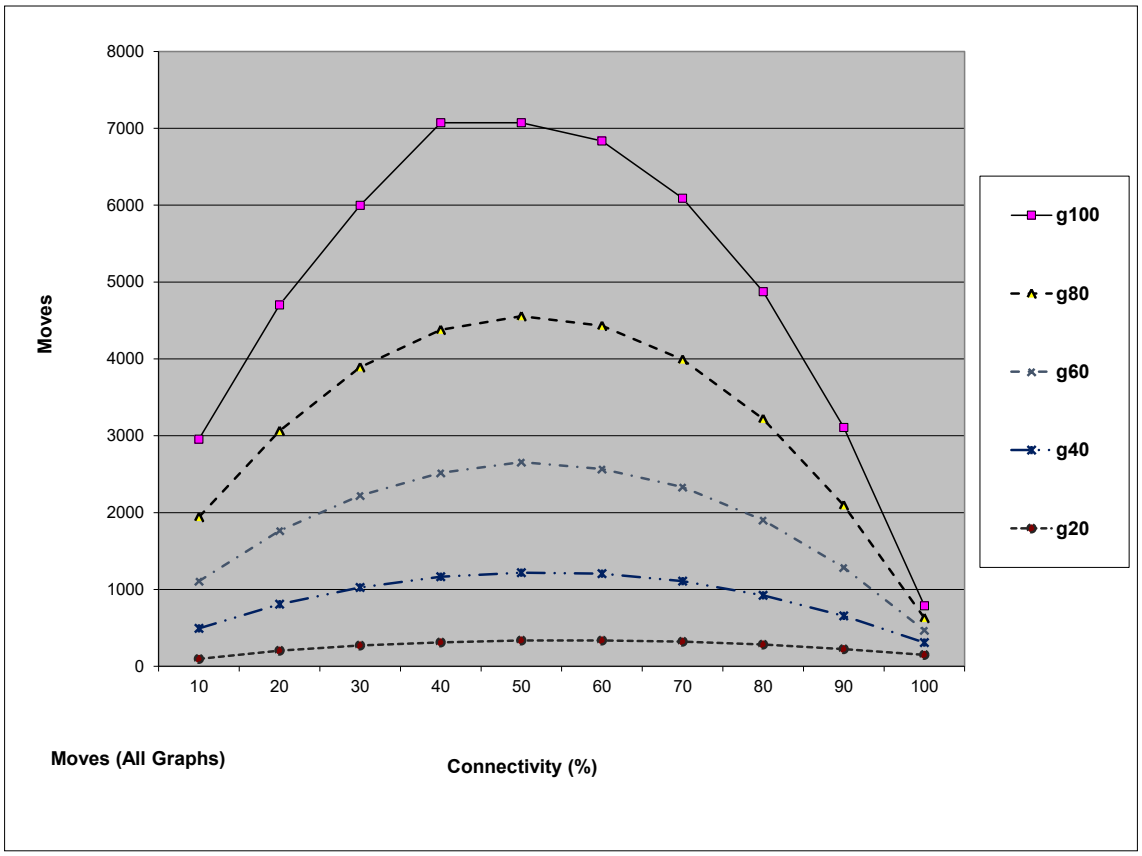

Figure 8.15: Movements vs Connectivity

Influence of connectivity density on complexity

For movement, agents, and time complexity in graph20s, graph40s, graph60s, graph80s, and graph100s, refer to the simulation results in Figure 8.15, 8.16, and 8.17.

For a given size of graph, with the increase of connectivity level, we have following observations:

a. Movement cost gradually increases to a maximum at 40%-60% connectivities, then it gradually decreases. It is interesting to note that movement costs are close for different graph sizes at 100% connectivity level. Given a graph size, the movement cost at 100% connectivity level is comparable or less than those at all or 10% connectivity levels. This demonstrates that graph density does not improve the movement

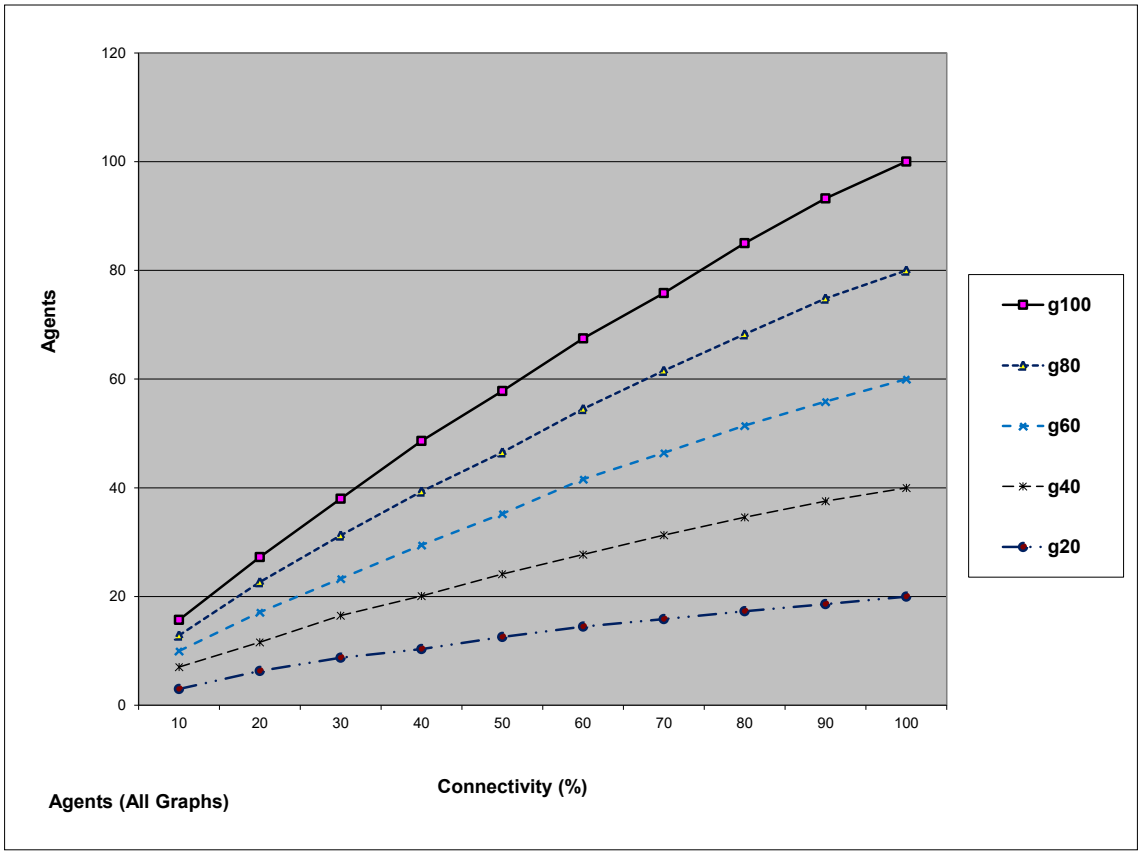

Figure 8.16: Agents vs Connectivity

cost. This also shows linear relationship be movement cost and graph size for complete graphs.

b. Simulation time cost shows gradually increase to maximum at middle connectivity levels, when it gradually decreases again. It is noted that time costs are close for different graph sizes at 100% connectivity level. Given a graph size, the time cost at 100% connectivity level is comparable or less than those at all or 10% connectivity levels. This demonstrates that graph density does not improve movement cost. This also shows linear relationship be time cost and graph size for complete graphs.

c. Team size continuously increases with the increase of connectivity. It reaches the maximum, i.e., graph size, at 100% connectivity level. Since high connectivity

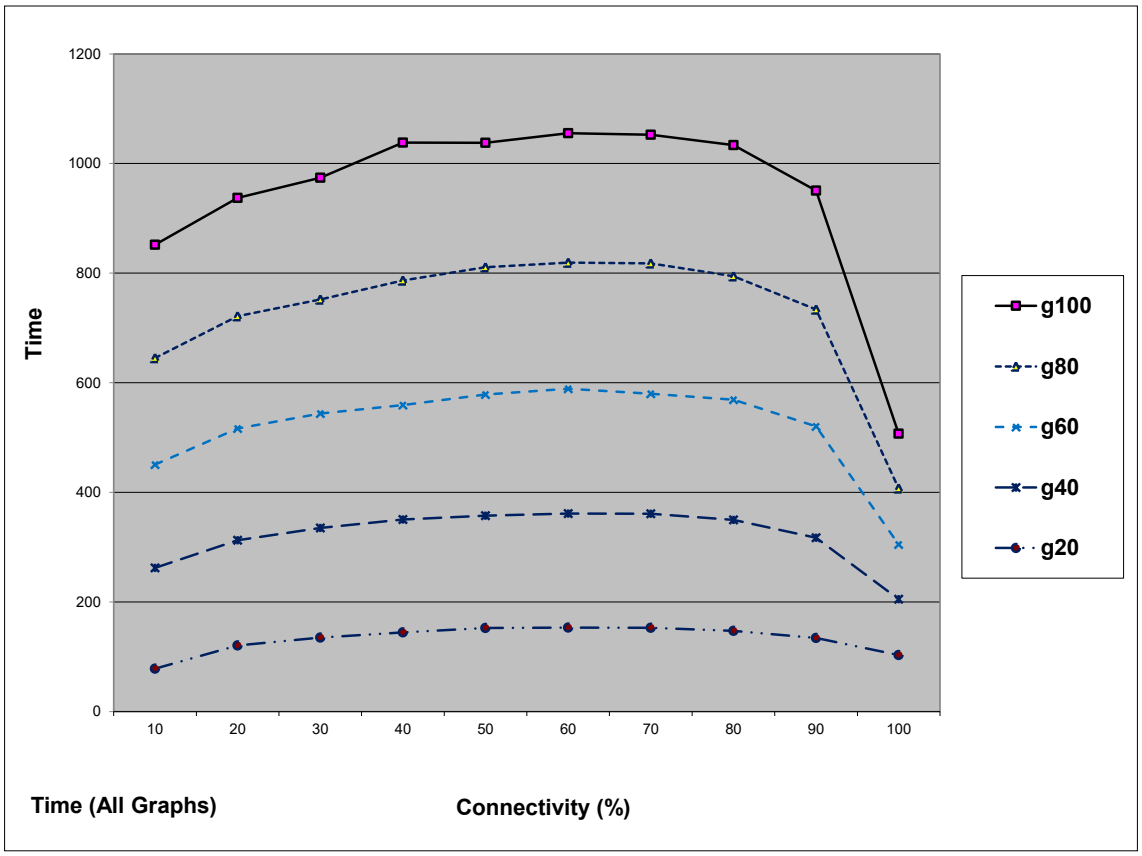

Figure 8.17: Time vs Connectivity

means high degrees, thus more shadow agents needed to shadow the exploration, this observation is actually quite obvious.

d. Regardless of the graph size, GREEDY and RANDOM differentiate in low connectivities, but gradually become similar at high connectivity. At 100% connectivity, the curves of movement cost, team size, and execution time for both algorithms all merge to one point. This can be easily explained. Complete graphs are completely symmetrical, i.e., choosing any one node as next target to explore gives exactly same simulation results.

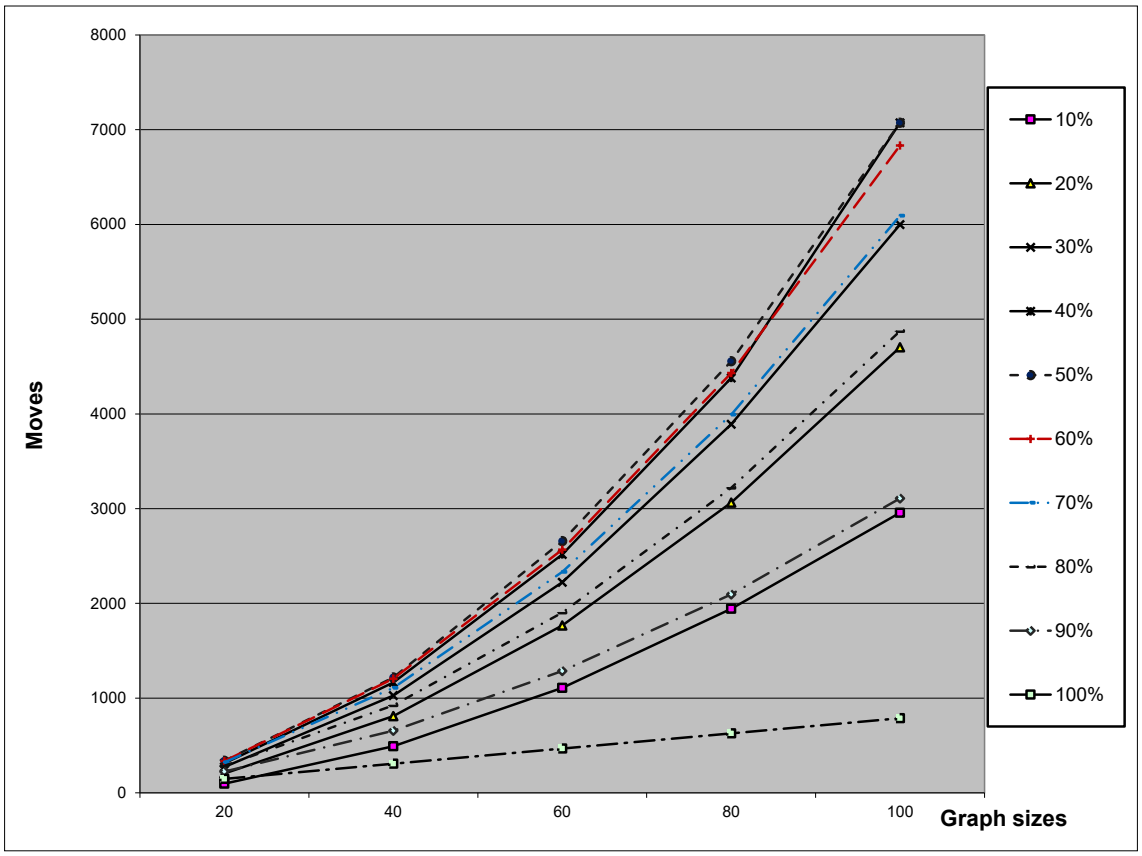

Figure 8.18: Movements vs Graph Size

Influence of graph size on complexity

For given connectivity levels, with the increase of graph sizes, the movement, agent, and time simulation results are shown in Figure 8.18, 8.19, and 8.20.

a. From Figure 8.18, it is observed that the movement costs seem to increase quadratically with the graph sizes. The movement costs for 10% and 90%, 20% and 80%, 30% and 70%, 40%, 50% and 60% are very close, respectively. Among all the connectivity levels, the complete graph has the lowest movement costs, while 40%, 50% and 60% connectivity levels have the highest movement costs. These behaviors demonstrate well that movement costs increase with connectivity levels. It saturates

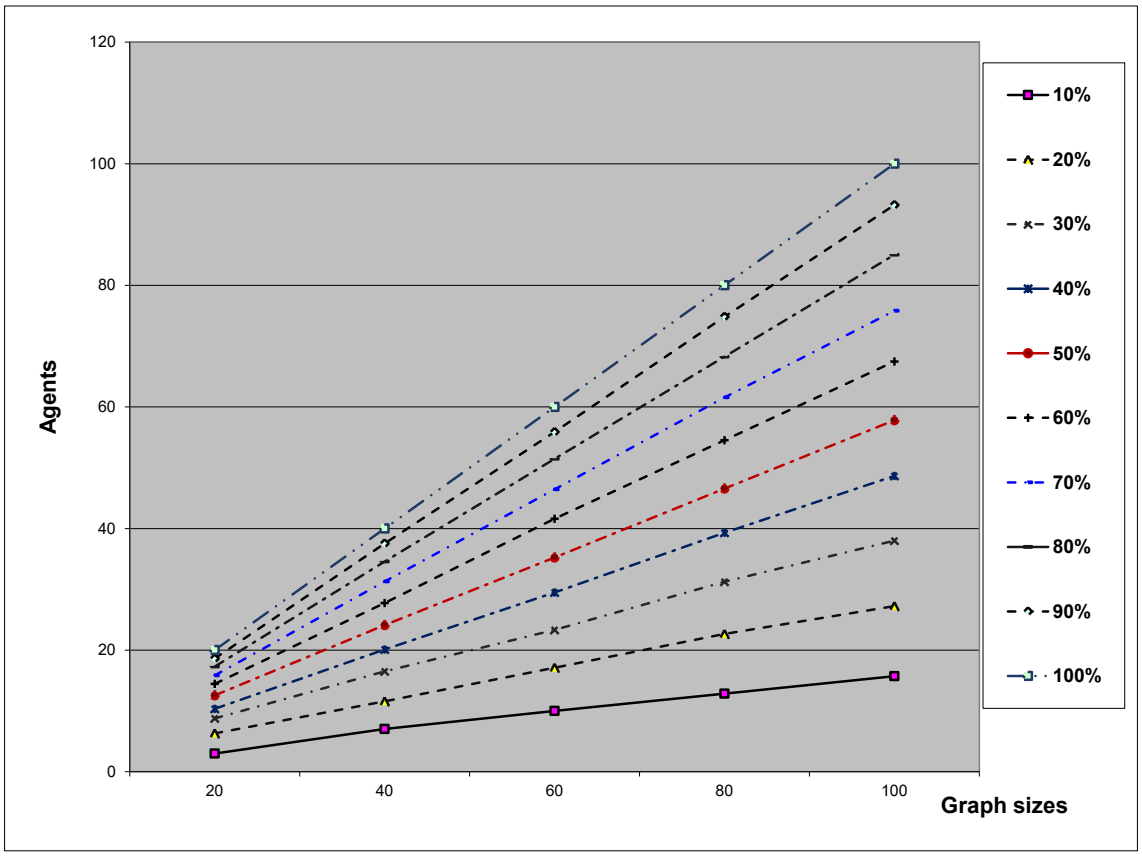

Figure 8.19: Agents vs Graph Size

close to the 50% connectivity, it then decreases toa minimum when connectivity is 100%.

b. Team size all increase when the graph size increases. From the Figure 8.19, it is observed that the Team size increases (linearly) with the graph size. This result seems to be counter-intuitive. However, analyzing the reason, we realize this is due to the way we generated random graphs. With the increase of connectivity, increasing the graph size means raising the average degrees of the graph. Therefore when the graph size increases, also the team size becomes large. As shown before, the connectivity linearly impacts the agents requirement.

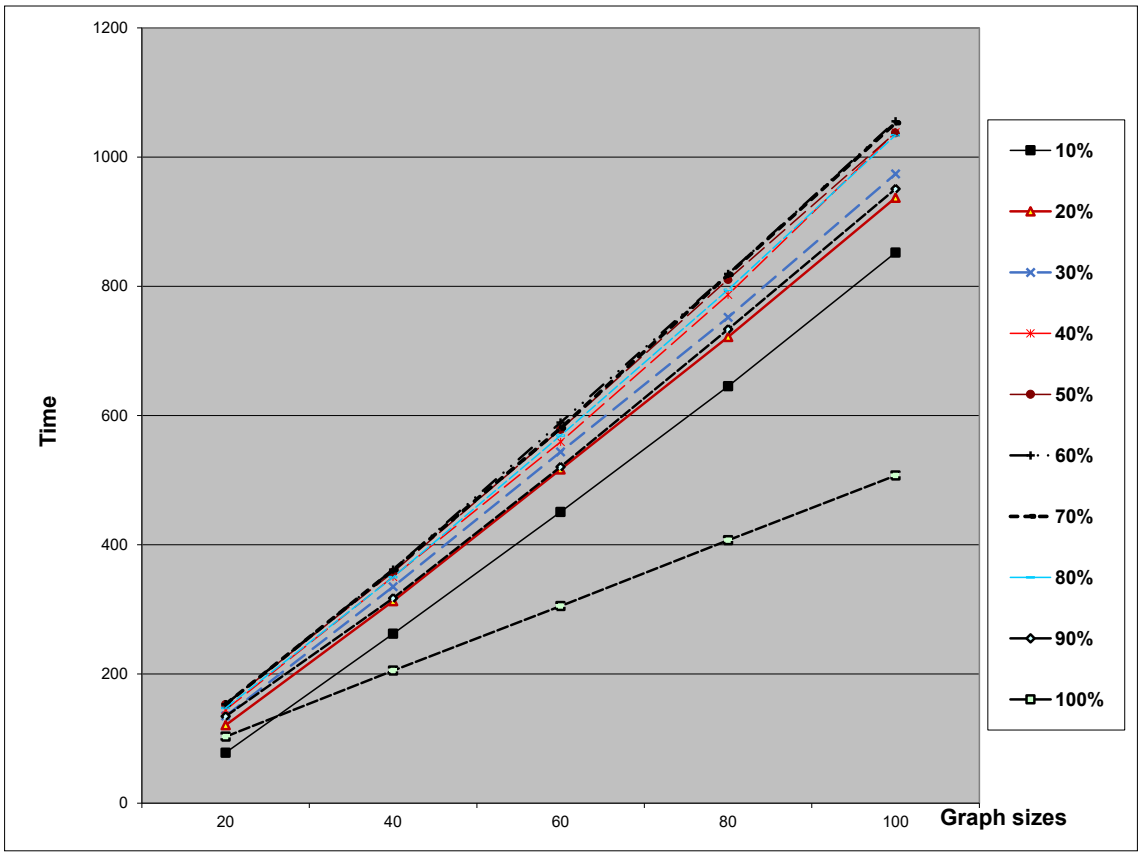

Figure 8.20: Time vs Graph Size

c. From the Figure 8.20, it is observed that the execution time seems to increase linearly with the graph sizes. The execution times for 20% - 80% are relatively close to each other. Among all the connectivity levels, the complete graph has the lowest time costs, while other connectivity levels have relatively high time costs. The reason is that when connectivity increases and approaches to 100%, the distances among nodes decrease and approach 1. Generally speaking, the time costs are lower than the movement costs for same graph size and connectivity level because many activities happen in parallel.

8.3 Discussion and Conclusions

The survey in Chapter 2 of some existing simulators for reactive distributed algorithms in network applications shows they provide/support various good features and advantages. However they usually have common limitations or disadvantages, for example, only support message passing and bi-directional link; algorithm implementation and network environment are tightly coupled, plus creating network topology needs some level of configuration or coding; statistics are not always provided by simulation results, so usually pre-coding or post processing are needed to obtain the statistics; do not support adversary events; and not always provide full capable debugging, and record and replay capabilities.

DisJ introduced in this research overcomes almost all the above shortcomings. In addition to providing basic functionalities, it offers mobile-agent support, flexible network element settings, simple APIs, user-friendly GUI, strong debugging capabilities. One of the main advantages is decoupling users' protocol developing activity from defining network topology and executing the protocol. This means intended protocol and network topology are developed, defined, and built separately from each other and from simulation engine. This also means a protocol can be run on different network topologies on simulation engine.

We considered the Greedy Exploration Algorithm for arbitrary networks described in Chapter 6 and we compared it it with Random Exploration. We investigated the behaviours of the two algorithms, their properties and performance through an extensive number of computer simulation runs. We observed that

Greedy beats Random for all graphs at each connectivity level. In Random, the residual degree varies dramatically, while in Greedy it changes smoothly. We also noticed that, when connectivity increases, the difference between the two solutions decreases. When connectivity approaches 100%, i.e., we have the complete graph, the

two strategies are the same.

By observing the exploration behavior starting from different home bases, we noticed that the exploration sequences are obviously influenced by graph structures. Some nodes with high degrees can block exploration (i.e., agents avoid these node at first). After breaking through the barriers, all search sequences follow the same/similar paths in the rest of the graph. It is observed that changing home base has effect on the exploration sequences only locally, but has no impacts on search sequence in far areas. Starting from the nodes with high degrees, we could reduce the $Size(G)$ and exploration cost.

With regard to statistics of simulation results, it is observed that the standard deviations for movements, agents, and time complexities are all very small compared with the average estimations. In particular, the larger graphs and the higher the connectivity levels, the better the statistical results are.

With regard to the influence of connectivity density on complexity, it is observed:

- Movement cost gradually increases to maximum at 40%-60% connectivities, then it gradually decreases. It is interesting to note that movement costs are close for different graph sizes at 100% connectivity level.

- Simulation time cost shows gradually increase to maximum at middle connectivity levels, then it gradually decreases. It is noted that time costs are close for different graph sizes at 100% connectivity level.

- Team size continuously increases with the increase of connectivity. It reaches the maximum, i.e., graph size, at 100% connectivity level.

- Regardless of the graph size, GREEDY and RANDOM differentiate in low connectivities, but gradually merge at high connectivity. At 100% connectivity, the

curves of movement cost, team size, and execution time for both algorithms all merge to one point.

With regard to the influence of graph size on complexity, it is observed:

- The movement costs seem to increase quadratically with the graph sizes.
- Team size all increase when graph size increases.
- it is observed that the execution time seems to increase quadratically close to linearly with the graph sizes.

Comparing the complexity analysis results in Chapter 6, in asynchronous and synchronous environments, the protocols all use $O(n^2\Delta)$ movements in the Greedy algorithm. When the graphs become denser, i.e., Δ approaches to graph size n, the worst case movement complexity seems to increase to $O(n^3)$. The actual movement complexity from our simulations, however, seems to be close to $O(n^2)$ instead of $O(n^3)$. The reason might be that the distance between any nodes decreases dramatically when the graphs become denser. In addition, SA and LEA do not need to move much to explore new nodes because nodes tend to share common neighbours.

DisJ has many features and advantages. However, one of the main inconvenience we observed is that running a protocol ona large number of graphs (in our case, 2255 graphs) is very time-consuming, mainly because there is no way to set up a batch set of simulations.

In our simulations, we do not use very large graphs (for example, with thousands of nodes) but relatively small and moderate size graphs whose sizes are less than 100 nodes. The reason is that the main purposes of the BVD simulation are to run simulations at different connectivity levels to understand how connectivity level impacts the exploration behavior, and the various complexities, i.e., movement, time,

and agent size. Graphs are randomly generated, and, at the same connectivity level, large graphs can be viewed as scaled up from relatively small or moderate size graphs. So the exploration behavior and complexity trends demonstrated in very large graphs should not be different from those observed in relatively small graphs or moderate size graphs.

8.4 Summary

In this chapter, the BVD problem is investigated experimentally by using DisJ. DisJ overcomes the shortcomings and combines many advantages of existing simulators surveyed. One of the main advantages is that the intended protocol (i.e., algorithm) and network topology can be developed, defined, and built separately. A large number of simulations on different sizes of graphs with many connectivity densities are carried out. The algorithm beats random exploration for all graphs at each connectivity level. The simulation results disclose many interesting behaviors and demonstrate the worst case complexity analysis of the solution protocol. In addition to proving the analytical results, the simulation also provides deep understanding on influence of graph connectivity density and size on complexities. The movement, time, and agent size increase with connectivity, but movement and time start decreasing after reaching maximum at 40quadratically and linearly respectively with the graph size. With regard to statistics of simulation results, it is observed that the standard deviations for movements, agents, and times are all very small compared with average estimations. Particularly the larger graphs and the higher the connectivity levels, the better statistical results are.

Chapter 9

Concluding Remarks and Future Work

9.1 Summary

Mobile agents are widely used in distributed and networked systems; however, the support of mobile agents can also cause security issues and threats to the network. In particular, a malicious agent can cause computer nodes to malfunction or crash by contaminating or infecting them; additionally, a contaminated or infected host in turn can destroy working agents for various malicious purposes. These situations are categorized as *harmful agent* and *harmful host*.

A harmful host is also called *black hole*: a network node infected by a process which destroys any incoming agent without leaving any detectable trace of the destruction. The problem of locating a black hole, called *Black Hole Search (BHS)*, has been extensively studied. A black hole is a presence which is harmful to agents but it is *static*, that is, it does not propagate in the network and so it is not harmful to other sites.

The theoretical work related to harmful agents has focused on the problem called *intruder capture* (IC) (also known as *graph decontamination* and *connected graph search*): an extraneous mobile agent, the intruder, moves through the network infecting the visited sites; the task is to decontaminate the network using a team of system agents avoiding recontamination. This problem has also been extensively studied. Let us point out that, in this problem, the harmful presence (the intruder) is mobile and harmful to the network sites, but does not cause any harm to the system agents.

The studies on *BHS* have not considered the transition characteristics of the harmful nodes: black holes do not move and their number does not change; similarly, the studies on IC have not considered the case of a mobile intruder harmful for both sites and agents. The problem we consider in this research combines the aspects of *BHS* with some of network decontamination: the harmful process is mobile (like an intruder) and harmful also to the system agents (like a black hole). In this thesis we define the *black virus decontamination* (BVD) problem and start the study of its solutions.

This chapter highlights the main contributions of this thesis to the BVD problem:

- The BVD problem integrates in its definition both the harmful aspects of the classical *BHS* problem with the mobility aspects of the classical *IC* or *network decontamination* problem. Thus it is the first attempt to model mobile intruders harmful not only for the sites but also for the agents. The main focus of the thesis work is on the protocols for agents to complete this task by causing minimum network damage and by scarifying minimum number of agents.

- The first part of the thesis describes the terminology and the model. We distinguish between *Sterile* and *Fertile* BV depending on whether or not, once triggered, the BV loses its spreading abilities. In the first case, the problem reduces to an exploration to find the black virus and in a simple disinfection of the triggered sites, in the other case, instead, the problem is also to protect the network from further spread after detecting the original black virus. Basic facts and characteristics of the problem are established. In particular, it is observed that monotonicity (i.e., any explored nodes are never re-contaminated) is necessary for optimality: in every network, to minimize the spread, a solution

protocol must be monotone. Basic tools and general strategy based on the concept of residual degree of the networks to solve the problem are provided. We propose a general solution strategy for the problem in various settings: with full knowledge of the topology, with only neighbouring knowledge (neighbours at maximum distance two), in both fertile and sterile models. We notice that full knowledge of the topology is actually not necessary as we can provide optimal solutions with just neighbouring knowledge. The general strategy consists of a careful exploration of the network to find and clean the BVs. The strategies behind the exploration depend on the topology under consideration, as well as on the model considered.

- The second part of the thesis focuses on the BVD problem for three important classes of interconnection network topologies: (multi-dimensional) *grids*, *tori*, and *hypercubes*. For each class, a monotone solution protocol, which decontaminates the networks regardless of the number of dimensions, is provided and the complexity of the solution is analyzed. All the protocols are optimal both in terms of spread (total number of casualties) and size (total number of agents), and are asymptotically optimal in the total number of movements. A summary of the results is shown Table 1.1.

- The third part of the thesis concentrates on exploring a BV in arbitrary graph. The main challenges of the BVD problem in arbitrary graph are discussed. Monotonic solution protocols are developed to achieve the objective of locating and cleaning the BV with optimal spreads in asynchronous settings. We notice that knowledge of the topology is not necessary for optimality; in fact, it is sufficient that each node has knowledge of its neighbours at distance 2. We devise two variants of the exploration strategy: Greedy and Threshold. For both

protocols, the worst case complexity (spreads, size, and movements) is analyzed and obtained. A summary of the results is shown in Table 1.2. The complexity results are not the same in the case of sterile and productive BV, assuming that, in case of fertile BVs, the triggered BVs do not disconnect the network. If instead the triggered BVs disconnects the network, the spread could be unavoidably high because it would infect all the disconnected components which do not contain any agent (in the thesis we do not consider this case). In addition, an interesting connection is established between the solutions of the BVD problem and the problem of determining rooted acyclic orientations of unoriented graphs with minimum outdegrees. An algorithm is developed to provide a distributed solution to this graph optimization problem. The complexity, i.e., agent movement, is analyzed and obtained.

- The fourth part of thesis extends the investigation to MBVD. The challenges due to the existence of multiple BVs in graphs and the basic characteristics of MBVD are discussed. Particularly *separation* and its impacts on spreads, and the distribution of original BVs in graphs and its influence on maximum casualties are investigated. The general strategy of the MBVD is to combine shadow exploration and cleaning phases together. Once triggering a BV, all exposed BVs are recorded, and all residual degrees of unexplored nodes (including exposed BV nodes) are updated. Agents continue to perform shadow exploration based on minimum residual degree. Whenever the exposed BVs residual degree reaches zero, agents explore (i.e., clean) it. The complexities of MBVD in arbitrary graph are studied. After understanding the solution to the MBVD in arbitrary graph, the investigation continue to the MBVD in special graphs previously studied, i.e., regular special graphs (i.e., Tori and Hypercube) and irregular special graphs (i.e., 2D, 3D, and qD grids). The solution protocols in

these graphs are presented, and the complexities of these solution protocols are discussed.

- The fifth part of thesis shifts the theoretical study to experimental investigation by using DisJ, a simulation platform. DisJ overcomes almost all the shortcomings and combines many advantages of existing simulators surveyed in Chapter 2. One of the main advantages is that the intended protocol and network topology can be developed, defined, and built separately. Large number of simulations on different sizes of graphs with many connectivity densities are carried out. The algorithm beats random exploration for all graphs at each connectivity level. The simulation disclosed many interesting behaviors of the solution protocol. Simulation demonstrates the worst case complexity analysis in [19]. In addition to proving the analytical results, simulation provides deep understanding on influence of graph connectivity density and size on complexities (movement, time, and agent size). The movement, time, and agent size increase with connectivity, but movement and time start decreasing after reaching maximum at 40%-60% connectivities. The movement, and time and agent size seem to increase quadratically and linearly respectively with the graph size. With regard to statistics of simulation results, it is observed that the standard deviations for movement, agent, and time are all very small compared with average estimations. Particularly the larger graphs and the higher the connectivity levels, the better statistical results are.

To conclude, this research makes a significant scientific contribution to address network security issues caused by mobile agents in networks. It combines the characteristics of black hole and intruder capture problems to creates a brand new model (BV) to depict dynamic natures of contamination or damages caused by malicious

mobile agents in networks. It provides the theoretical and practical framework to solve black virus decontamination problem in various networks. Solution protocols for special and arbitrary networks with single and multiple BVs are developed, their complexities related to spreads of BVs, team of agents, total agent movements, and execution time are analyzed. Computer simulation work has been performed to demonstrate the implementability of the solution protocols, to support theoretical analyses, and to provide deep understanding on the behavior of the agents' exploring BVs in graphs.

The research has covered decontamination from a single BV and from multiple BVs in asynchronous settings in special and arbitrary graphs.

9.2 Open Problems and Future Research

The results of this thesis open many research problems and pose new questions. Future studies will include the following research:

- In addition to grids, tori, and hypercubes, future research can be extended to explore and decontaminate BVs in other special network topologies.
- The BVD process always starts a given node, the homebase, and the agents try to find a search path which results in an overall minimum damage to the graph. Future research can be extended to design effective algorithms for agents to find a homebase for given a graph to decontaminate BVs from the graph with minimum damage.
- Current simulation work is mainly on graphs with one BV, more simulation work can be performed for the networks which contain multiple BVs.
- With regard to the DisJ simulation platform, one obvious shortcoming is its lack of automatic features to process a batch of simulation graphs. Each graph

has to be manually loaded to a simulation protocol (algorithm). It is time-consuming to run simulations on a large number of sample graphs to generate statistics. Future DisJ enhancement might include adding automatic features to process a batch of simulation graphs.

This investigation is a starting point for a better understanding of the impact that the severity of the security threat (i.e., the power of the harmful entity) has on the complexity of the defence mechanism (i.e., the complexity of the solution protocol). For example, although a black virus combines some harmful properties of a black hole (as defined by the black hole search problem) with some of an intruder (as defined by the intruder capture problem), a black virus it is not all powerful. In fact the mobility is restricted to be reactive, the spreading is limited to the immediate neighbourhood, and when it moves from a node it leave it clean. Interesting and important research directions are to investigate, for example, which of these restrictions is the most severe, which can be reduced or lifted and the problem still be solvable, etc.; in other words, to determine whether the system is capable of sustaining a more powerful threat, how much more power the agents would need to have to be able to sustain a stronger threat, etc.

Bibliography

[1] B. Alspach, "Searching and sweeping graphs: a brief survey," *Le Matematiche.*, Vol.LIX, pp. 5-37, 2004.

[2] K. A. Amin and A. R. Mikler, "Agent-based distance vector routing: a resource efficient and scalable approach to routing in large communication networks," *Journal of Systems and Software*, pp. 215-227, 2004.

[3] Y. Asahiro, J. Jansson, and E. Miyano, "Graph orientation to maximize the minimum weighted outdegree," *International Journal Foundations of Comput. Sci.*, 22(3), pp. 583, 2011.

[4] Y. Asahiro, J. Jansson, E. Miyano, H. Ono, and K. Zenmyo, "Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree," *Algorithmic Aspects in Information and Management*, Vol. 4508, pp. 167-177, 2007.

[5] Y. Asahiro, E. Miyano, and H. Ono, "Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree," *Discrete Applied Mathematics*, 159(7), pp. 498-508, 2011.

[6] Y. Asahiro, E. Miyano, H. Ono, and K. Zenmyo, "Graph orientation algorithms to minimize the maximum outdegree," *12th Australasian Theory Computing Symposium*, Vol. 51, pp. 11-20, 2006.

[7] B. Awerbuch, M. Betke, R. Rivest, and M. Singh, "Piecemeal graph exploration by a mobile robot," *8th Conference on Computational Learning Theory (COLT)*, ACM Press, pp. 321-328, 1995.

[8] B. Balamohan, S. Dobrev, P. Flocchini, and N. Santoro, "Asynchronous exploration of an unknown anonymous dangerous graph with O(1) pebbles," *19th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pp. 279-290, 2012.

[9] B. Balamohan, P. Flocchini, A. Miri, and N. Santoro, "Time optimal algorithms for black hole search in rings," *Combinatorial Optimization and Applications*, Vol. 6509, pp. 58-71, 2010.

[10] L. Barrière, P. Flocchini, F. V. Fomin, P. Fraignaud, N. Nisse, N. Santoro, and D. M. Thilikos, "Connected graph searching," *Information and Computation*, pp. 1-16, 2012.

[11] L. Barrière, P. Flocchini, P. Fraignaud, and N. Santoro, "Capture of an intruder by mobile agents," *14th Symposium on Parallel Algorithms and Architectures (SPAA)*, pp. 200-209, 2002.

[12] L. Barri'ere, P. Flocchini, P. Fraigniaud, and N. Santoro, "Can we elect if we cannot compare?" *15th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, ACM Press, pp. 324-332, 2003.

[13] L. Barri'ere, P. Flocchini, P. Fraigniaud, and N. Santoro, "Election and rendezvous in fully anonymous systems with sense of direction," *Proc. of the 10th International Colloquium on Structural Information Complexity (SIROCCO)*, Carleton Scientific, pp. 17-32, 2003.

[14] M. Ben-Ari, "Interactive execution of distributed algorithms," *ACM Journal of Education Resources in Computing*, Article 2, 2001.

[15] M. Bender and D. Slonim, "The power of team exploration: Two robots can learn unlabeled directed graphs," *35th Symposium on Foundations of Computer Science (FOCS)*, pp. 75-85, 1994.

[16] L. Blin, P. Fraignaud, N. Nisse, and S. Vial, "Distributed chasing of network intruders," *Theoretical Computer Science*, pp. 70-84, 2006.

[17] J. Cai, "Experimental analysis of black virus decontamination by DisJ," *4th International Conference on Intelligent Systems and Applications (INTELLI)*, 2015

[18] J. Cai, P. Flocchini, and N. Santoro, "Network decontamination from a black virus," *27th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 696-705, 2013.

[19] J. Cai, P. Flocchini, and N. Santoro, "Decontaminating a network from a black virus," *International Journal of Networking and Computing*, 4(1), pp. 151-173, 2014.

[20] J. Cai, P. Flocchini, and N. Santoro, "Black virus decontamination in arbitrary networks," *3rd Conference on Information Systems and Technologies*, Advances in Intelligent Systems and Computing 353, Springer, pp. 991-1000, 2015.

[21] J. Cai, P. Flocchini, and N. Santoro, "Distributed black virus decontamination and constrained acyclic orientations," *13th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*, 2015.

[22] J. Chalopin, S. Das, A. Labourel, and E. Markou, "Tight bounds for black hole search with scattered agents in synchronous rings," *Theoretical Computer Science*, 509, pp. 70-85, 2013.

[23] J. Chalopin, S. Das, A. Labourel, and E. Markou, "Black hole search with finite automata scattered in a synchronous torus," *25th International Symposium on Distributed Computing (DISC)*, pp. 432-446, 2011.

[24] J. Chalopin, S. Das, and N. Santoro, "Rendezvous of mobile agents in unknown graphs with faulty links," *21st International Symposium on Distributed Computing (DISC)*, pp. 108-122, 2007.

[25] J. Chalopin, E. Godard, Y. Metivier, and R. Ossamy, "Mobile agent algorithms versus message passing algorithms," *Principles of Distributed Systems, Lecture Notes in Computer Science*, pp 187-201, 2006.

[26] M. Charikar, "Greedy approximation algorithms for finding dense components in a graph," *3rd Intl. Workshop on Approximation Algoritms for Combinatorial Optimization*, Vol. 1913, pp. 84-95, 2000.

[27] I. Chatzigiannakis, A. Kinalis, A. Poulakidas, G. Prasinos, and C. Zaroliagis, "DAP: a generic platform for the simulation of distributed algorithms," *37th Annual Symposium on Simulation (ANSS)*, pp. 166-177, 2004.

[28] K. Cheung and S. Ng, "Protecting mobile agents against malicious hosts by intention of spreading," *Int. Conf. on Parallel and Distributed Processing and Applications (PDPTA)*, pp. 725-729, 1999.

[29] C. Cooper, R. Klasing, and T. Radzik, "Searching for black-hole faults in a network using multiple agents," *10th International Conference on Principles of Distributed Systems (OPODIS)*, pp. 320-332, 2006.

[30] C. Cooper, R. Klasing, and T. Radzik, "Locating and repairing faults in a network with mobile agents," *15th Structural Information and Communication Complexity (SIROCCO)*, pp. 20-32, 2008.

[31] J. Czyzowicz, S. Dobrev, R. Královic, S. Miklík, and D. Pardubská, "Black hole search in directed graphs," *16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pp. 182-194, 2009.

[32] J. Czyzowicz, D. R. Kowalski, E. Markou, and A. Pelc, "Complexity of searching for a black hole," *Fundamenta Informaticae*, pp. 229-242, 2006.

[33] J. Czyzowicz, D. R. Kowalski, E. Markou, and A. Pelc, "Searching for a black hole in synchronous tree networks," *Combinatorics, Probability & Computing*, pp. 595-619, 2007.

[34] S. Das, P. Flocchini, A. Nayak, and N. Santoro, "Map construction of unknown graphs by multiple agents," *Theoretical Computer Science* 385 (1-3): 34-48, 2007.

[35] D. Dereniowski. "Connected searching of weighted trees," *Theoretical Computer Science*, pp. 5700-5713, 2011.

[36] A. Dessmark, P. Fraigniaud, and A. Pelc, "Deterministic rendezvous in graphs," *11th European Symposium on Algorithms (ESA)*, Vol. 2832, pp. 184-195, 2003.

[37] Distributed Computing Group, "SinAlgo - simulator for network algorithms," *http://www.disco.ethz.ch/projects/sinalgo/*, 2015.

[38] S. Dobrev, P. Flocchini, R. Královic, P. Ruzicka, G. Prencipe, and N. Santoro, "Black hole search in common interconnection networks," *Networks*, pp. 61-71, 2006.

[39] S. Dobrev, P. Flocchini, R. Královic, and N. Santoro, "Exploring an unknown dangerous graph using tokens," *Theoretical Computer Science*, pp. 28-45, 2013.

[40] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro, "Searching for a black hole in arbitrary networks: optimal mobile agents protocols," *Distributed Computing*, pp. 1-18, 2006.

[41] S. Dobrev, P. Flocchini, and N. Santoro, "Mobile search for a black hole in an anonymous ring," *Algorithmica*, pp. 67-90, 2007.

[42] S. Dobrev, N. Santoro, and W. Shi, "Using scattered mobile agents to locate a black hole in a unoriented ring with tokens," *Int. Journal of Foundations of Computer Science*, pp. 1355-1372, 2008.

[43] R. C. Entringer, "Characterizations of graphs having orientations satisfying local degree restrictions," *Czechoslovak Mathematical Journal*, pp. 108-119, 1978.

[44] P. Erdös. "Some unsolved problems in graph theory and combinatorial analysis," *Combinatorial Mathematics and its Applications*, 3125, 1971.

[45] P. Flocchini, M. J. Huang, and F. L. Luccio, "Decontamination of hypercubes by mobile agents," *Networks*, pp. 167-178, 2008.

[46] P. Flocchini, M. J. Huang, and F. L. Luccio, "Decontaminating chordal rings and tori using mobile agents," *International Journal on Foundations of Computer Science*, pp. 547-563, 2006.

[47] P. Flocchini, D. Ilcinkas, and N. Santoro, "Ping pong in dangerous graphs: optimal black bole search with pebbles," *Algorithmica*, pp. 1006-1033, 2012.

[48] P. Flocchini, M. Kellett, P. Mason, and N. Santoro, "Map construction and exploration by mobile agents scattered in a dangerous network," *24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1-10, 2009.

[49] P. Flocchini, M. Kellett, P. Mason, and N. Santoro, "Searching for black holes in subways," *Theory of Computing Systems*, pp. 158-184, 2012.

[50] P. Flocchini, F.L. Luccio, and L. X. Song, "Size optimal strategies for capturing an intruder in mesh networks," *International Conference on Communications in Computing (CIC)*, pp. 200-206, 2005.

[51] P. Flocchini, F. Luccio, L. Pagli, N. Santoro. "Network decontamination under m-immunity," *Discrete Appplied Mathematics*, to appear, 2015.

[52] P. Flocchini and N. Santoro, *Distributed Security Algorithms for Mobile Agents*, Book Chapter of: J. Cao, S. Das (Eds.), Mobile Agents in Networking and Distributed Computing, Wiley 2012.

[53] F. V. Fomin and D. M. Thilikos, "An annotated bibliography on guaranteed graph searching," *Theoretical Computer Science*, pp. 236-245, 2008.

[54] F. V. Fomin, D. M. Thilikos, and I. Todineau, "Connected graph searching in outerplanar graphs," *7th International Conference on Graph Theory (ICGT)* , pp. 213–216, 2005.

[55] A. Frank, "On the orientation of graphs," *Journal of Combinatorial Theory, Series B*, 28(3), pp. 251-261, 1980.

[56] A. Frank, and A. Gyárfás, "How to orient the edges of a graph," *Colloqia Mathematica Societatis Janos Bolyai*, 1976.

[57] T. Gallai, "On the directed paths and circuits," *Theory of Graphs*, 38:2054, 1968.

[58] P. Glaus, "Locating a black hole without the knowledge of incoming link," *5th Int. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSOR)*, pp. 128-138, 2009.

[59] A. V. Goldberg, "Finding a maximum density subgraph," *Technical Report, University of California at Berkeley, USA*, 1984.

[60] S. L. Hakimi, "On realizability of a set of integers as degrees of the vertices of a linear graph: *Journal of the Society for Industrial and Applied Mathematics*, 10(3), pp. 496-506, 1962.

[61] S. L. Hakimi, "On the degrees of the vertices of a directed graph," *Journal of the Franklin Institute*, 279(4), pp. 290–308, 1965.

[62] F. Harary, E. Palmer, and C. Smith, "Which graphs have only self-converse orientation?" *Canad. Math.*, pp. 425-429, 1967.

[63] B. Heath, R. Hill, and F. Ciarallo, "A survey of agent-based modeling practices (January 1998 to July 2008)," *Journal of Artificial Societies and Social Simulation*, 12(4), 2009.

[64] F. Hohl, "Time limited black box security: protecting mobile agents from malicious hosts," *Mobile Agents and Security*, Lecture Notes in Computer Science, pp. 92-113, 1998.

[65] F. Hohl, "A framework to protect mobile agents by using reference states," *20th Int. Conf. on Distributed Computing Systems (ICDCS)*, pp. 410-417, 2000.

[66] D. Ilcinkas, N. Nisse and D. Soguet, "The cost of monotonicity in distributed graph searching," *Distributed Computing*, pp. 117-127, 2009.

[67] N. Imani, H. Sarbazi-Azadb, and A. Y. Zomaya, "Capturing an intruder in product networks," *Journal of Parallel and Distributed Computing*, pp. 1018-1028, 2007.

[68] N. Imani, H. Sarbazi-Azad, A. Y. Zomaya, and P. Moinzadeh, "Detecting threats in star graphs," *IEEE Transactions on Parallel and Distributed Systems*, pp. 474-483, 2009.

[69] M. Ji, and M. Egerstedt, "Distributed coordination control of multiagent systems while preserving connectedness," *IEEE Transactions on Robotics*, pp. 693-703, 2007.

[70] S. Khuller and B. Saha, "On finding dense subgraphs," *Automata, Languages and Programming*, Lecture Notes in Computer Science, Vol. 5555, pp. 597-608, 2009.

[71] W. Kin, V. Chan, and Y.J. Son, "Agent-based simulation tutorial - simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation," *2010 Winter Simulation Conference*, 2010.

[72] H. Kim, I. Miklos, and Z. Toroczkai, "On realizing all simple graphs with a given degree sequence," *Discrete Mathematics*, 2008.

[73] R. Klasing, E. Markou, T. Radzik, and F. Sarracco, "Hardness and approximation results for black hole search in arbitrary networks," *Theoretical Computer Science*, pp. 201-221, 2007.

[74] R. Klasing, E. Markou, T. Radzik, and F. Sarracco, "Approximation bounds for black hole search problems," *Networks*, pp. 216-226, 2008.

[75] M. Kona and Z. Xu, "A framework for network management using mobile agents," *27th IEEE International Parallel and Distributed Processing Symposium (IPDPS)* , pp. 15-19, 2002.

[76] A. Kosowski, A. Navarra, and C. M. Pinotti, "Synchronous black hole search in directed graphs," *Theoretical Computer Science*, pp. 5752-5759, 2011.

[77] A. Kowalik, "Approximation scheme for lowest outdegree orientation and graph density measures," *Algorithms and Computation*, Lecture Notes in Computer Science, Vol.4288, pp. 557-566, 2006.

[78] R. Královic and S. Miklík, "Periodic data retrieval problem in rings containing a malicious host," *17th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pp. 157-167, 2010.

[79] D. Lange and M. Oshima, "Seven good reasons for mobile agents," *Communication of the ACM*, 42(3), pp. 88-89, 1999.

[80] G. Liu and L. Wong, "Effective pruning techniques for mining quasi-cliques," *Machine Learning and Knowledge Discovery in Databases*, Vol. 5212, pp. 33-49, 2008.

[81] F. Luccio, "Contiguous search problem in Sierpinski graphs," *Theory of Computing Systems*, pp. 186-204, 2009.

[82] F. Luccio and L. Pagli, "A general approach to toroidal mesh decontamination with local immunity," *23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS)* , pp. 1-8, 2009.

[83] F. Luccio, L. Pagli, and N. Santoro, "Network decontamination in presence of local immunity," *International Journal of Foundation of Computer Science*, pp. 457-474, 2007.

[84] C. M. Macal and M. J. North, "Tutorial on agent-based modeling and simulation," *Journal of Simulation*, pp. 151-162, 2010.

[85] N. Minar, K. Kramer, and P. Maes, "Cooperating mobile agents for mapping networks," *1st Hungarian National Conference on Agent Based Computing*, pp. 287-304, 1999.

[86] C. Nikolai and G. Madey, "Tools of the trade: a survey of various agent based modeling platforms," *Journal of Artificial Societies and Social Simulation*, 12(22), 2009.

[87] N. Nisse, "Connected graph searching in chordal graphs," *Discrete Applied Mathematics*, pp. 2603-2610, 2009.

[88] R. Nowakowski and P. Winkler, "Vertex-to-vertex pursuit in a graph," *Discrete Math.* Vol. 43, pp. 235-239, 1983.

[89] R. Oechsle and T. Gottwald, "DisASTer (distributed algorithms simulation terrain): a platform for the implementation of distributed algorithms," *10th SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pp. 44-48, 2005.

[90] R. Olfati-Saber and R. M. Murray, "Consensus protocols for networks of dynamic agents," *Amer. Control Conf.*, pp. 951-956, 2003.

[91] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, pp. 1520-1533, 2004.

[92] S. F. Railsback, S. L. Lytinen, and S. K. Jackson, Agent-based simulation platforms: review and development recommendations. *SIMULATION*, pp. 609-623, 2006.

[93] C. W. Reynolds, "Flocks, herds, and schools: a distributed behavioral model 1," *Computer Graphics*, pp. 25-34, 1987.

[94] H. E. Robbins, "A theorem on graphs with an application to a problem of traffic control," *Amer. Math. Monthly*, Vol. 46, pp. 281-283, 1939.

[95] M. G. Rubinstein and O.C.M.B. Duarte, "Evaluating the performance of mobile agents in network management," *Global Telecommunications Conference (GLOBECOM)*, pp. 386-390, 1999.

[96] T. Sander and C.F. Tschudin, "Protecting mobile agents against malicious hosts," *Mobile Agents and Security, Lecture Notes in Computer Science*, pp. 44-60, 1998.

[97] N. Santoro, "Development of a simulation engine for distributed algorithms: system manual," *Internal Report, Carleton University*, 2001.

[98] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, 1(1), pp. 27-64, 2007.

[99] W. Schreiner, "A java toolkit for teaching distributed algorithms," *7th annual conference on Innovation and technology in computer science education*, pp. 111-115, 2002.

[100] P. Shareghi, H. Sarbazi-Azad, and N. Imani, "Capturing an intruder in the pyramid," *International Computer Science Symposium in Russia (CSR)* , pp. 580-590, 2006.

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

[102] B. Yanga, D. Dyerb, and B. Alspach, "Sweeping graphs with large clique number," *Discrete Mathematics*, pp. 5770-5780, 2009.

[103] M. Yubao and D. Renyuan, "Mobile agent technology and its application in distributed data mining," *1st International Workshop on Database Technology and Applications*, pp. 151-155, 2009.