uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Jun Qiu

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M. (Computer Science)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Best Effort Decontamination of Networks

TITRE DE LA THÈSE / TITLE OF THESIS

Paola Flocchini

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Amiya Nayak

Evangelos Kranakis

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Best Effort Decontamination of Networks

by

Jun Qiu

A thesis submitted to
the Faculty of Graduate and Postdoctoral Studies
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Supervisor: **Dr. Paola Flocchini**

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-49267-3
Our file Notre référence
ISBN: 978-0-494-49267-3

Abstract

In this thesis we consider the problem to find the optimal strategy to decontaminate the maximum possible number of nodes in a contaminated network with a fixed number of agents. We are given a team of mobile agents located on a node in a contaminated network and the number of agents is not enough to decontaminate the whole network to reach a state in which all nodes are simultaneously clean. We want to find what the maximum number of decontaminated nodes is and how to decontaminate them. In this thesis we consider meshes (regular, octagonal, and hexagonal) and trees and give optimal strategies for those topologies. We also analyze the performance of our strategies according to the number of decontaminated nodes, number of agents' movement and time.

Acknowledgments

I would like to extend my sincere appreciation to those who help me to complete this thesis. Especially I want to thank my supervisor Paola Flocchini, who has given me invaluable suggestion and illumination in the whole process to write this thesis. It is a great pleasure to work under her direction. Thanks to my family for their love to encourage me to go forward in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Decontamination Problem

Mobile agent computing has been a rapidly developing area of research in recent years. Mobile agents are self-contained programs that can move from a node to a neighbouring node to execute tasks independently of each other or to cooperate to solve problems.

The use of mobile agents is becoming increasingly popular for computing in networked environments, both as a theoretical framework and as a programming platform.

Some problems for mobile agents that have been systematically studied from an algorithmic point of view are, for example, the following: *Exploration*, *Map Construction*, and *Rendezvous*. The exploration problem consists of having the agents traverse an unknown network; in the map construction problem, the agents have to terminate the exploration with a map of the network. These two problems have been studied mostly for a single

agent (e.g., see [2, 13, 16, 37, 38]); some results about two agents exploring directed graphs have been given in [42] and k agents exploring trees have been studied in [22]. In the rendezvous problem, the agents have to meet on a node of the network. Also this problem has been extensively investigated, especially in randomized or synchronous settings; for recent surveys, see papers [3, 26].

In all these examples, however, the network is assumed to be a safe environment; that is, the agents are assumed to be able to freely and safely move in the network to perform their tasks. An interesting issue to consider is when either the nodes (hosts) or the agent can pose some danger to the network. For example, the problem to detect a harmful host has been intensively studied from a programming point of view [24, 46, 47], but there are not many algorithmic solutions. From an algorithmic point of view, the problem of determining and reporting the location of a bad host (the *Black-Hole Search*) has been studied using a team of mobile agents under various different scenarios (e.g., see [14, 15, 9, 11, 12, 29, 30]). Another important issue, which we consider in this thesis, concerns harmful agents (or intruders). Intruders are harmful mobile agents penetrated into a network environment. If some malicious intruders invade some nodes of network, they could "contaminate" the network and cause malfunction. A very important problem is to detect the intruder and clean the contaminated nodes to increase the security of the network. A large amount of research has been done for detecting the intruder, e.g. in [1, 18, 43]. Recently some effort has also been put into the decontamination problem (e.g., see [4, 6, 34, 19]). In this thesis we continue this line of research by studying strategies

for cleaning networks contaminated by intruders.

Consider a network environment where nodes represent hosts and edges represent connection between hosts, we assume that the network is initially contaminated by intruders. The intruders are dangerous pieces of software and they can spread arbitrarily fast in the network contaminating the nodes they pass by. Our goal is to deploy a team of agents to decontaminate the network. An agent is a program that can migrate on the network. When an agent resides on some node, it can detect all intruders at the node and clean the node if it is contaminated. However, when a clean node has a contaminated neighbour, it becomes contaminated. We assume that the agents start from the same node (the *homebase*) and can move to neighboring nodes.

Currently, there are several papers about the optimal strategy to clean a network (e.g., see [4, 6, 33, 34, 21, 19, 20]). The aim is always to minimize the number of agents to lead the graph to a state in which all nodes are simultaneously decontaminated. In this thesis we assume that the number of mobile agents is fixed in advance and is not necessarily sufficient to decontaminate the whole network. If the agents are not sufficient to decontaminate the whole network, only a portion of the nodes can be cleaned and all the agents have to guard the boundary of the clean area to protect it from the contaminated area. In this thesis we want to study what the maximum number of nodes that can be decontaminated with a given number of agents is(*Best Effort Decontamination*). We consider trees and meshes; we design optimal strategy to decontaminate the biggest number of nodes in those networks; and we analyze the number of moves as well as the

cost of time units for the mobile agents during the decontamination process.

1.2 Terminology

The network:

The network is represented by a simple undirected graph $G = (V, E)$ with vertices representing the processing elements and edges representing the communication links between them. Let $E(u)$ denote the set of edges incident to node $u \in V$, and let $\lambda_u : E(u) \rightarrow \mathcal{L}$ be an injective function that associates to each incident edge a distinct label, sometimes called *port number*, from a set of labels $\mathcal{L}$. Note that for each edge $e = (u, v)$ there are two associated labels, $\lambda_u(e)$ and $\lambda_v(e)$, which are possibly different. The set $\lambda = \{\lambda_u : u \in V\}$ constitutes the labeling of G, and by (G, λ) we shall denote the corresponding edge-labeled graph. Nodes are labeled with distinct Ids.

Mobile Agents:

Operating in (G, λ) is a team of autonomous and identical mobile agents. The agents can move from a node to a neighboring node in G, have computing capabilities and are bounded computational storage, and obey the same set of behavioral rules (the *protocol*). Initially, a fixed number of agents is located in the same node, called *homebase*, which is *guarded*, and all the other nodes are *contaminated*. Agents have local memory ($log\ n$ bits suffice for all our strategies), distinct Ids, and they know the topology of the network.

In the literature, agents communicate with each other in different ways, including:

- *Face to face*: agents can communicate when they reside on the same node.

- *Whiteboard*: this is an indirect communication mechanism. Each node contains a whiteboard (a limited storage area). When the agent arrives at a node, it can write on and read information from the whiteboard.

In this thesis, we mainly use the first communication method.

Timing Assumption :

Local computation is considered instantaneous. Referring to the mobile agent's movement, there are two models in literature: *synchronous* and *asynchronous*. In the synchronous model, agents have synchronized clocks. It costs one time unit to execute one movement for each agent. In the asynchronous model, there is no uniform clock and the time taken by the agent's movement is unpredictable. In this thesis, we will use the asynchronous model.

Knowledge of The Agents:

Regarding the knowledge of the agents, we consider the following two variations:

- *Local Model.* In this model, an agent knows only the state of the node at which it is located and does not know the state of any other nodes.

- *Visibility Model.* In this model, each agent has the ability to "see" the state of its neighbouring nodes. This means that an agent can detect if there are other agents on its neighboring nodes, and whether the neighboring nodes are contaminated or clean. Notice that this capability could be achieved if the agents have communication power and send a message (e.g., a single bit) to their neighbouring nodes after cleaning a node or guarding

a node.

Decontamination process:

At any time, each node in the network can be in one of three possible states: *clean, guarded or contaminated.* When at least one agent resides on the node, the node is said to be guarded. When a node is cleaned by agents and all its neighboring nodes are clean or guarded, the node is clean. When a node has a contaminated neighbour, it becomes contaminated. Initially all nodes are contaminated except for one, which is guarded by all the available agents.

Complexity Measures : In this thesis, after we develop the decontamination strategy, we will analyze the efficiency of the strategy. Efficiency will be measured by three parameters: the *number of agents* deployed, the *number of moves* executed by the agents and the *time units* elapsed to finish the decontamination. Since our model is asynchronous, we can not really measure the time complexity. In this thesis, we will then use *ideal time units*; that is only by the purpose of time complexity, we assume that each movement by mobile agent costs one time unit.

1.3 Related Work

Graph Search. The decontamination problem considered in this thesis is a variation of a well known and extensively studied problem known as *graph-search.* The graph-

search problem was first introduced in [8], [39], [40]. In these papers, the graph-search problem has been defined as follows: given a contaminated connected network, via a sequence of operation by using searchers, achieve a state in which all nodes and edges are simultaneously decontaminated. There are two versions of the graph-search problem: edge-searching problem and node-searching problem. In node searching, the allowable moves are: (1) placing a searcher on a node and (2) removing a searcher from a node. A contaminated edge is cleared if both of its endpoints contain searchers. In edge searching, one more move is allowed, (3) moving a searcher along an edge. In edge searching, a contaminated edge is cleared by moving a searcher along this edge. The decision problem corresponding to the computation of the search number of a graph is NP-hard [35] with NP-completeness being shown in [7, 32]. The minimum number of searcher is related to several graph measures, for example, intervalwidth, cutwidth, pathwidth and treewidth. For example, paper [27] shows that the minimum searcher number $s(G)$ of a network equal to the intervalwidth; in paper [28], it shows that $s(G)$ is equal to the pathwidth of the graph plus one; in paper [17], the number of $s(G)$ is between the vertex separation $vs(G)$ and $vs(G) + 2$. Both node search and edge search problem have been studied in many papers; for example, in bipartite graphs [31], planar graph with degree no more than three [28], cobipartite graphs [10], starlike graphs [23]. Some papers also give some linear-time algorithms for specific network topologies, like [36], [45], [44].

In those papers about node searching and edge searching, the agents can be arbitrarily placed and removed from any nodes in the network, in other words, the agents are allowed

to jump. In real network environments, the "jump" is not realistic. To overcome the requirement of allowing agents to jump, another model has been defined in [4]. In this paper, a contiguous, monotone decontamination strategy is defined as follows: (1) the agents cannot be removed from the network, (2) at any time of the search strategy, the set of clean nodes forms a connected subnetwork and (3) a clean node cannot be recontaminated. The continuous assumption changed the nature of the problem of graph search. In paper [5], the authors prove that the searching number in continuous model is equal to or greater than the search number of the original no-continuous model and this problem is NP-hard.

Contiguous Monotone Graph Search. Some specific topologies have been studied in this contiguous and monotone model. For example, optimal strategies have been studied in trees [4], meshes [21], hypercubes [19] and chordal ring [20]. Paper [4] develops a linear-time algorithm to capture the intruder in tree network and the movement sequences of mobile agents can be computed within $O(n)$ time units and coded in $O(n)$ space. This paper also shows that the tree that requires the maximum number of searchers needs $\log_2 n$ searchers to be cleared. For capturing an intruder in a $m \times n$ mesh ($m \leq n$), paper [21] shows that at least m searchers are required. The total number of movements for the mobile agents is $O(mn)$ and the total time units is O(n). For capturing an intruder in a chordal ring, paper [20] shows that the $s(G)$ depends only on the length of the longest chord and it is equal to $2 \times d_k$ in the local model and $2 \times d_k + 1$ in the visibility model. For the hypercube, paper [19] shows that $\Theta(\frac{n}{\sqrt{\log n}})$ searchers are required in the local

model while $\frac{n}{2}$ searchers are employed in the visibility model; it is open to show that $\frac{n}{2}$ is also a lower bound in that model.

In this thesis, we will use the continuous and monotone model to solve the decontamination problem for mesh and tree with fixed number of mobile agents.

Tree Decontamination [4].

In the process to develop our decontamination algorithm in chapter 4 of this thesis, we use some notions from paper [4]. So, we here review some important results from [4]. The tree has been the first topology to be investigated in the Local Model. For a given tree T, the minimum number of agent needed depends on the location of homebase (the node from which the team of agents start). Consider as an example, the tree shown in Figure 1.1. If the homebase is at the node A then it is easy to see that two agents can decontaminate the whole tree. However, at least three agent are needed if the homebase is at the node B.

In [4], the authors describe a simple and efficient strategy to determine the minimum number of agents necessary to decontaminate an arbitrary given tree from any initial starting node. The strategy is based on the following two observations.

Consider a node A, if A is not the homebase, the agents will arrive at A for the first time from some link e (see Figure 1.2). Let $T_1(A), \ldots, T_i(A), \ldots, T_{d(A)-1}$ be the subtrees of A from the other incident links, where $d(A)$ denotes the degree of A; let m_i be the number of agents needed to decontaminate $T_i(A)$ once the agents are at A, and let $m_i \geq m_{i+1}$, $1 \leq i \leq d(A) - 2$. The first observation is that to decontaminate A and all its other

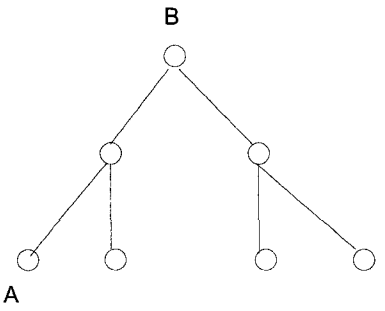

Figure 1.1: Different homebase require different number of agents

subtrees without recontamination, the minimum number $m(A)$ of agents needed is

$$m(A) = m_1 \text{ if } m_1 > m_2 \text{ and}$$

$$m(A) = m_1 + 1 \text{ if } m_1 = m_2$$

Consider now a homebase B and let $m_j(B)$ be the minimum number of agents needed to decontaminate the subtree $T_j(B)$, and let $m_j \geq m_{j+1}$, $1 \leq j \leq d(B)$. The second observation is that to decontaminate the entire tree starting from B the minimum number $m(B)$ of agents needed is

$$m(B) = m_1 \text{ if } m_1 > m_2 \text{ and}$$

$$m(B) = m_1 + 1 \text{ if } m_1 = m_2$$

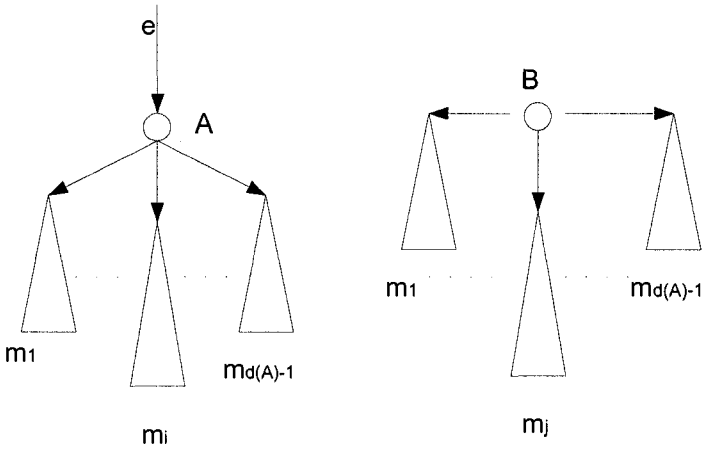

Figure 1.2: Determining the minimum number of cleaners

Based on these two observations, firstly the authors show how the determination of the optimal number of agents can be done through a saturation where simple information about the structure of the tree are collected from the leaves and propagated along the tree, until the optimal is known for each possible starting point. The most interesting aspects of this strategy is that it yields immediately a protocol for trees that uses exactly that minimum number of agents. The technique to determine the minimum number of agents and the corresponding decontamination strategy is done in $O(n)$ time and exchanges $O(n)$ messages. Secondly, the algorithm is naturally distributed, the minimum number of

agents and the search strategy can be computed in a decentralized manner.

The trees that require the largest number of agents are *complete binary trees*, where the number of agent is $O(\log n)$; by contrast, in the *line* two agents are sufficient.

1.4 Our Results

In this thesis we consider for the first time the problem of decontaminating as many nodes as possible in a network with a fixed number of agents . We study in details the cases of the Meshes (regular, hexagonal, octagonal), and of the Trees (general, complete).

In the case of the Meshes (Chapters 2 and 3) we show that the maximum number of nodes that can be decontaminated with a given number of agents depends on the location of the homebase. We consider three different types of meshes: the regular, the hexagonal, and the octagonal mesh. Moreover, for each type of mesh, three possible homebase locations: a corner, a border, a node inside the mesh are studied. For each case the maximum area that can be decontaminated is identified and the strategies to perform the decontamination are described. We then analyze their move and time complexities.

We then consider in Chapter 4 the Tree. In this case, the number of nodes that can be cleaned with a given number of agents depends on the location of the homebase. We first design a distributed algorithm for an arbitrary tree that determines, for a given homebase and a given number of available agents, the maximum area that can be decontaminated. After this algorithm is executed, the team of agents can follow the optimal strategy start-

ing from the homebase. We analyze the message complexity of the distributed algorithm as well as the move and time complexity of the cleaning strategy. We then concentrate on full binary trees and i-trees and design for these particular structures a simpler decontamination algorithm.

Finally, Chapter 5 describes some open problems and future directions.

Chapter 2

Disinfecting a Mesh

A $m \times n$ mesh is a graph $M_{m,n}$ = (V,E) with $V_{i,j} \in V, 0 \leq i \leq m-1, 0 \leq j \leq n-1$ and E contains exactly the edges ($V_{i,j}$, $V_{i+1,j}$), $i \neq m-1$ and ($V_{i,j}$, $V_{i,j+1}$), $j \neq n-1$. In a mesh each internal vertex $V_{i,j}$ with $0 < i < m-1$, $0 < j < n-1$ is connected to the four vertices $V_{i-1,j}$, $V_{i+1,j}$, $V_{i,j-1}$ and $V_{i,j+1}$. The four vertices $V_{0,0}$, $V_{0,n-1}$, $V_{m-1,0}$, $V_{m-1,n-1}$ locate at the corners of the mesh, each of them has only two connected neighbours and each vertex on the border of the mesh has three connected neighbours. We assume there is a compass sense of direction. For node $V_{x,y}$, its north neighbour is $V_{x-1,y}$; its south neighbour is $V_{x+1,y}$; its east neighbour is $V_{x,y-1}$; its west neighbour is $V_{x,y+1}$.

In [21] it has been shown that $Min(m,n)$ agents are necessary to clean the whole network if the agents are able to see the neighboring nodes' state ($Min(m,n)+1$ agents are needed if the agents do not have such capability). We now assume that k ($k < Min(m,n)$) agents are available. Since we cannot use k agents to clean the whole mesh, we can clean

a part of it. The biggest clean area must be in the state that no agents on the border of the clean part can move anymore without recontamination. In this chapter we will calculate the biggest area the agents can decontaminate and we will show that its size depends on the starting location of the homebase.

Before we describe our results, we define the notion of *Rigid Frontier* that will be used when analyzing our decontamination strategies for all the different Meshes considered in the thesis.

Definition 1. Rigid Frontier

Let X be a Vertex Cut that divides V in $A \subset V$ and $B \subset V$ ($V = A \cup X \cup B$). Let the vertices of A be contaminated, the vertices of B be clean (and not guarded), the vertices of X be guarded. The set X is called a Rigid Frontier *if $\forall x \in X$, $\exists$ at least two vertices $w, z \in N(x)$ such that $w, z \in A$.*

In other word, the nodes of a rigid frontier divide the graph in two areas and the agents which guard those nodes cannot move without causing recontamination. It is obvious that the biggest clean area must be surrounded by a rigid frontier.

Let "diagonal segment " refer to a set of nodes forming a diagonal in the mesh like $[V_{i,j}, V_{i\pm1,j\pm1}, ..., V_{i\pm h,j\pm h}]$, "vertical segment " refer to contiguous nodes on a column like $[V_{i,j}, V_{i\pm1,j}, ..., V_{i\pm h,j}]$, and "horizontal segment " refer to contiguous nodes in a row like $[V_{i,j}, V_{i,j\pm1}, ..., V_{i,j\pm h}]$. We now make the following observation about the "shape" of a rigid frontier.

Lemma 1. *In a mesh, a rigid frontier can not be composed by horizontal/vertical segments containing three nodes or more.*

PROOF

Assume that there is a vertical segment $L = [V_{i,j}, V_{i+1,j}, ..., V_{i+h,j}]$ with at least three nodes. By definition of rigid frontier, any node of L has at least two contaminated neighbours and a clean neighbour. Consider node $V_{i+1,j}$, node $V_{i+1,j}$ has 4 neighbours, two of which are guarded. By definition $V_{i+1,j}$ must have at least a clean (non guarded) neighbour and at least two contaminated neighbours. This is impossible, because $V_{i+1,j}$ has only two unguarded neighbours. The proof for horizontal segments is analogous. ∎

As a consequence of the previous Lemma, a rigid frontier in a mesh could be formed by: horizontal/vertical segments containing at most two nodes, and diagonal segments.

2.1 Decontamination Model

For all Mesh topologies (in this Chapter and in the next) we assume that the agents have the weakest method of communication: they can exchange information only when they reside on the same node (face-to-face). Moreover, we consider the two models defined in the previous chapter: Local Model and Visibility Model. In both models a strategy is composed by two phases. The first phase is the *Deployment Phase*, during which the agents move from the homebase to appropriate starting locations. The second phase is the *Decontamination Phase*, during which the agents perform the decontamination.

In the Local Model, the agents have only local knowledge and do not know the states of their neighboring nodes. In this case, an agent is used as the coordinator to arrange other agents' movement.

In the Visibility Model, each agent can "see" if there are other agents on its neighboring nodes and it can move independently. In the Visibility Model the decontamination phase is extremely simple; in fact, after the agents are appropriately deployed, they all follow the algorithm below:

Algorithm 2.1 CLEAN WITH VISIBILITY (for agent a)

```
If a has only one contaminated neighbouring node X
    move to X
else
    do nothing
```

The above algorithm can be applied to any topology.

Lemma 2. *Algorithm 2.1 performs a contiguous and monotone decontamination in any graph.*

PROOF

Let us assume there is an agent a on a node Y. According to Algorithm 2.1, if a finds there is only one contaminated neighbouring node X, it will move to X. While moving no recontamination will occur , because Y's all neighbouring nodes except for X

are clean or guarded. Moreover, after the movement, a new node will be decontaminated. So Algorithm 2.1 correctly performs a contiguous and monotone decontamination. ∎

2.2 Homebase on a Corner

When the homebase is located at one of the four corners of the mesh, we can calculate the maximum number of nodes that the agents can decontaminate.

Theorem 1. *Let k be the number of available agents, no deterministic algorithm can clean more than $k(k+1)/2$ nodes.*

PROOF

Let us assume that the home base is located at the upper left corner of the mesh, and k agents execute a monotone and contiguous decontamination. At some point, we obtain a state when the maximum number of nodes have been cleaned and no agents can move. In this final state, all agents must form a rigid frontier. Let us assume the column furthest to the right in the clean area is column i and the lowest row in the clean area is row j. Then from column 0 to column i, at least one agent should guard one node on each column and from row 0 to row j at least one agent should guard one node on each row otherwise some nodes would be recontaminated. Let us consider the connection among the agents in the frontier. According to Lemma 1, we know that in the frontier there are no vertical or horizontal segments with more than two nodes, so there are three cases:

1) the connection is a single diagonal (as in Figure 2.1 (a)); 2) the connection contains

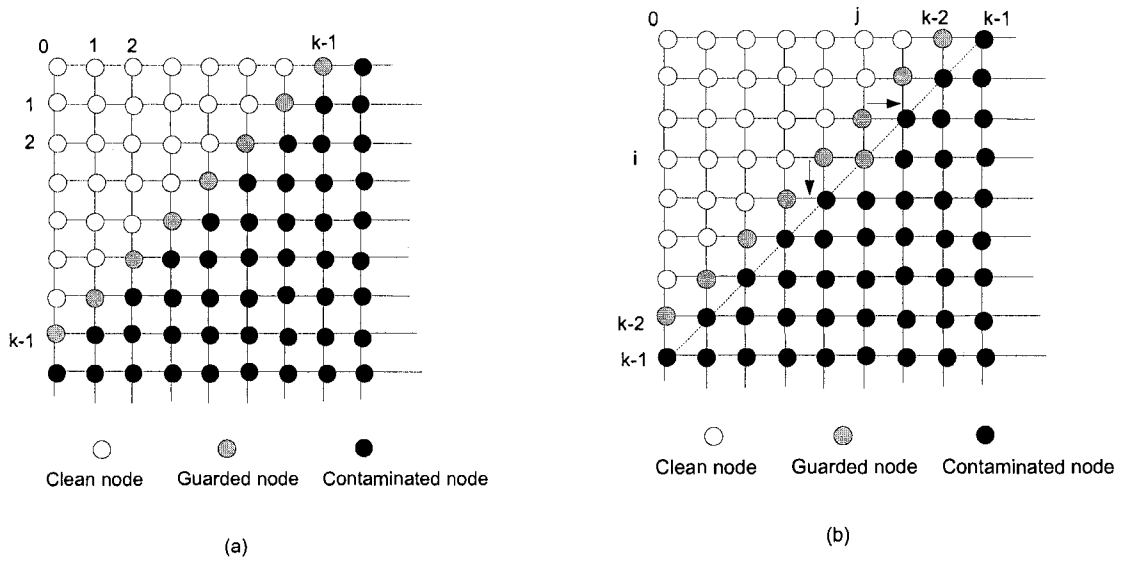

Figure 2.1: Biggest area cleaned with k agents

at least a vertical segment connected to an horizontal segment of length 2 (i.e, as in Figure 2.1 (b)); 3) the connection is made of several diagonals (as in Figure 2.2, and in Figure 2.3). 1) In the first case, the clean area is necessarily a triangle. The first column and the first row form the right triangle side and the connection of agents forms the hypotenuse. On the diagonal, there is only one agent on each column and each row. Since we have k agents available, the hypotenuse contains k agents and the total number of clean nodes is then $\sum_{i=0}^{k-1} i = \frac{k(k+1)}{2}$. 2) Let the connection of the agents contains some vertical or horizontal segments of length two. In this case a fragment of the frontier must be in the

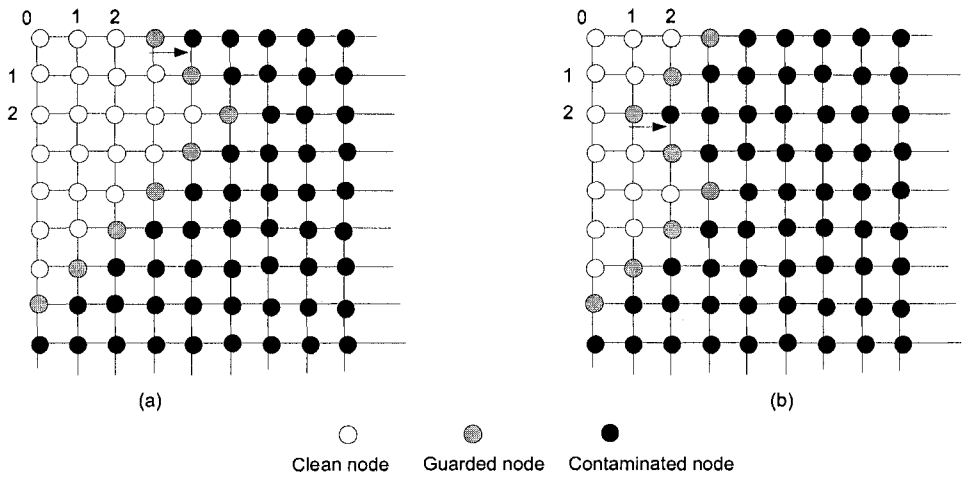

Figure 2.2: Border with several diagonals

form $V_{i,j}, V_{i-1,j}, V_{i,j-1}$ $(i, j > 0)$ (an example is in Figure 2.1 (b)). This however violates the definition of rigid frontier because the agent on $V_{i-1,j}$ could move to $V_{i-1,j+1}$ without causing recontamination. 3) More diagonals in most cases mean a contradiction to the definition of rigid frontier (see Figure 2.2 (a) where the agent on the first row could move to east, or Figure 2.2 (b) where the agent on the intersection of two diagonals could move to east). The only case that does not violate the definition of rigid frontier occurs when one (or more than one) enclosed area connects to the corner of the mesh with another diagonal (or diagonals), like the example of Figure 2.3. Obviously, the number of nodes decontaminated in this case is smaller than the case of a single diagonal. ∎

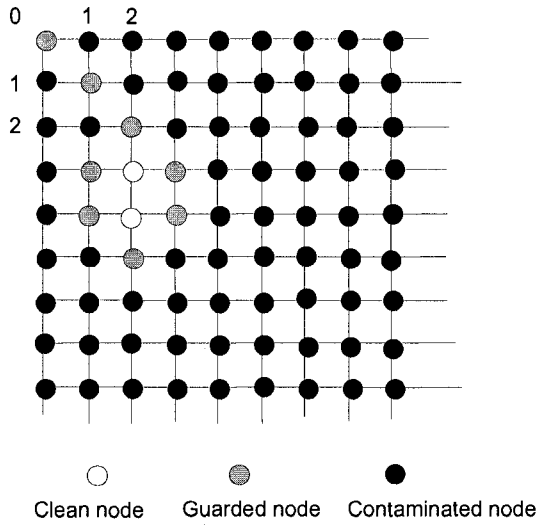

Figure 2.3: An enclosed area connected with corner

2.2.1 Cleaning without Visibility

Decontamination strategy

The idea of this strategy is to use a coordinator to move the other agents to form a connected sub-network and let the agents and the border of the mesh to besiege the border of the clean area from the contaminated network whenever moving occurs.

In the deployment phase, the agents are moved to occupy continuous positions on the first row. After that, the coordinator will move east along the first row. Every time the

coordinator meets an agent, it will let the agent move to the south neighboring node on the second row. The coordinator will let all agents move south except for the last one to guard the end of the continuous positions on the first row. Then the coordinator will move along the second row and let all agents move south to the third row except for the last one. This process is repeated until the coordinator reaches a row with only one agent. In this stage, the diagonal is formed. Then the coordinator can move to a neighbouring node of the agent on the first column. The description of the algorithm is provided below.

Algorithm 2.2 Disinfecting a $m \times n$ mesh with k-1 agents from home base $V_{0,0}$ with one coordinator.

Deployment Phase:

Start with agents in node $V_{0,0}$. One agent is labeled as coordinator and other agents are labeled from S_0 to S_{k-2};
For each agent S_i
    Agent S_i moves to west;
    When Agent S_i arrives node $V_{0,j}$ $(j < i)$
    If there is other agent on this node
        Agent S_i moves to west;
    else
        Agent S_i waits until S_j arrives at $V_{0,j}$;

Decontamination Phase:

for $i = 0$ to $k - 3$
    j=0;
    while $j < (k - i - 2)$
        The coordinator moves to $V_{i,j}$;
        When the coordinator meets an agent, the coordinator makes the agent move to $V_{i+1,j}$.
        j+=1;
    i+=1;
    j-=1;
    the coordinator moves to $V_{i,j}$;
    while $j > 0$
        j-=1;
        The coordinator moves to $V_{i,j}$;
        When the coordinator meets an agent, the coordinator makes the agent move to $V_{i+1,j}$.
    i+=1;
The coordinator moves to node $V_{k-2,0}$;
The coordinator lets the agent at node $V_{k-2,0}$ move to $V_{k-2,1}$;

The coordinator moves to node $V_{k-1,0}$;

Correctness and Complexity

Here we want to prove that this decontamination strategy performs a contiguous and monotone decontamination.

Before we start to prove this theorem, let us define "a step" to be one movement of an agent.

Theorem 2. *Algorithm 2.2 performs a contiguous and monotone decontamination.*

PROOF

Base Case: The deployment phase is contiguous and monotone.

In Algorithm 2.2, initially all agents are located at node $V_{0,0}$. There are $k-1$ agents that move along the first row and each agent will occupy a node. After this process, every node from $V_{0,0}$ to $V_{0,k-2}$ will be occupied by one agent. So the nodes from $V_{0,0}$ to $V_{0,k-2}$ on the first row are clean, and the clean process on the first row is contiguous and monotone.

Assumption

Let us assume that up to step $i(i > 0)$, the cleaning strategy is contiguous and monotone. At this step, let us assume the biggest row number in the clean area is j. Since the clean process has been contiguous and monotone, the nodes $V_{0,k-2}$, $V_{1,k-3}$, ...,$V_{j-1,k-j-1}$ on the diagonal must be clean and guarded by agents. Besides those agents on the diagonal, the remaining agents may guard row r_j $(0 < j < m)$ or two consecutive

rows r_j and r_{j+1} $(0 < j < m-1)$.This diagonal and the row r_j or rows r_j and r_{j+1} surround the clean part.

Induction

At step $i+1$, the coordinator will let one agent move to its neighbouring south node. If in step i, the clean area is surrounded with the diagonal and the row r_j, in step $i+1$ the coordinator will let the agent located at node $V_{j,0}$ or node $V_{j,k-j-3}$ move to its south neighbouring node. If in step i, the clean area is surrounded with the diagonal and the rows r_j r_{j+1}, in step $i+1$ the coordinator will let the terminal node on row r_j move to its south neighbouring node. If the coordinator moves to east on row r_j, the terminal node is located on the left end of row r_j; if the coordinator moves to west on row r_j, the terminal node is located on the right end of row r_j. In both cases, the node from which the agent has moved during step $i+1$ is now surrounded by clean or guarded nodes and thus no recontamination occurs. Moreover, the clean area adds with one new node which is guarded by the moving agent. ∎

Thus, we have proved that the decontamination strategy is contiguous and monotone.

Theorem 3. *The number of nodes decontaminated by Algorithm 2.2 is* $k(k-1)/2+2$.

PROOF

In Algorithm 2.2, before the coordinator let the agent located at node $V_{k-2,0}$ move to node $V_{k-2,1}$, the clean area is a triangle with $k-1$ nodes as bottom and $k-1$ nodes as height. So the total number of clean nodes in this triangle is $k(k-1)/2$. After that, the

coordinator let the agent located at node $V_{k-2,0}$ move to node $V_{k-2,1}$ and the coordinator itself moves to $V_{k-1,0}$. This step adds two more clean nodes than the triangle, so the total clean nodes will be $k(k-1)/2+2$. ∎

Notice that the number of decontaminated nodes in Algorithm 2.2 is less than the maximum number of nodes that can be decontaminated in Theorem 1. This is because in Algorithm 2.2 we have to use one agent as coordinator and use $k-1$ agents to surround a triangle.

Now let us compute the agents' moves and time units consumed in this algorithm.

Theorem 4. *The total number of moves by all agents in Algorithm 2.2 is* $(k-2)(k-1)+k(k-1)/2-\lfloor k/2 \rfloor+1$.

PROOF

In the process to occupy the nodes from $V_{0,0}$ to $V_{0,k-2}$ on the first row, agent S_i ($i \leq k-2$) performs i moves. So, the total number of moves in this process is $(k-2)(k-1)/2$. In the process to move south to form the biggest triangle area, agent S_i ($i \leq k-2$) performs $k-2-i$ moves. So, the total number of moves in this process is $(k-2)(k-1)/2$. At last, the agent on node $V_{k-2,0}$ executes an extra move. Therefore, the total number of moves performed by the searching agents is $(k-2)(k-1)+1$.

The number of moves performed by the coordinator is calculated as follows: the coordinator moves across every node on the clean area except for the nodes on the hypotenuse $V_{0,k-1}$, $V_{2,k-3}$,$V_{4,k-5}$,...,$V_{k-3,1}$. The number of these moves is $k(k-1)/2-\lfloor k/2 \rfloor-1$. At

last, the coordinator moves from $V_{k-2,0}$ to $V_{k-1,0}$. So the total number of moves performed by the coordinator is $k(k-1)/2 - \lfloor k/2 \rfloor$.

Therefore, the total number of moves for the node-search on a Mesh with k agents in the Local Model is $(k-2)(k-1) + k(k-1)/2 - \lfloor k/2 \rfloor + 1$. If k is even, the total number of moves is $(k-2)(k-1) + k(k-1)/2 - k/2 + 1$; if k is odd, the total number of moves is $(k-2)(k-1) + k(k-1)/2 - (k-1)/2 + 1$. ∎

Theorem 5. *The ideal time complexity in Algorithm 2.2 is* $k(k-1)/2 - \lfloor k/2 \rfloor + k - 2$.

PROOF

In the process to occupy the nodes from $V_{0,0}$ to $V_{0,k-2}$ on the first row, all agents move east to occupy their starting place, this process consumes $k-2$ time units. After that, the coordinator visits all nodes in the clean area except for those nodes on the hypotenuse $V_{0,k-1}$, $V_{2,k-3}$,$V_{4,k-5}$,...,$V_{k-3,1}$. While the coordinator traverses the clean area, it makes the agents move to the south direction. So, the coordinator consumes $k(k-1)/2 - \lfloor k/2 \rfloor - 1$ time units to move. At last, the coordinator moves from $V_{k-2,0}$ to $V_{k-1,0}$, with one more time unit. So the total number of time units consumed by the coordinator is $k(k-1)/2 - \lfloor k/2 \rfloor$.

Therefore, the total number of time units for the node-search on a Mesh with k agents in the Local Model is $k(k-1)/2 - \lfloor k/2 \rfloor + k - 2$. When k is even, the total number of time units is $k(k-1)/2 - k/2 + k - 2$; if k is odd, the total number of time units is $k(k-1)/2 - (k-1)/2 + k - 2$. ∎

2.2.2 Cleaning with Visibility

Decontamination strategy

We assume all agents are located at $V_{0,0}$ at first. During the deployment phase, all agents move to occupy continuous positions on the first row. After the deployment phase, each agent will execute Algorithm 2.1.

This procedure will end when no more agents can move. At last, the biggest number of nodes has been decontaminated.

The formal description of the algorithm is provided in Algorithm 2.3.

Algorithm 2.3 Disinfecting a $m \times n$ mesh with k agents from home base $V_{0,0}$ with Visibility.

Deployment Phase:

Start with agents on node $V_{0,0}$.
All agents are labeled from S_0 to S_{k-1};
For each agent S_i
    Agent S_i moves to west;
    When Agent S_i arrives node $V_{0,j}$ $(j < i)$
    If there is other agent on this node
        Agent S_i moves to west;
    else
        Agent S_i waits until S_j arrives $V_{0,j}$;

Decontamination Phase:

Algorithm 2.1

Correctness and Complexity

Theorem 6. *Algorithm 2.3 performs a contiguous and monotone decontamination.*

PROOF

In the deployment phase, agent S_i moves west until arriving at node $V_{0,i}$. While moving to $V_{0,i}$, S_i at node $V_{0,j}$ $(j < i)$ checks if there are other agents on it. If no agent is on this node, it waits until S_j arrives at this node. So, this process is contiguous and monotone. From lemma 2, we know that the decontamination phase is contiguous and monotone too. By now, we have proved Algorithm 2.3 performs a contiguous and monotone decontamination. ∎

Theorem 7. *The number of nodes decontaminated by Algorithm 2.3 is* $k(k+1)/2$.

PROOF

In the final state of Algorithm 2.3, the clean area is a triangle with k nodes as bottom and k nodes as height. So the total number of clean nodes in this triangle is $k(k+1)/2$, which is the biggest number of nodes we can clean with k agents proved by Theorem 1. ∎

Notice that this number of decontaminated nodes in Algorithm 2.3 is equal to the maximum number of clean nodes we can clean with k agents proved by Theorem 1. So this strategy is optimal.

Theorem 8. *The total number of moves performed by all agents in Algorithm 2.3 is* $k(k-1)$.

PROOF

The number of moves performed by the agents is calculated as following:

In the process to occupy the nodes from $V_{0,0}$ to $V_{0,k-1}$ on the first row, the k agents performs $1+2+...+k-1$ moves, so the total number of moves in this process is $k(k-1)/2$. In the process to move south to form the biggest triangle area, the agent on node $V_{i,j}$ performs $k-1-i$ moves. So, the total number of moves in this process is $k(k-1)/2$. Therefore, the total number of moves performed by k agents is $k(k-1)$. ∎

Theorem 9. *The ideal time complexity of Algorithm 2.3 is* $2 \times (k-1)$.

PROOF

Since the agent can detect whether there are other agents on its neighboring nodes instantly, the time consumption can be calculated as following:

In Algorithm 2.3, at time unit 1 when the agents arrive at node $V_{0,1}$, the agent guarding node $V_{0,0}$ can move south to node $V_{1,0}$. At time unit two, some agents will arrive at node $V_{0,2}$ and one agent will arrive and guard node $V_{1,0}$ as in Figure 2.4. To occupy the nodes from node $V_{0,0}$ to node $V_{0,k-1}$ on the first row , $k-1$ time units are required . Some agents move south at the same time to occupy the first row. It costs two time units for every agent to move to the south neighbouring node. One time unit is spent while waiting for the east neighbouring node to be guarded and one time unit is spent to move to the south neighbouring node. The last move is executed by the agent on the first column, which moves from node $V_{k-2,0}$ to node $V_{k-1,0}$. For this agent, it takes $2 \times (k-1)$ time units.

Therefore, the total number of time units for the node-search on a Mesh with k agents with visibility model is $2 \times (k-1)$. ∎

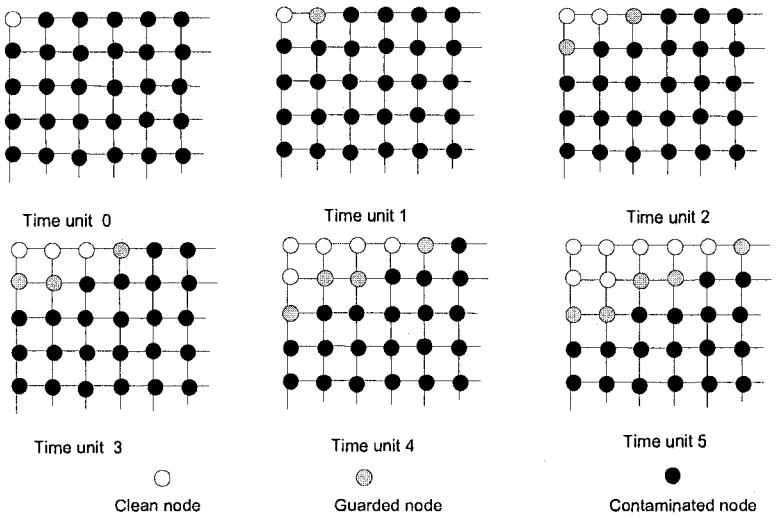

Figure 2.4: Time cost in Algorithm 2.3

2.3 Homebase on a Border

Now let us consider the situation when the home base is located at one node on the first column, and the distance between the homebase and the first node on the first column and that between the homebase and the last node on the first column are both bigger than k.

Theorem 10. *Let k be the number of available agents, no deterministic algorithm can clean more than $(k+1)^2/4$ nodes.*

PROOF

Let the homebase be located at node $V_{i,0}$ on the first column, with $i > k$. At any time of the contiguous decontamination strategy, the agents should cover the "perimeter" of the clean area and, when the highest possible number of nodes has been decontaminated, they must form a rigid frontier. For each column in the clean area, if the column contains more than one node, there are at least two agents to guard the two ends of this column to avoid re-contamination. Moreover, there must be at least one agent on each row to guard one end of the clean segment of the row from contamination (the border of the mesh guards the other end).

There are two cases: Case 1) The clean area does not contain rows or columns which have only one node. In this case the biggest possible clean area has no more than $\lceil k/2 \rceil$ adjoining columns and k adjoining rows to form a rectangle as in Figure 2.5 (a); in other words, it must be contained in that rectangle. Case 2) The clean area contains some rows or columns which have only one guarded node. In this case, the biggest possible area would be contained in an area shaped like in Figure 2.5 (b).

Let us now consider $Case1$: the clean area does not contain rows or columns which have only one node. Here we want to prove that the biggest area decontaminated by k agents is composed by two triangles like in Figure 2.7 (where the cases k even and k is odd are shown).

First, let us consider the case when k is even. Let row i and row $i + 1$ divide the possible clean area into two equal parts (Figure 2.5 (a)). There are two ways to arrange

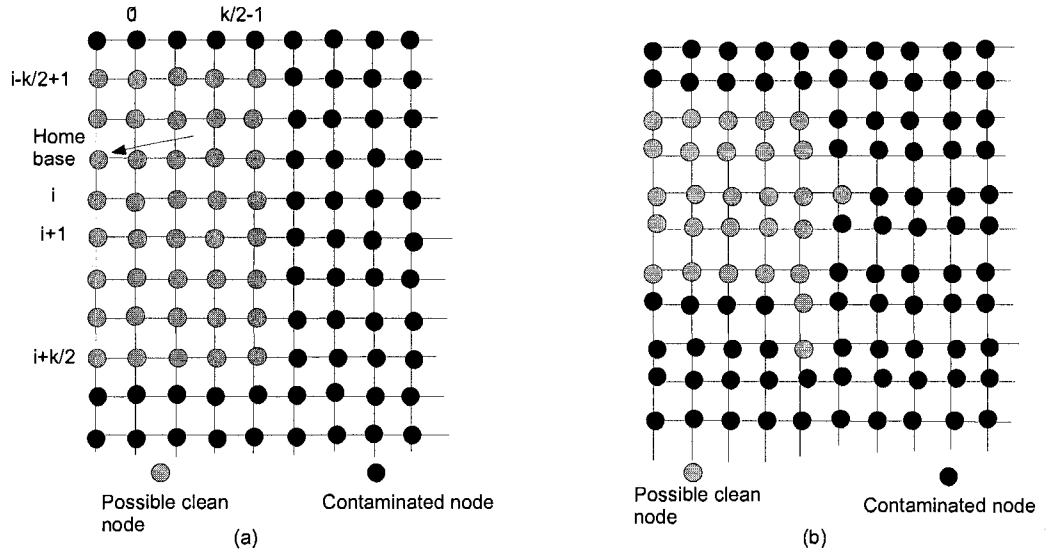

Figure 2.5: Possible clean area

the agents in the two parts. The first is using $k/2$ agents to clean the upper part and $k/2$ to clean the lower part, the second is to use a different number of agents in each part. In the first case the biggest area we can hope to clean in each part is a triangle; in fact, even assuming that the lower part is completely clean, the maximum area we can clean in the upper part is a triangle by Theorem 1; same reasoning holds for the lower part. Each of the triangles holds $\sum_{i=1}^{k/2} i$ nodes.

We now demonstrate that if we use more than $k/2$ agents $(k/2+j)$ on one part and less

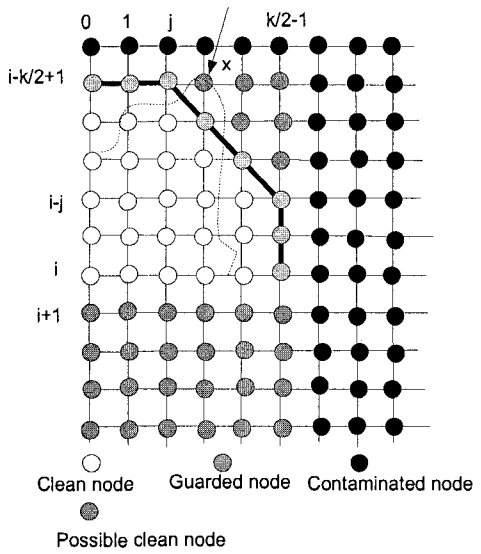

Figure 2.6: Biggest clean area with k/2+j agents in upper part

than $k/2$ agents $(k/2-j)$ on the other part, the size of the maximum possible clean area would be smaller. In the lower part, with $k/2-j$ agents, regardless of the distribution of the agents in the upper part, no more than $\sum_{i=1}^{k/2-j} i$ nodes could be cleaned (it follows from Theorem 1). Now we want to see what would be the biggest area that can be cleaned with $k/2+j$ agents on the upper part. It is obvious that the lower part cannot be completely clean, but for calculating the biggest number of nodes that can be decontaminated in the upper part, We assume that the lower part is all clean (i.e. assuming that row $i+1$ is all guarded). By definition, the upper part has size $k/2 \times k/2$. Consider the following shape S in the possible clean area: a triangle obtained by a diagonal with $k/2$ nodes,

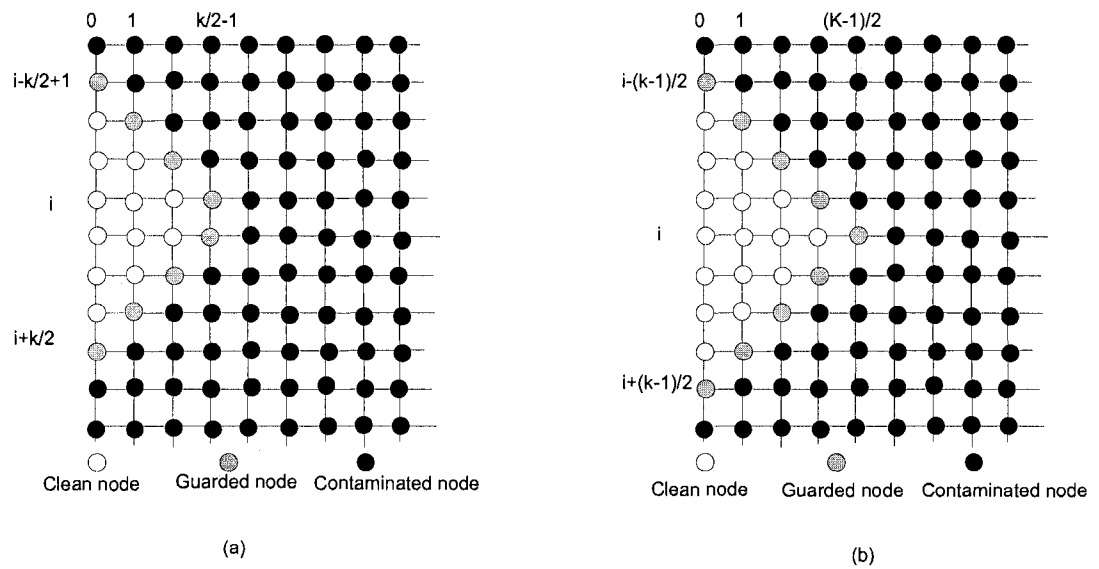

Figure 2.7: Clean area with an even (left) and odd (right) number of agents

enlarged by including the next j diagonal lines in the $k/2 \times k/2$ area. Notice that the frontier of S (see Figure 2.6) contains $k/2 + j$ agents, in fact: there are $j + 1$ nodes on row $i - k/2 + 1$ and $j + 1$ on column $k/2 - 1$. Besides, there are $k/2 - j - 2$ nodes in the upper end of column $j + 1$ to column $k/2 - 2$. So the total number of nodes in the frontier on the bold line in Figure 2.6 is $j + 1 + j + 1 + k/2 - j - 2 = k/2 + j$. We now want to prove that S is the maximum area that can be cleaned with a frontier of $k/2 + j$ agents. Let us imagine there is some other clean node outside the besieged area, like node $X_{m,n}$, and another besieged area which includes node $X_{m,n}$ is marked with a

dash line in Figure 2.6. This besieged area should have agents to guard from row m to row i and from column 0 to column n. Since node $X_{m,n}$ is outside the bold line, we have that $(m+n) > (k/2+j)$ and thus the new besieged area needs more than $k/2+j$ agents. This contradicts the fact that we have only $k/2+j$ agents on the upper part. So we have proved that there are no other nodes outside the besieged area within bold line that can be cleaned using $k/2+j$ agents. The number of node in this besieged area is $\sum_{i=1}^{k/2} i + \sum_{i=k/2-j}^{k/2-1} i = \sum_{i=1}^{k/2} i + k^2/8 + j^2/2 + k/4 - k*j/2 - j/2$. Adding this number to the maximum number of clean nodes in the lower part $\sum_{i=1}^{k/2-j} i = k*j/2 - j^2/2 - j/2$, we obtain that the possible biggest clean number is $\sum_{i=1}^{k/2-j} i + \sum_{i=1}^{k/2} i + \sum_{i=k/2-j}^{k/2-1} i = 2\sum_{i=1}^{k/2} i - j$, which is less than the number of nodes we could clean splitting equally the k agents as we did in the first situation. So, if we use more than $k/2$ agents on one part and less than $k/2$ agents on the other, the maximum number of clean nodes will be less than the number of clean nodes by splitting agents evenly to two parts. In conclusion, we have proved that the two triangles as in Figure 2.7 holds the biggest clean area. The highest node on the first column is at position $i-k/2+1$ and the lowest node on the first line is at the position $i+k/2$; the acme of the echelon is at row i and row i+1 on column $k/2-1$. Thus, there are $(k+2)*k/4$ clean nodes.

If k is odd, the proof is similar. The clean area is a triangle whose hemline's length is k and whose height is $(k+1)/2$. The upperest node on the first column is at position $i-(k-1)/2$ and the lowest node on the first line is at the position $i+(k-1)/2$; the acme of the triangle is at row i and column $(k-1)/2$. So the clean nodes are $(k+1)^2/4$.

We now consider *Case 2*: the clean area contains some rows or columns which have only one guarded node (as in Figure 2.5 (b)). Let us call those nodes singletons. If there are i singletons, according to the conclusions we have gotten in the previous part, the biggest number of clean nodes in the area not containing the singletons would be $(k-i+1)^2/4$. The total number of clean nodes would be $(k-i+1)^2/4+i$ which is less than $(k+1)^2/4$. ∎

2.3.1 Cleaning without Visibility

Decontamination strategy

At first, one agent is labeled as coordinator. Half of the remaining agents move to occupy contiguous positions on the north side of the homebase on the first column and concurrently half of the remaining agents move to occupy contiguous positions on the south side of the homebase on the first column. After that, the coordinator will move along the first column and let all agents move to their east neighboring nodes except for the two agents on the two ends of the contiguous positions on the first column. Then, the coordinator will move to the second column. The coordinator will then move along this column to let those agents on the second column move to their east neighboring nodes except for the two agents at the ends of the contiguous positions on this column. This process is repeated until there are only one or two agents to move to the east neighboring nodes in one column.

The formal description of the algorithm is provided in Algorithm 2.4.

Algorithm 2.4 Disinfecting a $m \times n$ mesh with k-1 agents from home base $V_{i,0}$ with one coordinator.

Start with agents in node $V_{i,0}$. One agent is labeled as coordinator;
if k is even then
  All other Agents are labeled as $S_{-\frac{k-2}{2}}$, $S_{-\frac{k-2}{2}+1}$,...,$S_{\frac{k-2}{2}-1}$, $S_{\frac{k-2}{2}}$;
  Agents S_j moves to nodes $V_{i+j,0}$;
  The coordinator moves from $V_{i,0}$ to $V_{i+\frac{k-2}{2}-1,0}$;
  The coordinator moves from $V_{i+\frac{k-2}{2}-1,0}$ to $V_{i-\frac{k-2}{2}+1,0}$;
  When the coordinator meets one agent, the coordinator will let the agent move to east neighboring node;
  The coordinator moves to $V_{i-\frac{k-2}{2}+1,1}$;
  $i = i - \frac{k-2}{2} + 1$;
else
  All other Agents are labeled as $S_{-\frac{k-1}{2}}$, $S_{-\frac{k-1}{2}+1}$,...,$S_{\frac{k-1}{2}-2}$, $S_{\frac{k-1}{2}-1}$;
  Agents S_j moves to nodes $V_{i+j,0}$;
  The coordinator moves from $V_{i,0}$ to $V_{i+\frac{k-1}{2}-2,0}$;
  The coordinator moves from $V_{i+\frac{k-1}{2}-2,0}$ to $V_{i-\frac{k-1}{2}+1,0}$;
  When the coordinator meets one agent, the coordinator will let the agent move to east neighboring node;
  The coordinator moves to $V_{i-\frac{k-1}{2}+1,1}$;
  $i = i - \frac{k-1}{2} + 1$
j=0;
for $x = 1$ to $\lfloor \frac{k-2}{2} \rfloor - 1$
  while $j < (k - x - 4)$)
      j+=1;
      The coordinator moves to $V_{i+j,x}$;
      When the coordinator meets one agent, the coordinator will let the agent move to $V_{i+j,x+1}$.
  x+=1;
  The coordinator moves to $V_{i+j,x}$;
  while $j > x$
      j-=1;
      The coordinator moves to $V_{i+j,x}$;
      When the coordinator meets one agent, the coordinator will let the agent move to $V_{i+j,x+1}$.
  x+=1;
  The coordinator moves to $V_{i+j,x}$;

Correctness and Complexity

Theorem 11. *Algorithm 2.4 performs a contiguous and monotone decontamination*

PROOF

Base Case: The decontamination in deployment phase is contiguous and monotone.

Assumption

Let us assume that the cleaning strategy is contiguous and monotone up to step $i(i > 0)$. At this step, all the clean nodes are connected and all the agents guard the contiguous border of the clean area. Let us assume the coordinator is located at node $V_{p,q}$.

Induction

At step $i + 1$, after the coordinator moves to the neighboring node of $V_{p,q}$, the coordinator will be at node $V_{p+1,q}$ (if it is moving south) or at node $V_{p-1,q}$ (if it is moving north). If it is moving south, the agent on node $V_{p+1,q}$ will move to node $V_{p+1,q+1}$. In this case, node $V_{p,q+1}$ and node $V_{p+2,q}$ are guarded by other agents. After this movement, node $V_{p+1,q+1}$ becomes part of border of the clean area while node $V_{p+1,q}$ is not anymore. The new border is still contiguous and the clean area increases of one new node. If the coordinator moves north, after the movement, node $V_{p-1,q+1}$ becomes part of border of the clean area while node $V_{p-1,q}$ is not anymore. The new border is still contiguous and the clean area increases of one new node. ∎

Theorem 12. *The number of nodes decontaminated by Algorithm 2.4 is* $k^2/4 + 2$ *if* k *is even,* $(k^2 - 1)/4 + 2$ *if* k *is odd,.*

PROOF

In Algorithm 2.4, the clean area is a triangle if k is even or echelon if k is odd. If k is even, the triangle has $k - 1$ nodes as bottom and $(k + 1)/2$ nodes as height. So the total

number of clean nodes are $k^2/4$. In fact, after the biggest triangle forms, the coordinator can move to one node on the border and let the agent on that node move to one uncleaned neighboring node and itself moving to another uncleaned node. So the total number of clean nodes is $k^2/4 + 2$. If k is odd, the echelon will include $(k^2 - 1)/4$, and the total number of clean nodes is $(k^2 - 1)/4 + 2$. ∎

Theorem 13. *The total number of moves by all agents in Algorithm 2.4 is* $(3k^2 - 6k)/4$ *if* k *is even,* $(3k^2 - 6k - 5)/4$ *if* k *is odd.*

PROOF

In the process to occupy the nodes on the first row, agent S_i $(i \leq k - 2)$ performs i moves. If k is even, the total number of moves in this process is $k(k - 2)/4$; if k is odd, the total number of moves in this process is $(k^2 - 2k + 1)/4$. In the process to move east to form the biggest area, the total number of moves is equal to the total number of clean nodes minus the number of clean nodes on the first row. If k is even, the number of moves in this process is $k^2/4 - k + 1$. If k is odd, the number of moves is $(k^2 - 1)/4 - k + 1$. At last, there will be an extra move for one agent on a node of the border when the coordinator arrives to this position. Therefore, if k is even, the total number of moves performed by the searching agents is $(k^2 - 3k + 4)/2$. If k is odd, the total number of moves performed by the searching agents is $(k^2 - 3k + 4)/2$ as well.

The number of moves performed by the coordinator is calculated as following: to form the biggest triangle or echelon, the coordinator moves across every node on the clean area

except one end on every row and nodes on last row. On the first row of that clean area, the coordinator repeats half nodes twice. After that, the coordinator executes two more extra moves. If k is even, the number of moves performed by the coordinator is $(k^2-8)/4$. If k is odd, the number of moves performed by the coordinator is $(k^2-13)/4$.

Therefore, if k is even, the total number of moves is $(3k^2-6k)/4$. if k is odd, the total number of moves is $(3k^2-6k-5)/4$. ∎

Theorem 14. *The ideal time complexity of Algorithm 2.4 is* $(k^2+2k-12)/4$ *if* k *is even,* $(k^2+2k-15)/4$ *if* k *is odd.*

PROOF

The time consumption can be calculated as follows:

In the process to occupy the nodes on the first row, all agents move to occupy its starting place, this process consumes $(k-2)/2$ time units if k is even, $(k-1)/2$ time units if k is odd. After that, the time units consumed by the coordinator is equal to the number of moves executed by the coordinator. So if k is even, the time units consumed by the coordinator is $(k^2-8)/4$. If k is odd, the time units consumed by the coordinator is $(k^2-13)/4$.

Therefore, if k is even the total number of time units in coordinator model is $(k^2+2k-12)/4$. if k is odd the total number of time units for the node-search on a Mesh with k agents in coordinator model is $(k^2+2k-15)/4$. ∎

2.3.2 Cleaning with Visibility

Decontamination strategy

We assume all agents are initially located at $V_{i,0}$. Then, all agents begin to occupy contiguous positions on the first column and make the homebase as the center among those contiguous positions on the first column. After an agent X occupies its position on the first column, it will move to clean a neighbouring node as soon as it sees that all neighbouring nodes but one are clean or guarded.

This procedure ends when no more agents can move.

The formal description of the algorithm is provided in Algorithm 2.5.

Algorithm 2.5 Disinfecting a $m \times n$ mesh with k agents from home base $V_{i,0}$ with Visibility.

Deployment Phase:

Start with agents in node $V_{i,0}$;
if k is even then
  All Agents are labeled as $S_{-\frac{k}{2}+1}$, $S_{-\frac{k}{2}+2}$,...,$S_{\frac{k}{2}-1}$, $S_{\frac{k}{2}}$;
else
  All Agents are labeled as $S_{-\frac{k-1}{2}}$, $S_{-\frac{k-1}{2}+1}$,...,$S_{\frac{k-1}{2}-1}$, $S_{\frac{k-1}{2}}$;
Agent S_j moves to nodes $V_{i+j,0}$;

Decontamination Phase:

Algorithm 2.1

Correctness and Complexity

Theorem 15. *Algorithm 2.5 performs a contiguous and monotone decontamination*

PROOF

Similar as in Algorithm 2.3. ∎

Theorem 16. *The number of nodes decontaminated by Algorithm 2.5 is* $(k+2)*k/4$ *if* k *is even,* $(k+1)^2/4$ *if* k *is odd.*

PROOF

In Algorithm 2.5, the clean area is a triangle with k nodes as bottom and $(k+1)/2$ nodes as height (if k is odd), the total number of clean nodes in this triangle is $(k+1)^2/4$. If k is even, the total number of clean nodes from Algorithm 2.5 is $(k+2)*k/4$. ∎

Theorem 17. *The total number of moves by all agents in Algorithm 2.5 is* $k(k-1)/2$.

PROOF

The number of moves performed by the agents is calculated as follows:

In the process to occupy the nodes on the first row, an agent labeled as S_i will perform i moves. If k is odd, the total number of moves during this process is $(k^2-1)/4$. If k is even, the total number of moves during this process is $k^2/4$. In the process to move east to form the biggest area, the number of moves is equal to the number of clean nodes in this process because with every move a new node becomes clean. So the number of moves is equal to the number of clean nodes minus the number of nodes in the first column. If k is odd , the number of moves in this process is $(k+1)^2/4-k$. If k is even, the number of moves in this process is $(k+2)*k/4-k$. Therefore, in both cases, the total number of moves performed by k agents is $k(k-1)/2$. ∎

Theorem 18. *The ideal time complexity of Algorithm 2.5 is* $k-1$.

PROOF

In Algorithm 2.5, at time unit 1, some agents occupy nodes $V_{i+1,0}$ and $V_{i-1,0}$. At time unit two, some agents occupy nodes $V_{i+2,0}$ and $V_{i-2,0}$. At the same time, the agent left on the homebase will occupy $V_{i,1}$. To occupy those nodes on the first column , $(k-1)/2$ time units are spent if k is odd; $k/2$ time units are spent if k is even. At the same time of occupying the first column, agents move to the east direction. It costs two time units for each agent to move to east neighbouring node. One time units is spent while waiting for the neighbouring node to be guarded and one time unit is spent to move to the east neighbouring node. The last move is executed by the agent on the middle column. So, if k is odd, the last move is performed by the agent located on the homebase. If k is even, the last move is performed by the agent located on the node $V_{i+1,0}$. In both cases, The number of time unit is $k-1$. ∎

2.4 Homebase inside the Mesh

Let us now consider the situation when the home base is located at a node inside the mesh.

Theorem 19. *Let k be the number of available agents, no deterministic algorithm can clean more than* $2*(k/4)^2 + k/2 + 1$ *nodes.*

PROOF

At any time of the contiguous decontamination strategy, the agents and the borders

of the mesh should besiege the clean nodes. If the clean area does not contain rows or columns with only one guarded node, for each column or row of the clean area, there are at least two agents to guard the two ends of this column or row to avoid re-contamination. For k agents, the biggest clean area has no more than $\lceil k/2 \rceil$ columns and rows. No any other nodes can be cleaned outside this area.

If k is a multiple of 4, it can be easily seen that the biggest clean area with k agents is a diamond like the one shown in Figure2.8 (a) for the same reasoning as in Theorem 10. The four vertices of the diamond are at nodes $V_{j-k/4,i}$,$V_{j,i-k/4}$,$V_{j+k/4,i}$,$V_{j,i+k/4}$. The number of clean nodes are $((k+2)/2+1)*(k/4+1)/2+((k-2)/2+1)*k/4/2 = 2*(k/4)^2+k/2+1$. If k is even but not multiple of 4, the biggest clean area with k agents is shown in Figure 2.8 (b). This figure consists of two identical triangles and each triangle includes $(k/2+1)*(k+2)/4/2$ nodes. The total number of clean nodes is $(k/2+1)^2/2$. So, if k is even, the total number of clean nodes is $\lceil (k/2+1)^2/2 \rceil$. If k is odd, at first we can use $k-1$ agents to clean a diamond area. Then we move the left agent to enlarge the clean area by including the next diagonal line from one peripheral diagonal line. So the biggest clean area with k agents is shown in Figure 2.9. The total number of clean nodes is $\lceil ((k+1)/2+2)*(k+1)/4/2+((k-1)/2+1)*(k+1)/4/2 \rceil = \lceil (k+3)*(k+1)/8 \rceil$.

Now consider the case where the clean area contains some singletons. Let us assume there are i singletons on those columns and rows. The biggest number of clean nodes in the part of clean area without singletons would be no more than $2*(k/4-i)^2+k/2-i+1$ as we showed in the previous part. So the total number of clean nodes would be

$2 * (k/4 - i)^2 + k/2 + 1$. This theorem is still correct in this situation. ∎

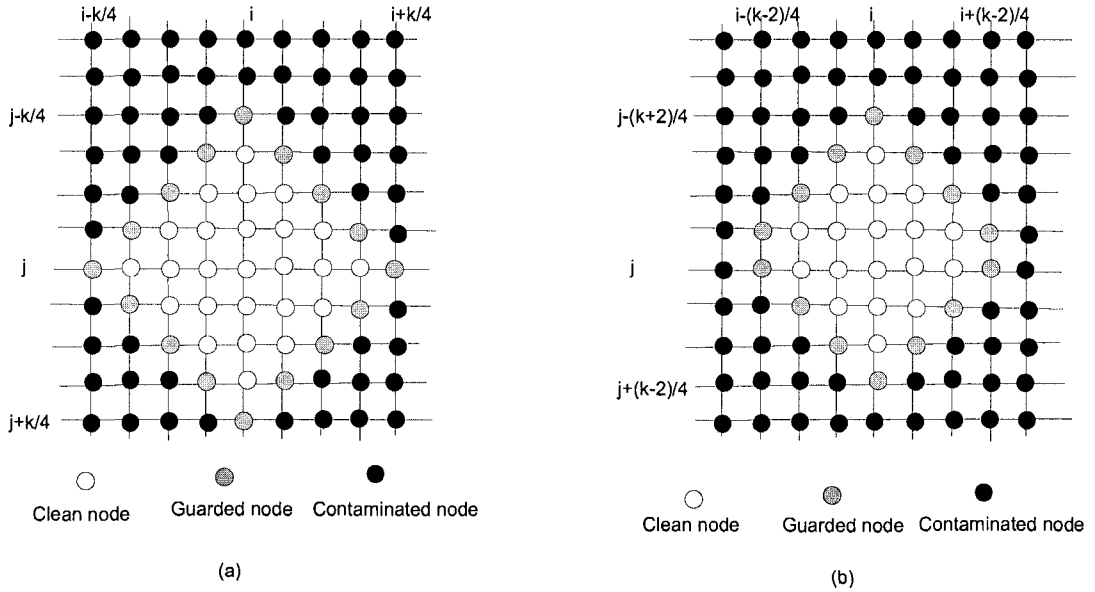

Figure 2.8: Clean area at center of Mesh with multiple of 4 agents(left) and even agents(right)

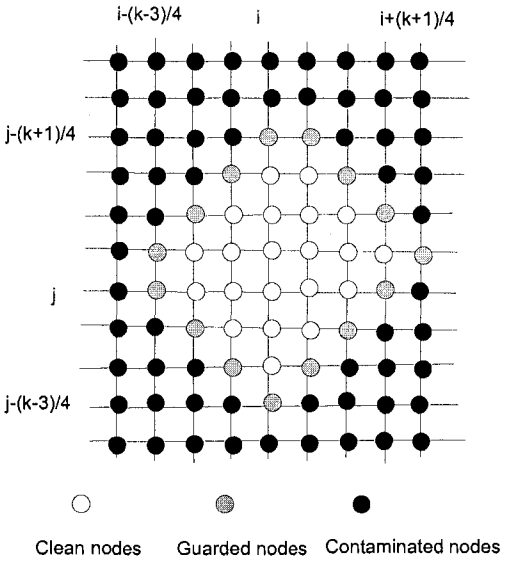

Figure 2.9: Clean area at center of Mesh with odd agents

2.4.1 Cleaning without visibility

At first we assume that k agents are located at the node $V_{i,j}$.

Decontamination strategy

In this strategy, since we can not make use of the borders of mesh, at any step of the decontamination, we have to keep some agents to guard all the borders of the clean part. First, one agent is labeled as coordinator and other agents will move to occupy the consecutive nodes on column j and make sure there are two agents on the same node except for the two ends of those consecutive positions on column j, where there is only

one agent. Beside that, the homebase should be located at the center of the continuous positions on column j. After that, the coordinator will move along column j and when it meets two agents on the same node it makes one agent move to the west neighbouring node and the other to east neighboring node. Then the coordinator will move along column $j-1$ and make the agent move to its west neighboring node when it meets this agent. Every time the coordinator moves along the column, it will make all agents move to the west neighboring nodes, except for two agents on the two ends of those consecutive positions on this column to guard the ends. The coordinator will repeat this procedure on the west part of column j until there are only one or two agents to be moved on that whole column. After that, the coordinator will move to the column $j+1$ and repeat the same process. This process will end when no mobile agents can move. If k is even, then we use $k-2$ agents to occupy the consecutive nodes on column j and let the remaining agent move to occupy the north neighboring node of the end among those consecutive positions on column $j+1$.

The formal description of the algorithm is provided in Algorithm 2.6.

Algorithm 2.6 Disinfecting a $m \times n$ mesh with k-1 agents from home base $V_{i,j}$ with one coordinator.

Start with agents in node $V_{i,j}$. One agent is labeled as coordinator and other agents are labeled from S_0 to S_{k-2};
if k is odd then
Agents S_l moves to node $V_{i+(l \mod \lfloor \frac{k}{2} \rfloor)-\lfloor \frac{k}{4} \rfloor,j}$;
The coordinator moves from node $V_{i,j}$ to node $V_{i+\lfloor \frac{k}{4} \rfloor-1,j}$;
When the coordinator meets agents, the coordinator will let agent S_l ($l < \frac{k}{2}$) move to the west neighboring node, and let agent S_l ($l > \frac{k}{2}$) move to east neighboring node;
The coordinator moves from node $V_{i+\lfloor \frac{k}{4} \rfloor-1,j}$ to node $V_{i-\lfloor \frac{k}{4} \rfloor+1,j}$;
When the coordinator meets agents, the coordinator will let agent S_l ($l < \frac{k}{2}$) move to the west neighboring node, and let agent S_l ($l > \frac{k}{2}$) move to east neighboring node;
The coordinator moves to node $V_{i-\lfloor \frac{k}{4} \rfloor+1,j-1}$

y=1;
for $x = 1$ to $\lfloor \frac{k}{4} \rfloor - 1$
while $y < ((k-1)/2 - x - 1)$
y+=1;
The coordinator moves to $V_{i+y,j-x}$;
When the coordinator meets agents, the coordinator will let the agent move to $V_{i+y,j-x-1}$;
x+=1;
The coordinator moves to $V_{i+y,j-x}$;
while $y > (x+1)$
y-=1;
The coordinator moves to $V_{i+y,j-x}$;
When the coordinator meets one agent, the coordinator will let the agent move to $V_{i+y,j-x-1}$;
x+=1;
The coordinator moves to $V_{i+y,j-x}$;
The coordinator moves to node $V_{i+1,j+1}$;
y=1;
for $x = 1$ to $\lfloor \frac{k}{4} \rfloor - 1$
while $y < ((k-1)/2 - x - 1)$
y+=1;
The coordinator moves to $V_{i+y,j+x}$;
When the coordinator meets agents, the coordinator will let the agent move to $V_{i+y,j+x+1}$;
x+=1;
The coordinator moves to $V_{i+y,j+x}$;
while $y > (x+1)$
y-=1;
The coordinator moves to $V_{i+y,j+x}$;
When the coordinator meets one agent, the coordinator will let the agent move to $V_{i+y,j+x+1}$;
x+=1;
The coordinator moves to $V_{i+y,j+x}$;

else

Agents S_l $(l < k-2)$ moves to node $V_{i+(l \mod \lfloor \frac{k-1}{2} \rfloor) - \lfloor \frac{k-1}{4} \rfloor, j}$;
The coordinator moves from node $V_{i,j}$ to node $V_{i+\lfloor \frac{k-1}{4} \rfloor - 1, j}$;
When the coordinator meets agents, the coordinator will let agent S_l $(l < \frac{k}{2})$ move to the west neighboring node, and let agent S_l $(l > \frac{k}{2})$ move to east neighboring node;
The coordinator moves from node $V_{i+\lfloor \frac{k-1}{4} \rfloor - 1, j}$ to node $V_{i-\lfloor \frac{k-1}{4} \rfloor + 1, j}$;
When the coordinator meets agents, the coordinator will let agent S_l $(l < \frac{k}{2})$ move to the west neighboring node, and let agent S_l $(l > \frac{k}{2})$ move to east neighboring node;
The coordinator moves to node $V_{i-\lfloor \frac{k-1}{4} \rfloor + 1, j-1}$
y=1;
for $x = 1$ to $\lfloor \frac{k-1}{4} \rfloor - 1$
while $y < ((k-2)/2 - x - 1)$
y+=1;
The coordinator moves to $V_{i+y,j-x}$;
When the coordinator meets agents, the coordinator will let the agent move to $V_{i+y,j-x-1}$;
x+=1;
The coordinator moves to $V_{i+y,j-x}$;
while $y > (x+1)$
y-=1;

The coordinator moves to $V_{i+y,j-x}$;
When the coordinator meets one agent, the coordinator will let the agent move to $V_{i+y,j-x-1}$;
x+=1;
The coordinator moves to $V_{i+y,j-x}$;
The coordinator moves to node $V_{i+1,j+1}$;
y=1;
for $x = 1$ to $\lfloor \frac{k-1}{4} \rfloor - 1$
while $y < ((k-2)/2 - x - 1)$
y+=1;
The coordinator moves to $V_{i+y,j+x}$;
When the coordinator meets agents, the coordinator will let the agent move to $V_{i+y,j+x+1}$;
x+=1;
The coordinator moves to $V_{i+y,j+x}$;
while $y > (x+1)$
y-=1;
The coordinator moves to $V_{i+y,j+x}$;
When the coordinator meets one agent, the coordinator will let the agent move to $V_{i+y,j+x+1}$;
x+=1;
The coordinator moves to $V_{i+y,j+x}$;
The agent S_{k-2} moves to node $V_{i-\lfloor \frac{k-1}{4} \rfloor, j+1}$;
for $x = 1$ to $\lfloor \frac{k-1}{4} \rfloor$
The agent at node $V_{i-\lfloor \frac{k-1}{4} \rfloor + x, j+x}$ moves to node $V_{i-\lfloor \frac{k-1}{4} \rfloor + x, j+x+1}$;
x+=1;

Correctness and Complexity

Theorem 20. *Algorithm 2.6 performs a contiguous and monotone decontamination*

PROOF

Base Case: The decontamination in deployment phase is contiguous and monotone.

Assumption

Let us assume that the cleaning process is contiguous and monotone up to step $i (i > 0)$. At this step, all the clean nodes are connected and all the agents guard the contiguous border of the clean area.

Induction

At step $i+1$, there are three cases of movement for agent on node $V_{x,y}$. The first possible case takes place when this agent is located on a node which is on column j. According to Algorithm 2.6, there are two agents on the same node on the continuous positions occupied by the agents except for the two ends on column j. Before the step $i+1$, nodes $V_{x+1,y}$ and $V_{x-1,y}$ are clean or guarded. In this step the two agents on node $V_{x,y}$ will move to $V_{x,y+1}$ and $V_{x,y-1}$. This moving is contiguous and monotone. The second case happens when the agent is located on a node to the left of the column j. Before step $i+1$, nodes $V_{x+1,y}$, $V_{x-1,y}$ and $V_{x,y+1}$ are clean or guarded. In step $i+1$, the agent will move to $V_{x,y-1}$. This moving is contiguous and monotone. The third case is that the agent is located on a node on the right of column j. Before step $i+1$, the nodes $V_{x+1,y}$, $V_{x-1,y}$ and $V_{x,y-1}$ are clean or guarded. In step $i+1$, the agent will move to $V_{x,y+1}$. This moving is still contiguous and monotone. So in all cases, the moving in step $i+1$ is contiguous and monotone and the clean area increases of a new node $V_{x,y}$ ∎

Theorem 21. *The number of nodes decontaminated by Algorithm 2.6 is* $\lceil (k+2)*k/8 \rceil + 2$ *if* k *is even,* $\lceil ((k+1)^2/8 \rceil + 2$ *if* k *is odd.*

PROOF

We are using one agent as a coordinator, so we have to use $k-1$ agents to perform the cleaning. The number of nodes decontaminated by $k-1$ agents is $\lceil ((k-1)/2+1)^2/2 \rceil$ when k is odd, $\lceil (k+2)*k)/8 \rceil$ when k is even. After that, the coordinator can move to

one node on the border of the clean part and let the agent on this node to move to one of its uncleaned neighboring node and the coordinator can move to another uncleaned neighboring node. So, the total number of clean nodes is $\lceil((k+1)/2)^2/2\rceil + 2$ when k is odd, $\lceil(k+2)*k/8\rceil + 2$ when k is even. ∎

Theorem 22. *The total number of moves by all agents in Algorithm 2.6 is* $\lceil(3k^2 - 10k + 20)/8\rceil$ *when k is even,* $\lceil(3k^2 - 8k + 17)/8\rceil$ *when k is odd.*

PROOF

In the process to occupy the nodes on the first row, agent S_i ($i \leq k-2$) performs (l mod $\lfloor\frac{k}{2}\rfloor$)$-\lfloor\frac{k}{4}\rfloor$ moves. If k is even, the total number of moves in this process is $\lceil(k-2)^2/8\rceil$; if k is odd, it is $\lceil(k-1)^2/8\rceil$. In the process to form the biggest area, the total number of moves is equal to the total number of clean nodes minus the number of clean nodes on column j. If k is even, the number of moves in this process is $\lceil(k+2)*k)/8\rceil + 2 - k/2 = \lceil(k-2)*k/8\rceil + 2$. If k is odd, the number of moves is $\lceil((k+1)/2)^2/2\rceil + 2 - (k+1)/2 = \lceil(k+1)(k-3)/8\rceil + 2$. Therefore, if k is even, the total number of moves consumed by the searching agents is $\lceil(k-2)*(2k-2)/8\rceil + 2 = \lceil(k^2-3k+10)/4\rceil$. If k is odd, it is $\lceil(k^2-2k+7)/4\rceil$.

The number of moves performed by the coordinator is calculated as following: the coordinator moves across every node on the clean area except for the border. On column j of the clean area, the coordinator passes by half nodes twice. If k is even, the number of moves consumed by the coordinator is then $\lceil(k^2-4k)/8\rceil$. If k is odd, it is $\lceil(k^2-4k+3)/8\rceil$.

Therefore, if k is even, the total number of moves in coordinator model is $\lceil(3k^2 - 10k + 20)/8\rceil$; if k is odd, it is $\lceil(3k^2 - 8k + 17)/8\rceil$. ∎

Theorem 23. *The ideal time complexity of Algorithm 2.6 is* $\lceil(k^2 - 2k - 4)/8\rceil$ *when k is even,* $\lceil(k^2 - 2k + 1)/8\rceil$ *when k is odd.*

PROOF

The number of time units is equal to the sum of total number of moves by coordinator plus the time to occupy the column j. If k is even, the time to occupy the column j is $\lceil(k-2)/4\rceil$; the total time units is $\lceil(k^2 - 2k - 4)/8\rceil$. If k is odd, the time to occupy the column j is $\lceil(k-1)/4\rceil$; the total time units is $\lceil(k^2 - 2k + 1)/8\rceil$. ∎

2.4.2 Cleaning with Visibility

Decontamination strategy

First, we assume all agents are located at node $V_{i,j}$. In the deployment phase, all agents occupy the consecutive nodes on column j and column $j+1$. The homebase is the center of these contiguous positions in column j. If k is even, there are two more agents on column j than agents on column $j+1$, if k is odd, there are one more agent on column j than agents on column $j+1$. After one agent arrives at its starting position, it will execute decontamination as in Algorithm 2.1.

The formal description of the algorithm is provided in Algorithm 2.7.

Algorithm 2.7 Disinfecting a $m \times n$ mesh with k agents from home base $V_{i,j}$ with Visibility.

Deployment Phase:

Start with agents in node $V_{i,j}$. Agents are labeled from S_0 to S_{k-1};
if k is even then
  if $j \leq \lfloor \frac{k}{2} \rfloor$ then
    Agents S_l moves to node $V_{i+l-\lfloor \frac{k}{4} \rfloor, j}$;
  else
    Agents S_l moves to node $V_{i+(l \mod \lfloor \frac{k}{2} \rfloor)-\lfloor \frac{k}{4} \rfloor, j+1}$;
else
  if $j \leq \lfloor \frac{k-1}{2} \rfloor$ then
    Agents S_l moves to node $V_{i+l-\lfloor \frac{k-1}{4} \rfloor, j}$;
  else
    Agents S_l $(l < k-1)$ moves to node $V_{i+(l \mod \lfloor \frac{k-1}{2} \rfloor)-\lfloor \frac{k-1}{4} \rfloor, j+1}$;
  The agent S_{k-1} moves to node $V_{i-\lfloor \frac{k-1}{4} \rfloor, j+1}$;

Decontamination Phase:

Algorithm 2.1

Theorem 24. *The number of nodes decontaminated by Algorithm 2.7 is* $\lceil (k/2+1)^2/2 \rceil$ *if k is even,* $\lceil (k+3)*(k+1)/8 \rceil$ *if k is odd.*

PROOF

With this algorithm we achieve the maximum number of clean nodes, therefore the number of clean nodes as in Theorem 19 can be achieved. ∎

Theorem 25. *The total number of moves performed by all agents in Algorithm 2.7 is* $\lfloor (k^2+6)/4 \rfloor$ *when k is even,* $\lfloor (k^2-k+8)/4 \rfloor$ *when k is odd.*

PROOF

The total number of moves is equal to all moves on column j and $j+1$ plus the total number of clean nodes minus the number of clean nodes on column j and $j+1$. If k is

even, the total number of moves is $\lfloor((k+1)^2 - 2(k+1) + 7)/4\rfloor = \lfloor(k^2+6)/4\rfloor$. If k is odd, the total number of moves is $\lceil((k+1)^2 - 3(k+1) + 10)/4\rceil = \lfloor(k^2-k+8)/4\rfloor$. ∎

Theorem 26. *The ideal time complexity of Algorithm 2.7 is $k/2$ if k is even, $(k+1)/2$ if k is odd.*

PROOF

Similar to Algorithm 2.5, each move on a row consumes 2 time units and each move on a column consumes 1 time unit. The last move is executed by the agent which is located originally on the middle node of column j. When k is even, the total time units is $k/2$. When k is odd, it is $(k+1)/2$. ∎

2.5 Summary

The complexities of decontaminating mesh with k agents following different strategies are summarized in table 2.1:

From the comparison of the complexity between the local model and visibility model in the table 2.1, we can easily get to a conclusion that the model with visibility is much more efficient than the one without. Especially, this model can decrease the total movements of all agents and the time costs to perform decontamination remarkably.

Decontamination Complexities			
Decontamination Strategy	Number Of	Model	
		Local model	Visibility Model
Homebase at Corner	Clean Nodes	$k(k-1)/2+2$	$k(k+1)/2$
	Moves	$(3k^2-7k)/2-\lfloor k/2 \rfloor+3$	$k(k-1)$
	Time Units	$(k^2+k-4)/2-\lfloor k/2 \rfloor$	$2\times(k-1)$
Homebase at Border	Clean Nodes	$k^2/4+2$	$(k+1)^2/4$
	Moves	$(3k^2-6k)/4$	$k(k-1)/2$
	Time Units	$(k^2+2k-12)/4$	$k-1$
Homebase inside Mesh	Clean Nodes	$\lceil (k+2)*k/8 \rceil+2$	$\lceil (k/2+1)^2/2 \rceil$
	Moves	$\lceil (3k^2-10k+20)/8 \rceil$	$\lfloor (k^2+6)/4 \rfloor$
	Time Units	$\lceil (k^2-2k-4)/8 \rceil$	$k/2$

Table 2.1: Decontamination Complexities in Mesh

Chapter 3

Disinfecting Octagonal Mesh and Hexagonal Mesh

A $m \times n$ octagonal mesh is a graph $OM_{m,n}$ = (V,E) with $V_{i,j} \in V, 0 \leq i \leq m-1, 0 \leq j \leq n-1$ and E contains the edges ($V_{i,j}$, $V_{i+1,j}$) ($i \neq m-1$), ($V_{i,j}$, $V_{i,j+1}$) ($j \neq n-1$), ($V_{i,j}$, $V_{i-1,j+1}$) ($i \neq 0$ and $j \neq n-1$) and ($V_{i,j}$, $V_{i+1,j+1}$) ($i \neq m-1$ and $j \neq n-1$). In the octagonal mesh each vertex $V_{i,j}$ with $0 < i < m-1$, $0 < j < n-1$ is connected to the eight vertices $V_{i-1,j}$, $V_{i-1,j-1}$, $V_{i,j-1}$, $V_{i+1,j-1}$, $V_{i+1,j}$, $V_{i+1,j+1}$, $V_{i,j+1}$ and $V_{i-1,j+1}$.

A $m \times n$ hexagonal mesh is a graph $HM_{m,n}$ = (V,E) with $V_{i,j} \in V, 0 \leq i \leq m-1, 0 \leq j \leq n-1$ and E contains exactly the edges ($V_{i,j}$, $V_{i+1,j}$) ($i \neq m-1$), ($V_{i,j}$, $V_{i,j+1}$) ($j \neq n-1$) and ($V_{i,j}$, $V_{i+1,j-1}$) ($i \neq m-1$ and $j \neq 0$). In the hexagonal mesh each vertex $V_{i,j}$ with $0 < i < m-1$, $0 < j < n-1$ is connected to the six vertices $V_{i-1,j}$, $V_{i+1,j}$, $V_{i,j-1}$, $V_{i,j+1}$, $V_{i+1,j-1}$ and $V_{i-1,j+1}$.

We remind that the notion of rigid frontier has been defined in Chapter 1 (definition 1) and it will be used also in this Chapter.

From paper [21], we know that $Min(m,n)$ agents are needed to clean the whole network for both of the octagonal mesh and hexagonal mesh. In this chapter we will demonstrate the maximum number of nodes that can be decontaminated with k agents($k < Min(m,n)$).

3.1 Disinfecting an Octagonal Mesh

Lemma 3. *In an octagonal mesh, a rigid frontier can not be composed by diagonal segments containing three nodes or more.*

PROOF

If there is a diagonal segment with three nodes $V_{i,j}$, $V_{i+1,j-1}$, $V_{i+2,j-2}$ in the rigid frontier (where the clean part is to the *west* of the diagonal, eg, Figure 3.1). Without contamination, another diagonal ($V_{i,j-1}$ and $V_{i+1,j-2}$, as Figure 3.1 (a) or $V_{i+1,j}$ and $V_{i+2,j-1}$ as Figure 3.1 (b)) should also be in the rigid frontier. In both cases, some agents on the diagonal segments can move, contradiction to the definition of rigid frontier. ∎

Theorem 27. *No deterministic algorithm can clean more than $(k+1)^2/4$ nodes if the homebase is located on the corner of octagonal mesh.*

PROOF

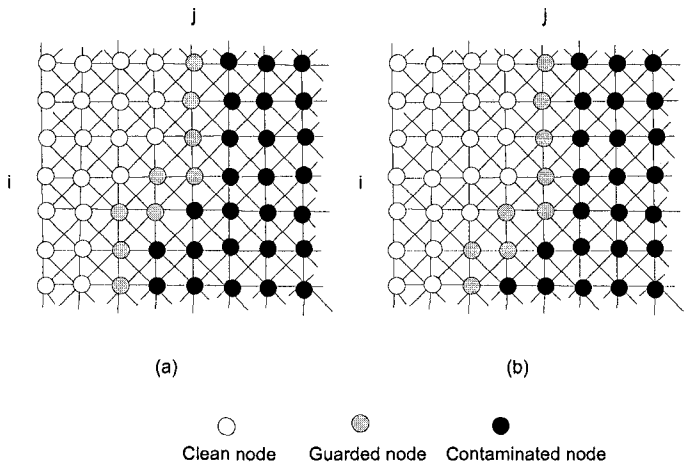

Figure 3.1: Diagonal segments on the rigid frontier

If the homebase is at node $V_{0,0}$, from lemma 3 we know the rigid frontier of the clean area will be formed only by horizontal segments, vertical segments and diagonals with a length of two. At first we want to prove that in the biggest decontaminated area, a diagonal segment $V_{i+1,j-1}, V_{i,j}$ can appear in the border of the clean part only if $V_{i+1,j}$, $V_{i+1,y}$ $(y < j-1)$, and $V_{x,j}$ $(x < i)$are also part of the border. If node $V_{i+1,j}$ is not on the border, then $V_{i,j-1}$ should be on the border of the decontaminated area. Without violation of the definition of rigid frontier, node $V_{i-1,j}$ should be contaminated and $V_{i-1,j-1}$ should be guarded, as in Figure 3.2 (a). It is easy to see that the number of decontaminated nodes in this case is not maximum, and Figure 3.2 (b) corresponds to a higher number

of decontaminated nodes with the same number of agents. So, the only possibility is that $V_{i+1,j}$ is guarded. However, this is possible only if the neighboring guarded node of $V_{i+1,j-1}$ is $V_{i+1,j-2}$, otherwise the agent on $V_{i+1,j-1}$ could move to $V_{i+2,j-1}$ violating the definition of rigid frontier. Following the same reasoning, $V_{i-1,j}$ also must be guarded. And nodes $V_{i+1,y}$ $(y < j-2)$, $V_{x,j}$ $(x < i-1)$ and $V_{x,j}$ $(x < i-1)$ must be guarded as well.

By now we know that the shape of the clean area will be a rectangle like in Figure 3.2 (c). The length of the rigid frontier is equal to k. Since the square is the rectangle that maximizes the area with a given perimeter, we have that the maximum number of clean nodes will be $k(k+2)/4$ if k is even and $(k+1)^2/4$ if k is odd. ∎

Theorem 28. *No deterministic algorithm can clean more than* $(\lfloor k/3 \rfloor + 1)^2$ *nodes if the homebase is located on the border of the octagonal mesh.*

PROOF

Let us assume that the homebase is located on the first row, we can use the available agents, as well as the first row to be the border of the clean area. Similarly to Theorem 27, it is easy to see that the biggest area is a rectangle or a rectangle plus one more node on the east border of the clean area. So the biggest area will be a rectangle whose three borders' perimeter sum is k. ∎

With the same reasoning, we can show that

Theorem 29. *No deterministic algorithm can clean more than* $(\lfloor k/4 \rfloor + 1)^2$ *nodes if the*

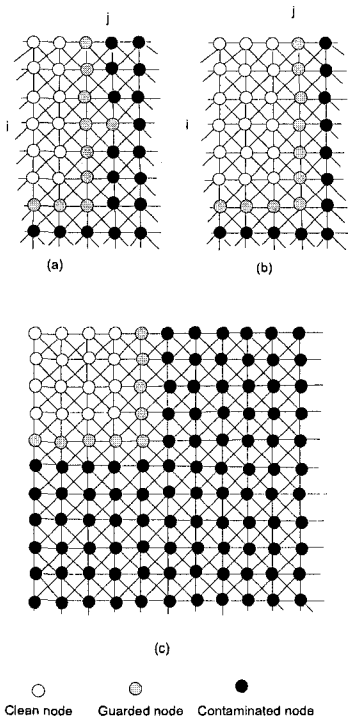

Figure 3.2: Clean area from homebase at corner of octagonal mesh

homebase is inside the octagonal mesh.

3.2 Disinfecting an Hexagonal Mesh

Theorem 30. *If the homebase is located at the North-West or South-East corner, no deterministic algorithm can clean more than* $k(k+1)/2$ *nodes; if it is located at the North-*

East or South-West corner, no deterministic algorithm can clean more than $(k+1)^2/4$ *nodes.*

PROOF

For a $m \times n$ hexagonal mesh, if the homebase is located at the corner $V_{0,0}$ or $V_{m-1,0}$, we can clean the same area as in the $m \times n$ regular mesh from a corner (as in Figure 3.3 (a)). So the biggest number of clean nodes with k agents is $k(k+1)/2$. If the homebase is located at the corner $V_{0,n-1}$ or $V_{m-1,n-1}$ (as in Figure 3.3(b)), we can clean the same area as in the octagonal mesh. The biggest number of clean nodes with k agents is $(k+1)^2/4$ in this case. ∎

Lemma 4. *If the homebase is located on a border of the hexagonal mesh the rigid frontier can be formed only by an horizontal segment on the south (or north) border, a vertical segment on east (or west) border, and a consecutive diagonal segment.*

PROOF

If the homebase is located on the border of the hexagonal mesh, the rigid frontier will be on some rows, columns or diagonals. Similarly to Theorem 27, we can prove that a segment of the type: $V_{i+1,j-1}$, $V_{i,j-1}$, $V_{i,j}$ cannot belong to the frontier. We now show that there is only one horizontal segment on the south (north) border , and only a vertical segment on the east border. By contradiction, assume there are more than one horizontal (vertical) segment on the south (east) border, it is easy to see that some agents would be able to continue the decontamination. We now want to prove that if there is a diagonal

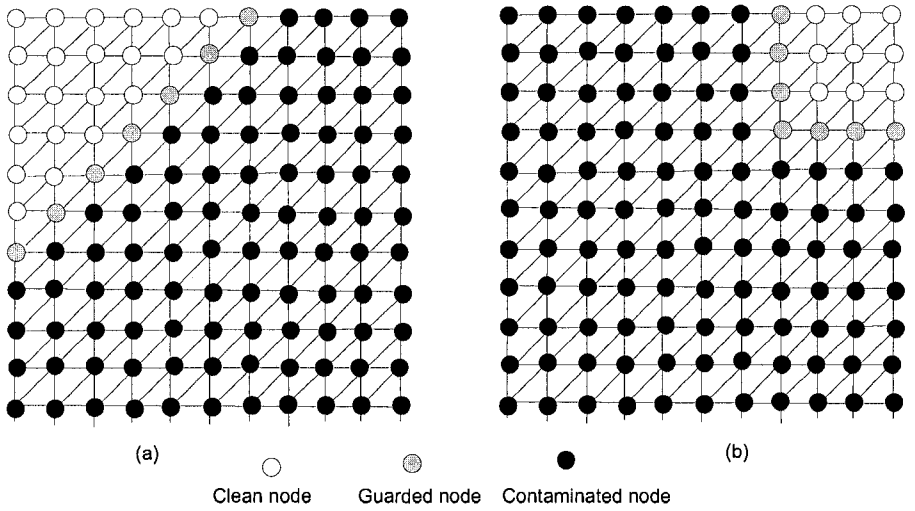

Figure 3.3: Clean area from homebase at corner of hexagonal mesh

segment, it must be consecutive; in other words, we want to show that there is no more than one diagonal segment in the rigid frontier. By contradiction, assume there are more than one diagonal segment in the rigid frontier as in Figure 3.4 (a). However, the agent on node $V_{i,j}$ can move to node $V_{i,j+1}$, giving a contradiction. In fact other agents between the two diagonal segments can move to the west direction. Those movements will continue until there is a single continuous diagonal segment on the rigid frontier, as in Figure 3.4 (b). ∎

Theorem 31. *No deterministic algorithm can clean more than* $\lfloor((k+1.5)^2 - 6k)/6\rfloor + k$

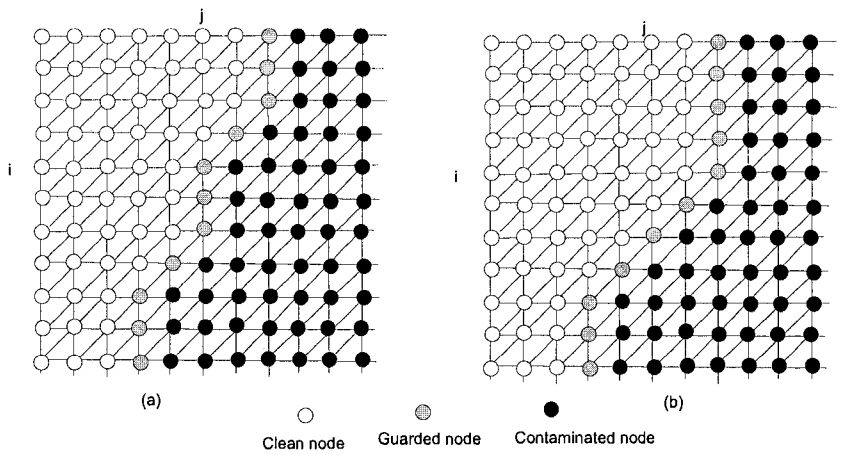

Figure 3.4: Hypotenuse border change

nodes if the homebase is located on the border.

PROOF

At first let us assume that the homebase is on the west border of hexagonal mesh. Let us now compare the number of clean nodes among all possible clean area shapes. According to the previous lemma, if the frontier occupied by the agents consists of only rows and columns, the biggest clean area will be a rectangle. The maximum number of clean nodes is the number of nodes in the rectangle whose three borders' perimeter sum is k (as in Figure 3.5 (a)). The total number of clean nodes is then no more than $(\lfloor k/3 \rfloor + 1)^2$, which is $\Theta(k^2/9)$. If the border occupied by the agents consists of only row

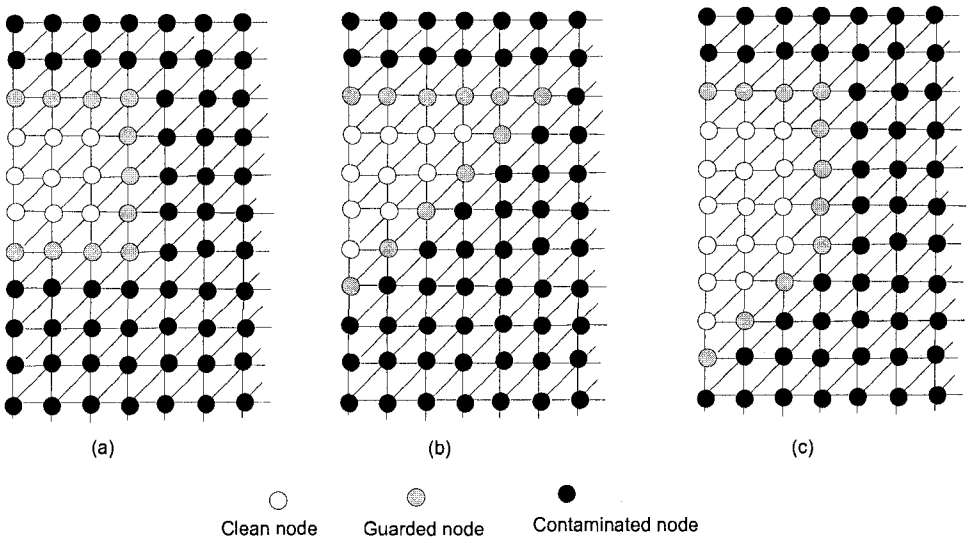

Figure 3.5: Possible clean area from homebase at border of hexagonal mesh

and diagonal, the biggest clean area will be a triangle. The maximum number of clean nodes is the number of nodes in a triangle whose bottom and height is equal to $\lfloor k/2 \rfloor + 1$ (as in Figure 3.5 (b)).The total number of clean nodes will be $(\lfloor k/2 \rfloor + 1)^2/2$, which is $\Theta(k^2/8)$. If the border occupied by the agents consists of row, column, diagonal, it will be an area like Figure 3.5 (c). It consists of a rectangle and a triangle. Let the number of agents on the diagonal be x, then the agents on the row will be x and the agents on the column will be $k+2-2x$. The total number of clean nodes except the border will be $(x-1)((k+2)-2x-1)+(x-1)(x-2)/2 = -1.5x^2+(k+1.5)x-k$. The maximum value of this function is $((k+1.5)^2-6k)/6$ when x is equal to $(k+1.5)/3$. So the total number

of clean nodes will be no more than $\lfloor((k+1.5)^2-6k)/6\rfloor+k$ in this case, which is $\Theta(k^2/6)$. So Figure 3.5 (c) keeps the biggest number of clean nodes when the homebase is located on the border of hexagonal mesh and we can clean no more than $((k+1.5)^2-6k)/6$ nodes with k agents. If the homebase is on the other border, the possible clean area shape will be same as the one on the west border. ∎

Theorem 32. *No deterministic algorithm can clean more than $\Theta(k^2/14)$ nodes if the homebase is located inside the hexagonal mesh.*

PROOF

If the homebase is located inside the hexagonal mesh, the border of the maximum clean area must be on some rows , columns or diagonal segments of mesh. We can prove that there is only one column (row) on each side of the clean area, and there is only one continuous diagonal segment on the clean area as we proved in last theorem. It is easy to see that if there is a diagonal segment in the border, it must be on the west border of the clean area. Here we want to prove that the diagonal segment must connect with the north border in the biggest clean area. Assuming that the diagonal segment does not connect with the north border, as in Figure 3.6 (a), then the agent on node $V_{i,j}$ can move to node $V_{i,j-1}$, giving a contradiction. In fact, in this case, the movement will continue and all agents on the diagonal segment and column j can move to the west direction until the diagonal segment connects with north border, as in Figure 3.6 (b).

Now let us compare the number of clean nodes among all possible clean area shapes.

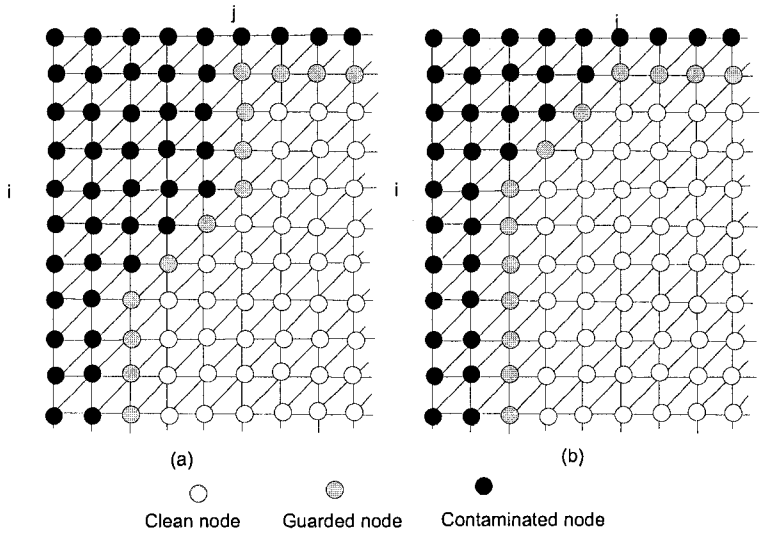

Figure 3.6: Hypotenuse border change inside hexagonal mesh

If the clean area is a triangle, the number of nodes on the height of the triangle is $\lfloor k/3 \rfloor + 1$, same number of nodes at bottom. The total number of clean nodes will be $1 + 2 + ... + \lfloor k/3 \rfloor + 1$, equal to $(\lfloor k/3 \rfloor + 2)(\lfloor k/3 \rfloor + 1)/2$, which is $\Theta(k^2/18)$. If the clean area is a rectangle, as in Figure 3.7 (a), the biggest number of clean nodes will be $(\lfloor k/4 \rfloor + 1)^2$. This number is $\Theta(k^2/16)$, more than the number of clean nodes in a triangle. Now let us consider there is a diagonal segment on the west border as in Figure 3.7 (b). Let x be the number of agents on the row of the north border, y the number of agents on the diagonal segment and z the number of agents on the column of the west border,

then the total number of clean nodes is $(2x + y - 1)y/2 + (x + y - 1)(z - 1)$, and this function satisfies the condition $2(x+y-1)+(z-2)+(y+z-3) = k$. This function can be simplified to $-2x^2 - 2y^2 - 3xy + (k+7)x + (k+7)y - (k+5)$. The maximum value of this function is $(k+7)^2/14 - (k+5)/2 = \Theta(k^2/14)$ when $x = y = (k+7)/7$. This number of clean nodes is the biggest number we can clean with k agents. ∎

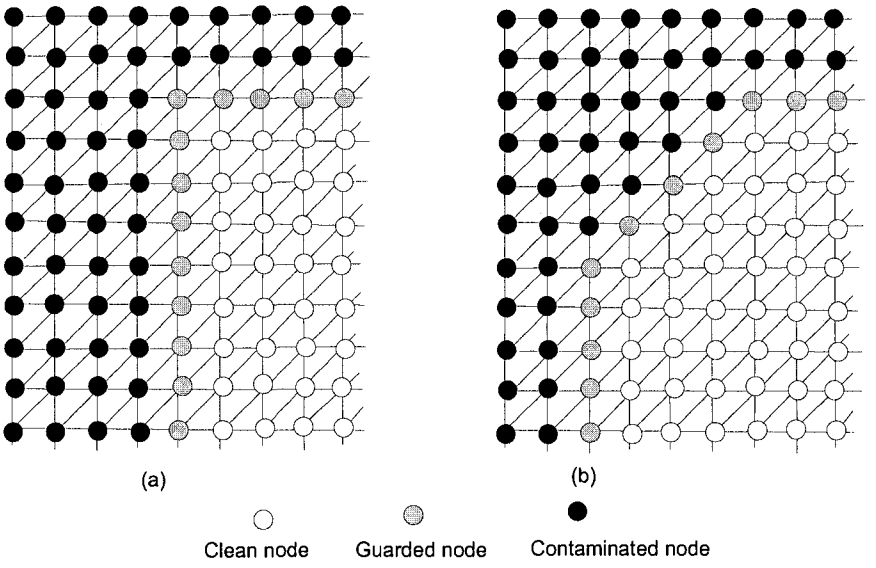

Figure 3.7: Possible clean area from homebase inside hexagonal mesh

Decontamination Capability		
Decontamination Strategy	Network Topology	Biggest Number of Clean Nodes
Homebase at Corner	Mesh	$k(k+1)/2$
	Hexagonal Mesh	$k(k+1)/2$
	Octagonal Mesh	$(k+1)^2/4$
Homebase at Border	Mesh	$\Theta(k^2/4)$
	Hexagonal Mesh	$\Theta(k^2/6)$
	Octagonal Mesh	$\Theta(k^2/9)$
Homebase at Inside	Mesh	$\Theta(k^2/8)$
	Hexagonal Mesh	$\Theta(k^2/14)$
	Octagonal Mesh	$\Theta(k^2/16)$

Table 3.1: Decontamination Capability in Mesh, Hexagonal Mesh and Octagonal Mesh

3.3 Summary

In Table 3.1, we want to compare the difference among the biggest number of clean nodes decontaminated with k agents in mesh, octagonal mesh and hexagonal mesh.

From the table, we can see that the decontamination capability depends on the network topology. For meshes, when the degree of the nodes increases, the maximum number of clean nodes with k agents decreases.

Chapter 4

Disinfecting Trees

In this chapter we consider a team of k agents decontaminating a tree. We describe an optimal algorithm for arbitrary trees as well as simple optimal algorithms for complete tree.

In this chapter we assume that each node in the tree has a storage area (whiteboard) used by the agents to communicate. The whiteboard can be accessed by the agents present on the node.

4.1 Disinfecting an Arbitrary Tree

The biggest number of nodes that can be decontaminated with k agents starting from a given location in an arbitrary tree depends on the tree's structure. The problem to arrange the agents' movements along the tree consists of choosing the best path along

which the k agents can clean as much nodes as possible.

4.1.1 Greedy algorithm with general saturation algorithm

Using a general saturation algorithm, one can determine for each node the minimum number of agents needed to decontaminate each subtree [4]. With this information, by using a simple greedy algorithm, we can decontaminate the general tree with k agents starting from a given homebase in the following way: at first $k-1$ agents can be used to clean all subtrees from the homebase which can be completely cleaned with less than k agents. After this procedure, the remaining unclean subtrees need more than $k-1$ agents to be cleaned. If there is only one unclean subtree left, then k agents can go directly to the root of the subtree and repeat the same clean procedure as the one performed before at the homebase; if there are more subtrees left, choose one that requires the smallest number of agents, move $k-1$ at its root, and repeat the same procedure.

Consider, for example, the tree of Figure 4.1, where 4 agents are located at node X (the homebase). Node X has four subtrees S_1, S_2, S_3, S_4. Let S_1, S_2 and S_3 have the same structure. Using the algorithm from [4], we could have the information about the number of agents necessary to fully clean each of its subtrees at each node. Let us observe how the greedy algorithm would proceed. Each of S_1 and S_2 and S_3 needs 4 agents to be decontaminated while S_4 needs 2 agents. With the greedy algorithm, S_4 will be cleaned. After that, one agent is left to guard node X and the other 3 agents move to one of uncleaned subtrees. Following the greedy algorithm, 9 nodes in that subtree will

be decontaminated. This simple greedy algorithm is clearly *not optimal.* For example, a better strategy would have been the following: We start cleaning S_4. After that, we move 2 agents to clean the fraction1 part of S_1 and leave one agent to guard the root of S_1 and return the other agents back to X. Now we have 3 agents in node X and we can move 2 agents to S_2 to clean the same part as fraction1 in S_1 and return one agent back to node X. At this time we have 2 agents located in node X. We move the remaining 2 agents to S_3 to clean the same part as fraction1 in S_1. At last we can move the last free agent to clean one extra node in S_3. In this way we can clean 23 nodes.

The reason that the simple greedy algorithm is not optimal is that there is no enough global information locally available at each node. For decontamination problem with k agents, according to the greedy algorithm, at some step, there must be more than one subtrees left which need more than $k-1$ agents to be fully decontaminated otherwise the tree can be decontaminated with k agents. At this step, the greedy algorithm is not able to determine, with the information locally available, the optimal agents' movement.

The key to solve this problem is to let every node acquire enough information about the structure of its subtrees. If every node has enough information about its subtrees, the agents could locally determine the best strategy. So, we need to find a simple form to express the required structure information and send it to related nodes. In the next section we will show an algorithm that can find the optimal movement for the agents if each node knows how many nodes can be decontaminated for each subtree, and for each $i < j < k$ how many nodes can be decontaminated deploying j agents in the subtree, at

the same time leaving i of them to guard this subtree.

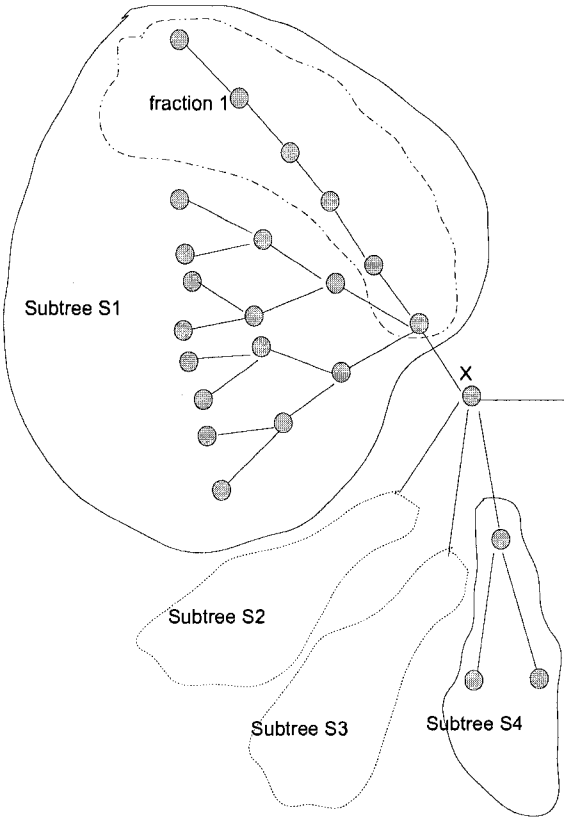

Figure 4.1: Biggest part clean with k agent from home base in a tree

4.1.2 Optimal Decontamination

In this section, we develop an algorithm to clean the tree. Before we describe the algorithm, we define some notations. Let $\mathbf{L}$ be a vector $[N_1, N_2, ..., N_{r-1}, N_r]$, where each N_i $(1 \leq i \leq r)$ is a vector of size i, $(n_{i,1}, n_{i,2}, ..., n_{i,(i-1)}, n_{i,i})$. For instance, $\mathbf{L} = [\underbrace{(n_{1,1})}_{N_1} \underbrace{(n_{2,1}, n_{2,2})}_{N_2} \underbrace{(n_{3,1}, n_{3,2}, n_{3,3})}_{N_3} \underbrace{(n_{4,1}, n_{4,2}, n_{4,3}, n_{4,4})}_{N_4}]$. Let $s(\mathbf{L})$ refer to the number of components (size) of vector $\mathbf{L}$. We call $n_{s(\mathbf{L}),s(\mathbf{L})}$ the *extreme number* in the vector. In the example above, $s(\mathbf{L}) = 4$ and the extreme number is $n_{4,4}$. The decontamination algorithm is preceded by a preprocessing phase. During the preprocessing, each node X collects information from its subtrees and computes a vector **myL** to be sent to its parent. As we will show later, **myL** contains the information about the biggest number of nodes that can be decontaminated from X by using a certain number of agents in its subtrees. More precisely, in vector $N_i = (n_{i,1}, n_{i,2}, ..., n_{i,(i-1)}, n_{i,i})$ of **myL**, the component $n_{i,j}$ contains the highest number of nodes that can be decontaminated in the subtrees rooted at X by using i agents and leaving j agents in the subtrees to guard the portion decontaminated. After the preprocessing phase, in Algorithm 4.2, the agents will be able to choose the best cleaning movement according to the information collected at the homebase.

We divide our strategy in two parts. The first part *preprocessing* is to calculate the necessary information for each node in the tree. The second part is to execute the decontamination movements.

Preprocessing.

The key point of the preprocessing is to perform a saturation from the leaves to the homebase. Receiving vectors $\mathbf{L_1}$, $\mathbf{L_2}$, ..., $\mathbf{L_m}$ ($s(\mathbf{L_1}) \geq s(\mathbf{L_2}) \geq ... \geq s(\mathbf{L_m})$) from its neighbours, node X has to determine, for all values p ($p \leq k$) and q ($q \leq p$), how many nodes can be decontaminated from X in its subtrees by using p agents and leaving q behind. This information is sent to X's parent in message **myL**. When the homebase receives the information from all children, it can determine the optimal movements.

More precisely, the preprocessing is a saturation algorithm according to following rules:

1. Every leaf sends a message $[N_1]$ to its parent, where $[N_1] = [(n_{1,1})] = [(1)]$; $n_{1,1}$ contains the biggest number of nodes that can be decontaminated by using 1 agents to clean and leaving 1 agents to guard. From each leaf of the tree, obviously a single agent can only clean this leaf, so $[(n_{1,1})] = [(1)]$

2. If a node X has only one child, it receives one message $\mathbf{L}$. Node X can compute **myL** by adding 1 to each component of $\mathbf{L}$, e.g. in Figure 4.2 (a), when $\mathbf{L} = [(1)(1,2)(7,7,7)]$, $\mathbf{myL} = [(2)(2,3)(8,8,8)]$, the reason is that with the same number of agents, an additional node (i.e., node X) can be decontaminated.

3. If a node X has more than one child, it receives messages $\mathbf{L_1}$, $\mathbf{L_2}$, ..., $\mathbf{L_m}$ ($s(\mathbf{L_1}) \geq s(\mathbf{L_2}) \geq ... \geq s(\mathbf{L_m})$). Let $\mathbf{L_i} = [N_1^i, N_2^i, ..., N_{s(L_i)}^i]$ ($1 \leq i \leq m$), and $N_j^i = (n_{j,1}^i, n_{j,2}^i, ..., n_{j,j}^i)$ ($j \leq s(\mathbf{L_i})$). Moreover, let us use $\alpha(i)$ to denote he extreme number $n_{s(\mathbf{L_i}),s(\mathbf{L_i})}^i$ of message $\mathbf{L_i}$. Node X calculates **myL** according to the following rules:

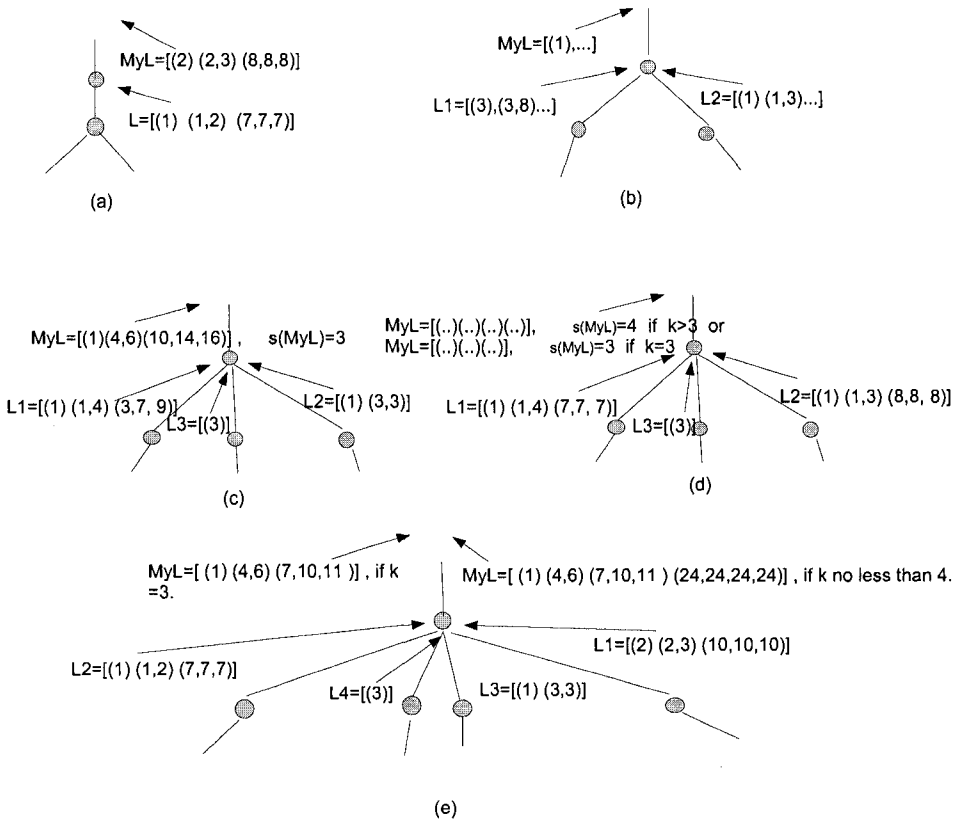

Figure 4.2: Compute Biggest node number

Calculation of $r = s(\mathbf{myL})$: the size of **myL**

- If $s(\mathbf{L_1}) > s(\mathbf{L_2})$, the size of **myL** is equal to the biggest message size X has received, $s(\mathbf{myL}) = \mathbf{s(L_1)}$. For example, in Figure 4.2 (c) $\mathbf{myL} = [N_1, N_2, N_3]$.

- If $s(\mathbf{L_1}) = s(\mathbf{L_2})$, it means there is more than one message with the biggest message size. If $s(\mathbf{L_1}) \leq (k-1)$, then the message size of **myL** is equal to the biggest message size X has received incremented by 1, $r = s(\mathbf{L_1}) + 1$; otherwise $r = s(\mathbf{L_1})$; e.g. Figure 4.2 (d).

Calculation of component N_j of $\mathbf{myL}(1 \leq j \leq r)$

In the following, we denote by N_j^i the j- component of vector $\mathbf{L_i}$, while N_j denotes the j- component of vector **myL**.

Since X has more than one child and one agent can only clean and guard X, we have that $N_1 = (n_{1,1}) = (1)$, e.g. Figure 4.2 (b). Let us now see how to compute $N_j(1 < j \leq r)$:

a) If the message size of **myL** is bigger than all received message size, we have enough agents to clean all subtrees of X. So, each component of N_r is equal to 1 plus the sum of all extreme numbers of the received messages. Let $N_r = [n_{r,1}, n_{r,2}, ..., n_{r,r-1}, n_{r,r}]$, then $\forall i \leq r \; n_{r,i} = \sum_{i=1}^{m} \alpha(i) + 1$ ($\alpha(i)$ is the extreme number from message $\mathbf{L_i}$).

b) If X receives only one message $\mathbf{L_1}$ whose message size is $\geq j$, it means j agents from X can clean all subtrees of node X except for the subtree from which

$\mathbf{L_1}$ is received. After that, j agents can go to the root of this subtree to clean same N_j^1 nodes. So, N_j is computed by increment by 1 to all components of N_j^1 and adding the sum of the extreme number of the messages whose message size is less than j, that is:

$$N_j = N_j^1 + \sum_{i=2}^{m} \alpha(i) + 1 = [n_{j,1}^1 + \sum_{i=2}^{m} \alpha(i) + 1, n_{j,2}^1 + \sum_{i=2}^{m} \alpha(i) + 1, ..., n_{j,j}^1 + \sum_{i=2}^{m} \alpha(i) + 1]$$

For example, in Figure 4.2(c), $N_3^1 = (3, 7, 9), \sum_{i=2}^{m} \alpha(i) + 1 = 3 + 3 + 1 = 7$, $N_3 = (10, 14, 16)$.

c) If X receives more than one message $\mathbf{L_i}$ whose message size is $\geq j$, we have to compute each component in vector $N_j (1 < j \leq r)$ respectively according to the following rules:

- The value $n_{j,1}$ is equal to the sum of the number of nodes in the subtrees which need less than j agents to be totally decontaminated incremented by 1.

  $$n_{j,1} = \sum \alpha(i) + 1$$

  $\forall i$ such that $s(\mathbf{L_m}) \leq s(\mathbf{L_i}) \leq j$.

  For example, in Figure 4.2(e), $n_{3,1}$ is equal to the sum of extreme numbers in message $\mathbf{L_3}$ and $\mathbf{L_4}$ plus 1, equal to $3 + 3 + 1 = 7$.

- Let $\mathcal{S}$ be the set of indices of the $\mathbf{L_i}$'s whose size is greater than or equal to j (i.e., $\mathcal{S} = \{i : s(\mathbf{L_i}) \geq j\}$). The value of $n_{j,h}$ in N_j $(1 < j \leq r,\ 1 <$

$h \leq j$) is calculated by exhaustively trying all possible feasible combinations of decontamination strategies that can be performed in the subtrees of X, given that j agents are available and h can be left down in the subtrees to possibly guard some decontaminated portions. The number of decontaminated nodes can be calculated in two parts: first part is the total number of nodes in those subtrees which need less than j agents to be fully decontaminated; the second part is the number of nodes decontaminated in those subtrees which need more than $j-1$ agents to be fully decontaminated. To calculate the number of nodes in the second part, for each subtree corresponding to an $\mathbf{L_i}$ ($i \in S$), we consider the size $n^i_{p_i,q_i}$ ($p_i < j$, $q_i < h$) of the area that can be contaminated with p_i agents leaving q_i in the subtree; for all possible feasible combinations of p_i, q_i, the combination that allows the highest decontamination will be chosen.

A set $\mathcal{S}' = \{n^i_{p_i,q_i}\}$ (with $i \in \mathcal{S}$) is *feasible* if and only if:

1- $p_i < j, q_i < h$

2- $\sum q_i = \begin{cases} h & \text{If} |\mathcal{S}'| = |\mathcal{S}| \\ h-1 & |\mathcal{S}'| < |\mathcal{S}| \end{cases}$

3- there exists a permutation of the elements of $\mathcal{S}'$: $n_{a_1,b_1}, \ldots, n_{a_{s'},b_{s'}}$ (where $s' = |\mathcal{S}'|$) such that:

$$j - \sum_{l=1}^{x} b_l \geq a_{x+1} \qquad 1 \leq x < s'$$

Given a feasible set $\{n^i_{p_i,q_i}\}$, we compute $n_{j,h}$ as follows:

$$n_{j,h} = \sum_{f:s(\mathbf{L_f})<j} \alpha(f) + Max_{p_i,q_i}\{\sum_{i\in\mathcal{S}} n^i_{p_i,q_i}\} + 1$$

$(1 < h \leq j \leq r)$

This procedure will be called: FIND-MAX-FEASIBLE-COMBINATION in the preprocessing algorithm.

Consider for example, Figure 4.2 (e). To calculate $n_{3,2}$ in **myL**, first we calculate the total number of nodes in the subtrees that need less than 3 agents to be fully decontaminated. This number is the sum of extreme numbers from $\mathbf{L_3}$ and $\mathbf{L_4}$, it is equal to $\sum_{f:s(\mathbf{L_f})<3} \alpha(f) = 6$. Then we need to calculate the number of nodes in the subtrees which need more than 2 agents to be fully decontaminated. Here we have $S = \{1, 2\}$. From all $n^i_{p_i,q_i}$ $i \in S$, we need to find the feasible combination that allows the highest decontamination. In this case $Max_{p_i,q_i}\{\sum_{i\in\mathcal{S}} n^i_{p_i,q_i}\} = n^1_{2,1} + n^2_{2,1} = 3$. So $n_{3,2} = 6 + 3 + 1 = 10$.

Explanation of Feasibility. Let us use an example to better understand the definition of feasibility. When we need to compute $n_{j,h}$ we have to use j agents to perform the decontamination of the subtrees of X and keeping h agents to guard X and its subtrees. The first two constraints state that we need a set $\mathcal{S}' = \{n^i_{p_i,q_i}\}$ (with $i \in \mathcal{S}$) such that either h agents are left behind to guard the subtrees ($\sum q_i = h$) or $h-1$ are left behind because

one is needed to stay in X ($\sum q_i = h - 1$). This second case occurs when some subtree is kept fully contaminated and thus X needs an agent to guard from it. The third constraint insures that the combination of numbers are sufficient. That is, there must be a scheduling for the subtrees decontamination such that, after leaving behind the prescribed number of agents to clean some subtrees, there are still enough to proceed with the decontamination of the others. For example, assume that we want to calculate $n_{7,6}$ for some node X, and X has three subtrees which need more than 6 agents to be fully decontaminated. Suppose that X has received $N_6^1 = (1, 2, 6, 7, 9, 13)$ from $\mathbf{L_1}$, $N_5^2 = (1, 4, 5, 6, 9)$ from $\mathbf{L_2}$, $N_2^3 = (1, 3)$ from $\mathbf{L_3}$. The set S' containing $\{n_{6,3}^1, n_{5,2}^2, n_{2,1}^3\}$ satisfies the first two constraints only and gives the biggest sum of total 11 nodes. However, we can never allocate available agents to decontaminate those parts in the subtrees, because to clean $n_{6,3}^1$ nodes, we have to use 6 agents; to clean $n_{5,2}^2$ nodes, we have to use 5 agents; no matter which subtree is chosen to be decontaminated first, after leaving behind the guarding agents we do not have enough agents remaining to decontaminate the other subtrees. This set does not respect the third constraint: in fact, for $n_{p_i,q_i} \in (n_{6,3}^1, n_{5,2}^2)$, $j - p_1(p_2) < q_2(q_1)$; for any set S' containing $(n_{6,3}^1, n_{5,2}^2)$, there is no permutation of this set to satisfy the third constraint. Instead of choosing the set $\{n_{6,3}^1, n_{5,2}^2, n_{2,1}^3\}$, we choose $\{n_{6,3}^1, n_{5,1}^2, n_{2,2}^3\}$. The sum of this set is 10 and it is a feasible one. For this combination, we can allocate available agents to clean those parts in subtrees.

The formal description of the algorithm is provided in Algorithm 4.1.

Algorithm 4.1 Preprocessing

States S{Available, Active, Processing, Saturated}

Available

Spontaneously

homebase send (Active) to N(x);
homebase become Processing;

Receiving(Activate)

send(Activate) to N(x)-sender;
Neighbours:=N(x);
if $|Neighbours|$=1 *then*
$\mathbf{L} = N_1 = 1$;
$parent \Longleftarrow Neighbours$;
send($\mathbf{L}$) to parent;
become Processing;
else become Active

Active

Receiving($\mathbf{L}$)

Neighbours:=Neighbours-sender;
if $|Neighbours|$=1 *then*
myL=*ComputeBig()*;
$parent \Longleftarrow Neighbours$;
send(**myL**) to parent;
become Processing;

Processing

Receiving($\mathbf{L}$)

Neighbours:=Neighbours-sender;
if $|Neighbours|$=0 *then*
myL=*ComputeBig()*;
become Saturated;

The procedure ComputingBig() is to calculate **myL** for each node.

ComputeBig()

Case1: Receiving one message $\mathbf{L}$, *then*
$\mathbf{myL} = \mathbf{L} + \underline{1}$ ($\mathbf{L} + \underline{1}$ *means adding 1 to all numbers in the vector* $\mathbf{L}$) ;
Case 2: receiving $\mathbf{L_1}$, $\mathbf{L_2}$, ..., $\mathbf{L_m}$ ($s(\mathbf{L_1}) \geq s(\mathbf{L_2}) \geq ... \geq s(\mathbf{L_m})$), *then*
$N_1 = (n_{1,1}) = 1$;
$\mathbf{myL} \leftarrow N_1$;
if $s(\mathbf{L_1}) > s(\mathbf{L_2}) \geq ... \geq s(\mathbf{L_m})$, *then*
$r = s(\mathbf{L_1})$ *(The message size of* **myL** *is equal to* $\mathbf{L_1}$*)*;
if $s(\mathbf{L_1}) = s(\mathbf{L_2}) \geq ... \geq s(\mathbf{L_m})$, *then*
if $(s(\mathbf{L_1}) + 1) \leq k$, *then*
$r = s(\mathbf{L_1}) + 1$ *(The message size of* **myL** *is equal to* $s(\mathbf{L_1})$ *plus 1)*;
else $r = s(\mathbf{L_1})$;
j=2;
While $j \leq r$
$a1 = \sum \alpha(i) + 1$, $\forall i$ *such that* $s(\mathbf{L_m}) \leq s(\mathbf{L_i}) < j$; (*here* $\alpha(i)$ *is the extreme number from message* $\mathbf{L_i}$)

if $s(\mathbf{L_1}) < j$ *then* *(All message size is smaller than j and $j = r$.)*
  $n_{r,1} = n_{r,2} = ... = n_{r,(r-1)} = n_{r,r} = a1$;
  $N_r = [n_{r,1}, n_{r,2}, ..., n_{r,(r-1)}, n_{r,r}]$;
else *(There are some message size $\geq j$.)*
  if only one message $\mathbf{L_1}$ has size $\geq j$, *then*
    $N_j = N_j^1 + \sum_{i=2}^{m} \alpha(i) + 1$
  else
    $n_{j,1} = a1$;
    $N_j \leftarrow n_{j,1}$;
    h=2;
    While $h \leq j$
        $n_{j,h}$ = FIND-MAX-FEASIBLE-COMBINATION;
        $N_j \leftarrow n_{j,h}$;
        $h+ = 1$;
myL $\leftarrow N_j$;
$j+ = 1$;

Let us consider again the tree of Figure 4.1. We showed before that the greedy algorithm could not give an optimal decontamination. Let us observe the steps taken by Algorithm 4.1 instead. The root of S_1 sends X the following information: $[(1), (6, 7), (6, 7, 8), (20, 20, 20, 20)]$. Here 1 means that if we use 1 agent to clean and leave 1 agent to guard we can clean 1 node in S1; $(6, 7)$ means that if we use 2 agents to clean and use 2 agents to guard we can clean 7 nodes in S_1; if we use 2 agents to clean and after clean procedure we leave 1 agent to guard then we can clean 6 nodes in S_1; and so on. In this way, node X receives three identical messages from S_1, S_2 and S_3, and a different message $[(1), (1, 3)]$ from S_4. After executing Algorithm 4.3, node X calculates the new message **myL** as $[(1), (1, 2), (4, 10, 12), (4, 10, 22, 23), (63, 63, 63, 63, 63)]$ (if $k > 4$) or $[(1), (1, 2), (4, 10, 12), (4, 10, 22, 23)]$ (if $k = 4$). Then node X sends message **myL** to its parent. As a consequence, from X, the local information shows 23 nodes could be

decontaminated with 4 agents.

Decontamination.

When the homebase has all the necessary information, it means that all nodes have the information for their subtrees; the agents can now choose the best movement strategy. The necessary information is written on the local whiteboard of the node.

The decontamination process is an iteration process. The k agents from the homebase first decontaminate the subtrees that need less than k agents to be fully decontaminated. If there is only one subtree left, all available agents will move to the root of that subtree to repeat the same process; if there are more than one subtrees remaining, according to the Max-Feasible-Combination set S', find the permutation of the elements of $\mathcal{S}'$: $n_{a_1,b_1}, \ldots, n_{a_{s'},b_{s'}}$ (where $s' = |\mathcal{S}'|$) such that: $j - \sum_{l=1}^{x} b_l \geq a_{x+1} \quad 1 \leq x < s'$. Send a_1 agents to the root of the subtree whose message **L** contains n_{a_1,b_1}. After cleaning this subtree, leave b_1 agents to guard and return the remaining agents to the homebase. Repeat the same process, at last $a_{s'}$ agents are sent to the subtree whose message **L** contains $n_{a_{s'},b_{s'}}$.

The formal description of the algorithm is provided in Algorithm 4.2.

Algorithm 4.2 Disinfecting an arbitrary tree with k agents from homebase

---

Saturated

$\alpha = \phi$;
CleanUp(X,s,t) *(X is homebase, s is the number of available agents on X and t is the number of guarding agents on X and all its subtrees.)*

CleanUp(X,s,t)

While there are removable agents on node X
From X, clean all subtrees with message size $< |s|$;
$\alpha = \alpha \cup$ cleaned nodes $\cup$ X;
Case 1: There is only one subtree left with message size $> |s|$
All agents on X move to the root Y of the subtree;
CleanUp(Y,s,t);
Case 2: There are more than one subtrees left with message size $\geq |s|$
Find all $n^f_{p,q}$ which consist $n_{s,t}$ in message **myL** of X, let those $n^f_{p,q}$ be a set V;
array1[ ]=*Sort(V,|s|)*;
i=size(array1[ ]);
While $i \geq 1$
Move p agents which is associated with $n^f_{p,q}$ in array1[i] to the root Y of the subtree from whose message the number $n^f_{p,q}$ is chosen;
CleanUp(Y,p,q);
Return back $p - q$ number of agents to X;
$i- = 1$
Return α;

Sort(V,$|s|$)
j=1;
While $j \leq$ size(V)
choose one $n^e_{p,q}$ from V which is satisfied with $p + \sum_{n^x_{p',q'}, x \neq e} q' \leq j$
Array1[j]:=$n^e_{p,q}$;
$j+ = 1$;
V:=V-$n^e_{p,q}$;
Return Array[ ];

Theorem 33. *In Algorithm 4.1, when a node finishes to calculate message* $N_j = (n_{j,1}, n_{j,2}, ..., n_{j,(j-1)}, n_{j,j})$ $(j > 1)$, $n_{j,i}$ *contains the the biggest number of nodes that can be decontaminated with* j *agents and leaving* i *agents to guard.*

PROOF

1. Base Case

In Algorithm 4.1 every leaf send $\mathbf{L} = [N_1] = [(n_{1,1})] = 1$ to its parent. Form every leaf, one agent can only clean itself. So this theorem is correct for j=1.

2. Assumption Step

let us now assume that when a node X receiving message $\mathbf{L_1}$, $\mathbf{L_2}$, ..., $\mathbf{L_m}$ ($s(\mathbf{L_1}) \geq s(\mathbf{L_2}) \geq ... \geq s(\mathbf{L_m})$) from all children and all vectors $N_j^i = (n_{j,1}^i, n_{j,2}^i, ..., n_{j,(j-1)}^i, n_{j,j}^i)$ ($j > 1, s(\mathbf{L_m}) \leq s(\mathbf{L_i}) \leq s(\mathbf{L_1})$) in those message stand for the biggest number of clean nodes.

3. Induction Step

Now we want to prove $N_j = (n_{j,1}, n_{j,2}, ..., n_{j,(j-1)}, n_{j,j})$ ($j > 1$) in message **myL** calculated by node X contains the biggest number of clean nodes from node X and all its subtrees with i agents leaving to guard($i = 1, ..., j$). There are different possible cases to calculate **myL**: 1) Node X has only one subtree. In this case, all agents can move to the root of the subtree without any node guarding X. So the clean number with j agents from node X is the clean number with j agents in its subtree plus X itself. In this case, the node X will receive one message **L**. According to Algorithm 4.1, the **myL** calculated by X is equal to **L** adding 1 to every number in all vectors. So in this case the $N_j = (n_{j,1}, n_{j,2}, ..., n_{j,(j-1)}, n_{j,j})$ ($j > 1$) in **myL** stands for the biggest clean node number. 2) Node X has more than one subtree. In this case, the biggest clean number with j agents from node X is equal to the sum of the number of nodes in its subtrees which require less than j agents to clean, plus the biggest number of nodes cleaned with j agents in remaining subtrees which require no less than j agents to clean, and plus the node X itself. To

calculate the biggest nodes number cleaned with j agents in remaining subtrees, there are two different cases. a) The first case is that there is only one subtree left which requires more than j agents to clean. In this case, that biggest clean number in the remaining subtree is equal to the biggest number of nodes cleaned with j agents in this subtree. In Algorithm 4.1, in this case, node X will receive only one message $\mathbf{L_1}$ ($s(\mathbf{L_1}) > j$). We find the $N_j^1 = (n_{j,1}^1, n_{j,2}^1, ..., n_{j,(j-1)}^1, n_{j,j}^1)$ in message $\mathbf{L_1}$. Since $N_j^1 = (n_{j,1}^1, n_{j,2}^1, ..., n_{j,(j-1)}^1, n_{j,j}^1)$ stands for the biggest number of nodes in this subtree, then after adding each number of $N_j^1 = (n_{j,1}^1, n_{j,2}^1, ..., n_{j,(j-1)}^1, n_{j,j}^1)$ with sum a1, the new N_j in message **myL** stands for the biggest node number from X. b) The second case is that there are more than one subtree left which requires no less than j agents to be fully decontaminated. In this case, we need to compare every possible situation to allocate those j agents to each of those subtrees or to use one agent to guard node X and then allocate $j-1$ agents to some of those subtrees. After we compare the number of nodes cleaned with each possible allocation, the allocation case that hold the biggest clean number of nodes is the optimal allocation of j agents to clean the maximum number of nodes. In Algorithm 4.1, the node will receive messages $\mathbf{L_1}$,$\mathbf{L_2}$,...,$\mathbf{L_i}$ ($s(\mathbf{L_i}) \geq j$). We compare all number combination to find the Max-Feasible-Combination set S', which stands for the biggest decontaminated nodes' number in those subtrees. So the N_j in message **myL** stands for the biggest node number from X. By now we have proved $N_j = (n_{j,1}, n_{j,2}, ..., n_{j,(j-1)}, n_{j,j})$ in message **myL** stands for the biggest clean node number with j agents to clean. ∎

Theorem 34. *Algorithm 4.2 cleans the biggest number of nodes in an arbitrary tree.*

PROOF

In Algorithm 4.2, after the homebase receives all messages from its subtrees, it knows how many agents are needed to clean the subtrees and can calculate its own **myL**. According to the last theorem, the homebase can know for all values of H ($H \leq K$) and S ($S \leq H$), how many nodes can be decontaminated from homebase in each subtree using H agents and leaving S to guard. At first some agents move to clean all the subtrees that require less than k agents to be decontaminated. After that, all k agents return to homebase. If there is only one subtree left, k agents will move to the root of that subtree and repeat the clean procedure above again. If there are more than one subtrees left, according to Algorithm 4.1, the homebase can calculate the biggest number of decontaminated nodes with k agents. The Max-Feasible set $S' = \{n^f_{p,q}\}$ that consists in the extreme number of **myL** of homebase stands for the biggest clean part in those remaining subtrees. In the sorting function, the homebase first find one $n^f_{p,q}$ whose $p + \sum_{n^x_{p',q'}, x \neq f} q' \leq j$. It means after decontaminating all other subtrees, we still have available agents to clean this $n^f_{p,q}$. So, this subtree can be decontaminated at last. Then the sorting function finds the second last decontaminated subtree, etc. After sorting completed, the homebase can send p agents to the root of the subtree that should be decontaminated at first. Once p agents arrive on the root of this subtree, the same procedure will be executed as before. After the biggest part of this subtree has been decontaminated, $p - q$ agents will return to the homebase and the homebase will send required agents to the second subtree. This process will stop until no agents can move. ∎

Theorem 35. *In Algorithm 4.1, the number of messages for computing the preprocessing phase is $O(n)$. The size of a message is $O(k^2 \log n)$. The total size of all messages is $O(nk^2 \log n)$ bits.*

PROOF

Every node except the leaf sends an activation message to its children once, so there are $O(n)$ activation messages in total. Every node except the homebase sends one message **myL** to its parent, so there are $n-1$ **myL** messages. Each message contains vectors $[N_1, N_2, ..., N_{r-1}, N_r]$ and each N_i contains i components $(n_{i,1}, n_{i,2}, ..., n_{i,(i-1)}, n_{i,i})$. So there are $\sum_{i=1}^{r} i = \frac{r(r+1)}{2}$ elements in each message. For k agents problem, r is no more than k. So, there are $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ elements in each message. Each of $n_{i,j}$ value is smaller than n, so it contains no more than $\log n$ bits. The total size in all messages is no more than $n * k(k+1)/2 * \log n$, equal to $O(nk^2 \log n)$. ∎

Notice that k is at most $\log n$, so the complexity never exceeds $O(n \log^3 n)$.

4.2 Disinfecting Complete Trees

In this section, we study the optimal strategy to decontaminate a complete tree. Since the complete tree has its unique structure and each node knows the structure of all its subtrees naturally, we do not need to find a way to express the structure of tree. So, we have a much simpler algorithm to decontaminate the biggest number of nodes.

Before we give that algorithm, let us define the notion of *level*. The level of a node is

equal to its distance to the closest leaf. All leaves are at level 0 and the level of the root is the height of the tree.

Lemma 5. *No deterministic algorithm can clean more than $2k-1$ nodes in a complete binary tree if the homebase is located on a node with level greater than $k-1$.*

PROOF

If the homebase is located on a node with level greater than $k-1$, this node has two subtrees each needs at least k agents to be fully decontaminated [4]. k agents are not enough to clean one subtree because one agent has to guard the homebase. If we use k agents to clean the tree from the homebase, all clean nodes will be on level greater or equal to 1. Now let us consider the clean part in the complete binary tree, the clean part will be a subgraph of the complete binary tree, and thus a binary tree. In this clean binary tree, all leaves are guarded by agents because all leaves are at level greater or equal to 1 in the complete binary tree and connected with contaminated children nodes. Each clean node must have two clean or guarded children. If a node has only one child in this clean binary tree, the node must be guarded because it has another contaminated neighbour in the complete binary tree. To construct such a binary tree, it is easy to see that the number of clean nodes must be less than the number of guarded nodes. We have k guarded nodes, so the total number of decontaminated nodes is no more than $2k-1$. ∎

Theorem 36. *No deterministic algorithm can clean more than 2^k+2k-1 nodes in complete binary tree.*

PROOF

From the last lemma we have proved that if the homebase is located on a node with level greater than $k-1$, the biggest number of decontaminated nodes is no more than $2k-1$ using k agents. Let us now consider the case where the homebase is located on a node with level less than or equal to $k-1$. The homebase has two subtrees which need no more than $k-1$ agents to be cleaned, we can leave one agent to guard the homebase and move the other agents to clean the two subtrees. After that, k agents can move to the parent of the homebase and repeat the clean procedure again until k agents reach a node with level k. At this point, k agents can clean no more than $2k$ nodes in the left part of the complete binary tree (one more node than the number of decontaminated nodes in previous lemma, because now one subtree of node with level k is fully decontaminated, and the node with level k has only one contaminated child node.). The node with level $k-1$ has 2^k-2 nodes in its two subtrees. So, the biggest number of clean nodes contains the two subtrees of the node with level $k-1$ plus the the node itself and nodes cleaned in the left part of the tree, that is 2^k+2k-1. ∎

Decontamination strategy

If the homebase is located on a node with level greater than $k-1$, in the first step, one agent guards the homebase and the remaining $k-1$ agents move to the parent node. In the second step, when they all reach the parent, $k-2$ agents continue to move "up" and the remaining agent moves to another uncleaned neighboring node. In step i, when $k-i$ agents reach a node, $k-i-1$ agents will move "up" and the remaining agent moves to

another uncleaned neighboring node. This process is repeated until no agents can move. If the agents can move across the root of the tree, those agents can go directly to another uncleaned child of the root without keeping one agent to guard the root. In this case, we can clean one extra node.

If the homebase is located on a node with level less or equal to $k-1$, then at first some agents move to clean the two subtrees of the homebase (we know that this number of agents is less than k from [4].) and then all agents move to the parent of the homebase. The clean procedure repeats until k agents reach a node with level k. At this point, k agents continue the decontamination as described above, as in Figure 4.3 (a).

The formal description of the algorithm is provided in Algorithm 4.3.

Algorithm 4.3 Disinfecting a complete binary tree with k agents from homebase

---

```
if homebase with level i < k
  send i agents to clean two subtrees of homebase;
  return back i agents to homebase;
  for j = 1 to k − i, j++
      k agents move to node with level i + j;
      send i + j agents to clean uncleaned subtree of this node with level i + j;
      return back i + j agents to this node;
for j = 1 to k − 1, j++
  If the node is the root(deg(v)=2) of this complete binary tree, then
      move all agents on this root to its uncleaned child node;
  else
      if the up level node is uncleaned then
        Move k − j agents to the up level node;
        Move remaining one agent to clean and guard one uncleaned child of this node;
      else
        Move k − j agents to one child node;
        Move remaining one agent to clean and guard another child of this node;
```

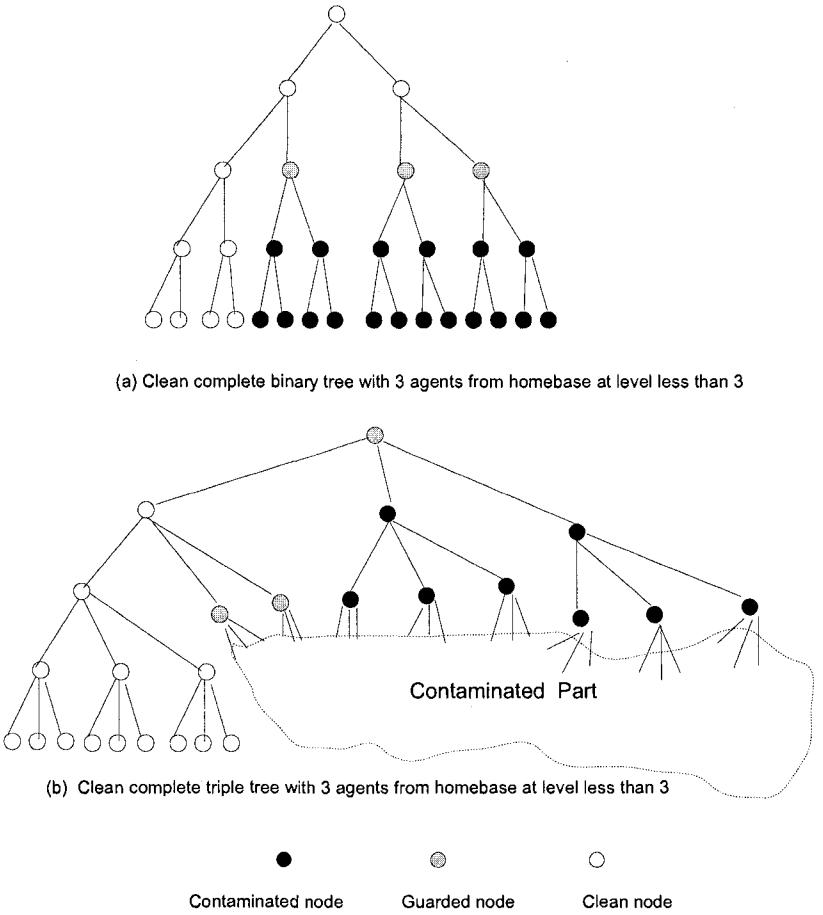

Figure 4.3: Clean Complete Tree

Theorem 37. *The number of nodes decontaminated by Algorithm 4.3 is* $2^k + 2k - 1$ *if the height of the complete binary tree is less than* $2k - 1$*; the number of nodes decontaminated by Algorithm 4.3 is* $2^k + 2k - 2$ *if the height of the complete binary tree is greater or equal to* $2k - 1$.

PROOF

The biggest number of nodes decontaminated by Algorithm 4.3 can be achieved when the homebase is located on a node with level less or equal to $k - 1$. In this case, the number of decontaminated nodes is equal to the number of nodes in the two subtrees of the node with level $k - 1$, plus itself, and the nodes cleaned in the remaining part of the complete binary tree. The node with level $k - 1$ has $2^k - 2$ nodes in its two subtrees. For the clean binary tree in the remaining part of the complete binary tree, every leaf of this binary tree will be guarded by one agent. There are in total k leaves, so the total number of nodes in that clean binary tree is $2k - 1$. The number of nodes decontaminated by Algorithm 4.3 is $2^k + 2k - 2$. If the height of the tree is less than $2k - 1$, we can move agents across the root of the tree without keeping one agent to guard this root. In this case we can clean one extra node and the number of clean nodes in the binary tree is $2k$, so the biggest number of nodes decontaminated by Algorithm 4.3 is $2^k + 2k - 1$. ∎

Theorem 38. *If the homebase is on a node with level* $k - 1$*, Algorithm 4.3 executes at most* $\sum_{i=1}^{k-1} (2 * (k - i) * 2^i) + k(k + 1)/2 + 2(k - 1)$ *moves.*

PROOF

If the homebase is located on a node with level $k-1$, each edge connected with two nodes with level $i-1$ and i on the subtree of the homebase which needs less than k agents to be decontaminated will be visited by i agents twice: going forward and returning. There are 2^{k-i} edges connected with two nodes with level $i-1$and i. So, the total number of moves on these edges is $\sum_{i=1}^{k-1} 2*(k-i)*2^i$. After that, the k agents will go to clean other nodes and leave one agent to guard one node each time. The total number of moves in this phase is $k(k+1)/2+k-1$. If the tree has a height smaller than $2k-1$, we need to add at most $k-1$ moves across root. So the total number of moves is $\sum_{i=1}^{k-1}(2*(k-i)*2^i)+k(k+1)/2+2(k-1)$. ∎

Theorem 39. *If the homebase is on a node with level $k-1$, the time complexity is at most $2^{k+1}+k-5$.*

PROOF

To clean the subtrees from a homebase with level $k-1$, each edge will spend 2 time units. There are 2^k-2 edges in a complete binary tree connected with nodes at level $k-2$ and $k-1$, so $2^{k+1}-4$ time units are needed to clean all the edges. After that, k agents will move to clean a binary tree with k leaves. It will cost one time unit to occupy one leaf of the tree, so the number of time units in this process is k. If the height of the complete binary tree is less than $2k-1$, one more time unit is needed to cross the root. The total number of time units is at most $2^{k+1}+k-5$. ∎

Disinfecting a complete i-tree

We now generalize our results for the complete binary tree to a complete i-tree where every node has $deg(v) = i + 1$ except for root with $deg(v) = i$ and leaf with $deg(v) = 1$.

Theorem 40. *No deterministic algorithm can clean more than* $\frac{i^k-1}{i-1} + \lfloor \frac{k-1}{i-1} \rfloor + k$ *nodes in a complete i-tree.*

PROOF

Similarly to the theorem for the complete binary tree, if the hombase is on a node with level more than $k - 1$, k agents are not enough to fully clean each subtree of the homebase. It is easy to see that the biggest clean part with k agents from this homebase is a i-tree. This i-tree will use k agents to guard from the contaminated part. The clean nodes are those nodes whose i children nodes are clean or guarded. If we don't consider the degree related to the contaminated part for each node in this i-tree, the degree of clean node would be $i + 1$ except for the root with degree i and the degree of guarded node would be equal or less than i. The number of guarded node is equal to k. When the i-tree only includes nodes (clean nodes) with degree equal to $i + 1$ and nodes (leaves) with degree equal to 1 and node (root) with degree i, the number of clean nodes will be maximum. In this case, the number of nodes of i-tree is $\lfloor \frac{k-1}{i-1} \rfloor + k$.

If the homebase is on a node with level less or equal to $k - 1$, the subtrees of the homebase can be decontaminated with no more than k agents. In this case, after the decontamination of subtrees, all agents can move to the parent of the homebase to execute the same process. This procedure will repeat until all agents move to a node with level

k. The number of nodes of complete i-tree with level $k-1$ are $\frac{i^k-1}{i-1}$. So the maximum number of clean nodes will be $\frac{i^k-1}{i-1} + \lfloor \frac{k-1}{i-1} \rfloor + k$ if the homebase is on a node with level less or equal to $k-1$. ∎

Decontamination strategy

If the homebase is on a node with level more than $k-1$, at step 1, we move i agents to clean and guard those i child nodes of the homebase and then move $k-i$ agents to the parent of the homebase. At step j, we move i agents to clean and guard those i child nodes of the new homebase and then move $k-i \times j$ agents to the parent of the homebase. This process will repeat until no agents can move.

If the homebase is on a node with level less or equal to $k-1$, at first needed agents will clean the subtrees of the homebase and then all agents will move to the parent of the homebase. This process will repeat until all agents move to a node with level k. At this point, agents will execute decontamination process as same as the first part, as in Figure 4.3 (b) if $i = 3$.

The formal description of the algorithm is provided in Algorithm 4.4.

Algorithm 4.4 Disinfecting a complete i-tree with k agents from homebase

---

if homebase with level $h < k$
  for $j = 1$ to i, $j++$
    send h agents to clean one subtree;
    return back h agents to homebase;
  for $j = 1$ to $k-h$,$j++$
    k agents move to node with level $h+j$;
    for $l = 1$ to $i-1$, $l++$
      send $h+j$ agents to clean one uncleaned subtree of node with level $h+j$;
      return back $h+j$ to this node;
for $j = 1$ to $\lfloor \frac{k-1}{i-1} \rfloor$, $j++$
  if the up level node is uncleaned then

send $i-1$ agents to guard those $i-1$ uncleaned children of node with level $k+j$;
send $k-(i-1)*j$ agents to the parent of the node with level $k+j$;
else
send $i-1$ agents to guard $i-1$ uncleaned children of this node with level $k+j$;
send $k-(i-1)*j$ agents the remaining uncleaned child node;

Theorem 41. *The number of nodes decontaminated by Algorithm 4.4 is* $\frac{i^k-1}{i-1}+\lfloor\frac{k-1}{i-1}\rfloor+k$.

PROOF

In the Algorithm 4.4, if the homebase is on a node with level less or equal to $k-1$, the biggest number of clean nodes is equal to the number of nodes in a complete i-tree with level $k-1$ plus the number of nodes in a i-tree with k leaves. In this i-tree, all guarded nodes has degree equal to 1. So the total number of nodes decontaminated by Algorithm 4.4 is $\frac{i^k-1}{i-1}+\lfloor\frac{k-1}{i-1}\rfloor+k$. ∎

Theorem 42. *If the homebase is on a node with level* $k-1$, *Algorithm 4.4 executes at most* $\sum_{j=1}^{k-1}(2*(k-j)*i^j)+\lfloor\frac{k}{i}\rfloor(k+1)/2+k$ *moves.*

PROOF

There are i^{k-j} edges connected with nodes at level $j-1$ and j and each of them will visited by j agents twice. So the total number of moves on those edges in a complete i-tree at level $k-1$ are $\sum_{j=1}^{k-1}2*(k-j)*i^j$. After that, the k agents will go to clean other nodes and leave one agent to guard one node each time. The total moving of this stage is $\lfloor\frac{k}{i}\rfloor(k+1)/2+k$. So the total number of moves is $\sum_{j=1}^{k-1}(2*(k-j)*i^j)+\lfloor\frac{k}{i}\rfloor(k+1)/2+k$. ∎

Theorem 43. *If the home base is on a node with level $k-1$, the time complexity is at most $2 \times (i^k - i)/(i-1) + \lfloor \frac{k}{i} \rfloor + 1$.*

PROOF

To clean subtrees from a homebase at level $k-1$, each edge will cost 2 time units. There are i^{k-j} edges connected with nodes at level $k-2$ and $k-1$ in a complete i-tree, the total number of edges on a complete i-tree at level $k-1$ is $(i^k - i)/(i-1)$. So $2 \times (i^k - i)/(i-1)$ time units are needed to clean all edges. After that, in the process to decontaminate the i-tree, assuming that i agents can move concurrently to a node's i children nodes, the total time units in the process is $\lfloor \frac{k}{i} \rfloor + 1$. So the total number of time units is at most $2 \times (i^k - i)/(i-1) + \lfloor \frac{k}{i} \rfloor + 1$. ∎

4.3 Comparison of Complete Tree and General Tree

Lemma 6. *In a binary tree (every node has $deg(v) \leq 3$), if there is a node (not root of the tree) from which k agents are necessary to fully decontaminate its subtrees and the node itself, there are at least $2^k - 2$ nodes in its subtrees.*

PROOF

Base Case: for $k = 1$ agent, this lemma is obviously correct.

Assumption Step

Let us assume that for up to $k-1$ agents, this lemma is correct. If there is a node (not root of the tree) that needs $k-1$ agents to fully decontaminate its subtrees and itself,

there are at least $2^{k-1} - 2$ nodes in its subtrees.

Induction Step

In a binary tree, if there is a node X (not root of the tree) that needs k agents to fully decontaminate its subtrees and itself, there are only two cases: 1)X has two subtrees that need $k-1$ agents to be fully decontaminated. 2)X has one subtree that needs k agents to be fully decontaminated. In the first case, we know from the assumption that each subtree of X has at least $2^{k-1} - 1$ nodes (include the root of the subtree). So, the total number of nodes in the subtrees of X is at least $2 * (2^{k-1} - 1) = 2^k - 2$. In the second case, we can still find a node (a descendant of X) that has two subtrees which need $k-1$ agents to be fully decontaminated. In this case the total number of nodes will be no less than $2 \times (2^{k-1} - 2) + 2 = 2^k - 2$ too. ∎

Theorem 44. *The number of nodes that can be decontaminated in a binary tree with k agents is greater than or equal to the number of nodes that can be decontaminated in a complete binary tree.*

PROOF

For a complete binary tree, if the homebase is located on a node with level greater than $k-1$, the biggest clean part will be a binary tree with no more than $2k$ nodes (Lemma 5). In binary tree, if the homebase is located on a node which needs no less than k agents to clean its subtrees, we can move the agents in the same way as we move them in a complete binary tree. The only difference is that we do not need to keep an

agent to guard a node with one child only. If there is a node with $deg(v) = 2$, the agent can move across this node to its child node. So we can clean no less number of nodes in binary tree than in complete binary tree in this case. If the hombase is located on a node with level smaller than k, the biggest clean part in a complete binary tree will be a complete binary with $2^k - 1$ nodes plus a binary tree with no more than $2k$ nodes. To clean the binary tree, we can use k agents to clean a part containing no less than $2^k - 1$ nodes (by previous lemma). After that we can clean no less nodes than in a complete binary when the homebase is located on a node which need more than k agents to fully clean its subtrees. So the total number of clean nodes in the binary tree is no less than the number of nodes in complete binary tree. ∎

This theorem can not be generalized to a general tree in which some nodes have $deg(v) \geq 3$. *It is possible to decontaminate more nodes in a complete i-tree than in a general i-tree with k agents.* For a complete i-tree, the biggest number of clean nodes with k agents is $\Theta(i^k)$; for a general i-tree, the biggest number of clean nodes is no less than the number of clean nodes in a binary tree with k agents, equal to $\Omega(2^k)$, the exact number of clean nodes will depend on the structure of the tree.

Other interesting observations concern *the local model and the visibility model.* In the local model, we do not need one extra agent to act as the coordinator. When an agent arrives at a node, it waits if it is the first one to arrive. This agent can be the coordinator for the other agents that arrive at this node later. In the visibility model, the visibility can not improve the efficiency of the decontamination strategy. For an agent on a node,

there are only two ways to move: 1) moving to the parent of this node; 2) moving to the children of this node. In fact, to decontaminate a new node, there is only one way to choose: either moving to parent or to children. If the agent comes from the parent of this node, it can only move to the children of this node to decontaminate some nodes, and vice versa. Without visibility, each agent can still know when and how to move : 1) If there is only one path, all agents can move to that path directly; 2)If there are multiple paths, the agent first arriving at this node will wait and it will be the coordinator to direct other agents arriving later. Even without visibility, the first agent can know the state of all neighbouring nodes.

4.4 Summary

Table 4.1 contains the biggest number of clean nodes with k agents in different types of trees.

Tree Decontamination	
Network Topology	Biggest Number of Clean Nodes
Complete Binary Tree	$\Theta(2^k)$
Binary Tree	$\Omega(2^k)$
Complete i-Tree	$\Theta(i^k)$
General Tree	$\Omega(2^k)$

Table 4.1: Tree Decontamination

Chapter 5

Conclusions

5.1 Results

In this thesis, we have studied the best effort decontamination problem by mobile agents in different network topologies. Our aim is to find optimal strategies to decontaminate the biggest possible number of nodes with a fixed given number of agents. We have shown optimal strategies for meshes and trees, and we have analyzed their performances in terms of number of moves and time.

We now summarize the results of the previous chapters.

Meshes. From chapter two and three, we obtain the following conclusions. Our first conclusion is that, in the mesh, the biggest number of nodes that can be decontaminated with a fixed number of agents k depends on the position of the homebase. The number

of decontaminated nodes is listed in the following table.

Strategy	Maximum Number of Decontaminated Nodes
Homebase at Corner	$\Theta(k^2/2)$
Homebase at Border	$\Theta(k^2/4)$
Homebase at Inside Mesh	$\Theta(k^2/8)$

Our second conclusion is that the number of decontaminated nodes decreases while the degree of nodes increases. The biggest number of clean nodes in different mesh topology is listed in the following table.

Decontamination Strategy	Mesh	Hexagonal Mesh	Octagonal Mesh
Homebase at Corner	$\Theta(k^2/2)$	$\Theta(k^2/2)$	$\Theta(k^2/4)$
Homebase at Border	$\Theta(k^2/4)$	$\Theta(k^2/6)$	$\Theta(k^2/9)$
Homebase at Inside Mesh	$\Theta(k^2/8)$	$\Theta(k^2/14)$	$\Theta(k^2/16)$

From the table above we can see that the number of decontaminated nodes in the mesh is the biggest, and while octagonal mesh is the smallest among those three types of mesh structures.

The third conclusion about the mesh is that the power of visibility decreases the number of moves and the total time units while increasing the total number of clean nodes. The following table shows the efficiency difference between our local model strategies and visibility model strategies.

Decontamination Strategy	Number Of	Model	
		Local model	Visibility Model
Homebase at Corner	Moves	$O(3k^2/2)$	$O(k^2)$
	Time Units	$O(k^2/2)$	$O(2k)$
Homebase at Border	Moves	$O(3k^2/4)$	$O(k^2/2)$
	Time Units	$O(k^2/4)$	$O(k)$
Homebase inside Mesh	Moves	$O(3k^2/8)$	$O(k^2/4)$
	Time Units	$O(k^2/8)$	$O(k/2)$

Trees. An interesting observation is that the conclusions we obtain from the mesh do not apply to trees.

We did not find any advantage to have visibility power in a tree; we can not improve the efficiency of the decontamination in the visibility model. Furthermore, the biggest number of decontaminated node does not seem to decrease when the degree of the tree increases. On the contrary, we have observed that in complete trees it increases with the degree. As same as for the case of the mesh, also in the tree the number of decontaminated nodes depends on the location of the homebase.

In the case of the arbitrary tree we have described a distributed algorithm to determine, given a starting location, what is the optimal strategy. The algorithm exchanges $O(n)$ messages of size at most $O(k^2 \log n)$ bits. The decontamination itself is then very efficient since it require $O(n)$ moves and $O(n)$ time.

Table 5.1 summarizes the results on the biggest number of nodes that can be decontaminated with k agents in trees.

Tree Decontamination	
Network Topology	Maximum Number of Clean Nodes
Complete Binary Tree	$\Theta(2^k)$
Binary Tree	$\Omega(2^k)$
Complete i-Tree	$\Theta(i^k)$
General Tree	$\Omega(2^k)$

5.2 Future Work

The decontamination problem with a fixed number of mobile agents is very complicated. What we have done in this thesis is only moving a first step toward its solution; a lot of further work is needed to solve this problem in a more general setting. In the following we list some possible further studies:

- **Decontamination strategy for other network topology.**

  In this thesis, we studied the mesh and the tree. The study of other topologies is still open.

- **Protecting nodes.**

  In this thesis, we try to find some optimal strategies to clean as more as possible

nodes with fixed number of mobile agents and we do not consider cleaning some specific area of nodes in the network. An interesting variation of the problem would be the following: Some nodes are especially important and must absolutely be included in the decontaminated area: find a strategy to decontaminate as many nodes as possible with a fixed number of agents while making sure that those nodes are among the decontaminated ones. This new problem is similar to the decontamination problem discussed in this thesis but requires completely different approach. This problem could be practical and useful in real network environment.

- **Better preprocessing for the tree.**

  Our decontamination of the tree has a necessary preprocessing phase to determine the optimal strategy to clean the tree. Its bit complexity is quite high when the number of agents is high ($O(nk^2 \log n)$). We know that $k < \log n$, thus the bit complexity could reach $O(n \log^3 n)$. An open problem is to determine a more efficient preprocessing algorithm.

Bibliography

[1] M. Asaka, S. Okazawa, A. Taguchi, S. Goto. "A method of tracing intruders by use of mobile agent." *In 9th Annual Conference of the Internet Society, www.isoc.org, 1999.*

[2] S. Albers, M. Henzinger. "Exploring unknown environments." *29th Annu. ACM Sympos. Theory Comput. 416–425, 1997.*

[3] S.Alpern, S.Gal. "The Theory of Search Games and Rendezvous." *Kluwer, 2003.*

[4] L.Barrière, P.Flocchini, P.Fraigniaud, N.Santoro. "Capture of an Intruder by Mobile Agents. " *Proc. of 2002 Symposium on Parallel Algorithms and Architectures (SPAA 2002),200-209.*

[5] L. Barrière, P. Fraigniaud, N. Santoro and D.M. Thilikos. "Searching is not jumping." *Proceedings of the 29th International Workshop on Graph Theoretic Concepts in Computer Science (WG), Elspeet, the Netherlands, Springer Verlag, Lecture Notes in Computer Science 2880, 34-45, 2003.*

[6] L. Blin, P. Fraigniaud, N. Nisse, and S. Vial, Distributed chasing of network intruders by mobile agents, Proc 3th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Chester, UK, Lecture Notes in Computer Science, Vol. 4056, Springer, 2006, pp. 70–84.

[7] D. Bienstock and P. Seymour, Monotonicity in graph searching, Journal of Algorithms, 12 (1991) 239–245.

[8] R. Breish, "An intuitive approach to speleotopology." *Southwestern cavers, VI (5), 72-28, 1967.*

[9] C. Cooper, R. Klasing, T. Radzik. "Searching for black-hole faults in a network using multiple agents." *10th Int. Conf. on Principle of Distributed Systems (OPODIS), 2006.*

[10] D.G. Cornei and A. Proskurowski. "Complexity of findding embeddings in a k-tree." *SIAM Journal of Alg. Disc. Meth., 8, 277-284, 1987.*

[11] J. Czyzowicz, D. Kowalski, E. Markou, A. Pelc. "Searching for a black hole in tree networks." *8th Int. Conf. on Principle of Distributed Systems (OPODIS), 35-45, 2004.*

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

[13] A.Dessmark, A.Pelc. "Optimal graph exploration without good maps." *10th European Symp. on Algorithms (ESA), 374–386, 2002.*

[14] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro. 'Mobile search for a black hole in an anonymous ring." *Algorithmica, to appear.*

[15] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro. "Searching for a black hole in arbitrary networks: optimal mobile agents protocols." *Distributed Computing to appear.*

[16] X. Deng, C. H. Papadimitriou. "Exploring an unknown graph." *J. of Graph Theory 32(3),265–297, 1999.*

[17] J.Ellis,H.Sudborough, and J.Turner. "The vertex separation and search number of a graph." *information and Compution, 113(1):50-79,1994.*

[18] N. Foukia,J. G. Hulaas, J. Harms. "Intrusion Detection with Mobile Agents." *www.isoc.org, 2001.*

[19] P. Flocchini, M. J. Huang, F.L. Luccio,"Contiguous search in the hypercube for capturing an intruder." *Proc. of 18th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS 2005).*

[20] P. Flocchini, M. J. Huang, F.L. Luccio, "Decontamination of chordal rings and tori. " *Proc. of 8th Workshop on Advances in Parallel and Distributed Computational Models (APDCM 2006).*

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

[22] P.Fraigniaud, L.Gasieniec, D.Kowalski, A.Pelc. "Collective tree exploration." *LATIN 2004: Theoretical Informatics,141-151.*

[23] J. Gustedt. "On the pathwidth of chordal graphs" *Discrete Applied Mathematics,45 (3), 233-248, 1993.*

[24] F.Hohl. "Time limited blackbox security: Protecting mobile agents from malicious hosts." *Proc. of Conf on Mobile Agent Security, LNCS 1419, pages 92–113, 1998.*

[25] E.Korach,D.Rotem and N.Santoro,"Distributed algorithms for finding centers and medians in networks."*ACM Trans. on Programming Languages and Systems 6(3):380-401,1984.*

[26] E.Kranakis, D.Krizanc, S. Rajsbaum. "Mobile agent rendezvous."*13th Int. Coll. on Structural Information and Communication Complexity (SIROCCO), 1–9, 2006.*

[27] L.M.Kirousis and C.H.Papadimitriou. "Interval graphs and searing." *Discrete Mathematics 55, 181-184,1985.*

[28] L.M. Kirousis and C.H. Papadimitriou, "Searching and pebbling." *Theoretical Computer Science,47:205-218 1986.*

[29] R. Klasing, E. Markou, T. Radzik, F. Sarracco. "Approximation bounds for black hole search problems." *9th Int. Conf. on Principle of Distributed Systems (OPODIS), 2005.*

[30] R. Klasing, E. Markou, T. Radzik, F. Sarracco. "Hardness and approximation results for black hole search in arbitrary graphs." *12th Int. Coll. on Structural Information and Communication Complexity (SIROCCO), 200-215, 2005.*

[31] T. Kloks. Treewidth. PhD thesis, Utrecht University, Utrecht, The Netherlands,1993.

[32] A. Lapaugh, Recontamination does not help to search a graph, Journal of the ACM, 40,2, (1993), 224–245.

[33] F.L. Luccio, Intruder capture in Sierpiński graphs, Proc 4th International Conference on Fun with Algorithms (FUN), Lecture Notes in Computer Science, Vol. 4475, Springer, Castiglioncello, Italy, 2007, pp. 249–261.

[34] F. Luccio, L. Pagli, and N. Santoro, Network decontamination with local immunization, Proc 8th Workshop on Advances in Parallel and Distributed Computational Models, (APDCM), Rodi, Greece, 2006.

[35] N. Megiddo and S. Hakimi and M. Garey and D. Johnson and C. Papadimitriou, "The complexity of searching a graph." *Journal of the ACM, 35 (1), 18-44, 1988.*

[36] R.H. Moehring. "Graph problems related to gate matrix layout and PLA folding." *Computational Graph Theory, G. Tinnhofer et al. editors, Springer, Wien, 17-32,1990.*

[37] P. Panaite, A. Pelc. "Exploring unknown undirected graphs." *Journal of Algorithms, 33 281-295, 1999.*

[38] P. Panaite, A. Pelc. "Impact of topographic information on graph exploration efficiency." *Networks, 36, 96–103, 2000.*

[39] T. Parson, "Pursuit-evasion problem on a graph." *Theory and applications in graphs, Lecture Notes in Mathematics, Springer-Verlag 426-441, 1976.*

[40] T. Parson, "The search number of a connected graph." *Proc. of 9-nth Southeastern Conference on Combinatorics, Graph Theory and Computing, Utilitas Mathematica 549-554, 1978.*

[41] V. A. Pham, A. Karmouch, "Mobile Software Agents: An Overview." *IEEE Communications Magazine volume 36, issue 7, 26–37, July 1998.*

[42] D.K. Slonim. "The power of team exploration: two robots can learn unlabeled directed graphs." *35th Symp. on Foundations of Computer Science (FOCS), 75–85, 1994.*

[43] E. H. Spafford, D. Zamboni. "Intrusion detection using autonomous agents." *Computer Networks, 34(4):547–570, 2000.*

[44] K.Skodinis. "Computing optimal linear layouts of trees in linear time." *In 8th European Symp. on Algorithms (ESA'00), Spring, LNCS 1879,pages 403-414,2000.*

[45] P. Scheffler. "A linear algorithm for the pathwidth of trees." *Proc. of Topics in Combinatorics and Graph Theory, in R. Bodendiek and R. Henn, editors, Physica-Verlag, 613-620, Heidelberg, 1990.*

[46] T.Sander, C.F. Tschudin. "Protecting mobile agents against malicious hosts." *Proc. of Conf on Mobile Agent Security LNCS 1419, pages 44–60, 1998.*

[47] J. Vitek, G. Castagna. "Mobile computations and hostile hosts." *Mobile Objects, pages 241–261. University of Geneva, 1999.*