Université d'Ottawa · University of Ottawa

Université d'Ottawa • University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Miao Jun HUANG

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

Master of Computer Science

GRADE - DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Contiguous Search by Mobile Agents in Cube Networks and Chordal Rings

P. Flocchini

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

A. Nayak

E. Kranakis

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Contiguous Search by Mobile Agents in Cube Networks and Chordal Rings

by

Miao Jun Huang

Thesis

Submitted to Faculty of Graduate and Postdoctoral Studies

for the degree Master in Computer Science

Ottawa - Carleton Institute for Computer Science

University of Ottawa

Ottawa, Ontario, Canada K1N 6N5

Your file Votre référence
ISBN: 0-494-01496-2
Our file Notre référence
ISBN: 0-494-01496-2

Abstract

In this thesis we consider the problem of searching for an intruder in a network. There is a team of collaborative software agents that are deployed to capture a hostile intruder (e.g., a virus). These agents move along the network links and the intruder has the capability of escaping arbitrarily fast. We propose different strategies for the solution of the problem in some widely used topologies for the local area network and interconnection network: hypercube network, butterfly network and chordal ring network. In each topology, different models are studied depending on the capabilities of agents; i.e., agents' synchronicity, cloning power and visibility. For each model, we analyze the strategies in terms of number of agents employed, number of moves performed by the agents, and time.

Acknowledgments

My supervisor, Paola Flocchini, has placed her extensive knowledge of computer science at my disposal during our discussions and has given me invaluable feedback at every stage in the preparation of this thesis. It has been very pleasant to work with her and I am grateful for her guidance.

I am deeply thankful for my family's continuing support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Problem

We consider a networked environments where nodes are hosts and links represent connections between hosts. A possibly dangerous piece of software (e.g., a virus) is moving in the network from host to host; we will call such an element, the *intruder*. A team of collaborative software agents is deployed to protect the network with the goal of capturing the intruder, in other words, "touching" the intruder.

We make the following assumptions about the software agents: agents are initially located at the same node (the *homebase*), communicate through whiteboards located at the nodes; each agent is autonomous in the sense that it has some local memory ($O(\log n)$ bits suffice for all our algorithms), can perform local computations, can write to and read from a whiteboard located at the node it resides in, can move from node to neighbouring node. Furthermore, if we assume agents move asynchronously; i.e., it takes them an unpredictable (but finite) amount of time to move on a link.

The intruder is also a software agent. We assume that, although it could possibly harm the hosts, it cannot damage the other agents. Moreover, to concentrate on the worst case scenario, we assume that given any strategy, the intruder always behaves in the most powerful way. In other words, the intruder moves as if it can "see" the whereabouts of the team of agents, thus

avoiding them as much as possible. However, we assume that agents cannot "see" the intruder.

Our goal is to devise an efficient strategy for deploying the agents in the network so to capture the intruder. Efficiency will be measured in terms of number of agents to be involved, traffic (i.e., number of moves the agents have to perform), and ideal time.

1.2 Related Works

1.2.1 Graph Search

A variation of the intruder capturing problem has been extensively studied in the literature under the name of *graph search* (e.g., see [10, 18, 20, 21, 27]). The graph search problem consists of having a system of tunnels represented by edges of a graph. These tunnels are initially all *contaminated* by a gas, and have to be decontaminated (or cleaned) by a sequence of actions (moves) executed by a set of *searchers*. The problem was first introduced by Breish [7] and Parson [25, 26]. There are different versions of this problem that depend on the allowed moves.

In the *node search problem*, an action consists of one the following operations: 1) place a searcher on a vertex; 2) remove a searcher from a vertex. An edge (x, y) is decontaminated when two searchers are placed, one on x and the other on y.

In the *edge search problem*, a new operation is allowed: 3) move a searcher along an edge. In this problem a contaminated edge (x, y) can be decontaminated by : either i) placing two searchers on x and moving one of them to y along the edge (x, y) (i.e., traversing the edge); or ii) if all the edges incident to x, but (x, y), are decontaminated, and x contains a searcher, by moving the searcher along (x, y).

In both problems, a node is said to be *guarded* if it contains a searcher; a cleaned node x may be recontaminated unless x is guarded or all its incident links are clean; a clean edge can be recontaminated if there ever exists an unguarded path to a contaminated edge due to the removing of a searcher.

The aim of both problems is to find a strategy that permits to reach a state in which all edges are simultaneously decontaminated. The strategy is *optimal* if it globally requires a *minimum*

number of searches. This number is called the *node-search number* ($ns(G)$) or the *edge-search number* ($es(G)$) respectively in the node or edge search problem.

This graph search problem has been extensively studied. The goal of most investigation is to prove that finding the search number is NP-complete. For example, it has been shown that the node search number problem is NP-complete in bipartite graphs [16], cobipartite graphs [1], bipartite distance hereditary graphs [17], planar graph with maximum degree three [18], and starlike graphs [14]. There exist linear-time algorithms in trees [10, 22, 28], one in k-trees for fixed k [4], in cographs [6], and an $O(pm)$-time algorithm in permutation graphs (where p is the pathwidth) [5], and finally, for any fixed k, an $O(mn^k)$-time and $O(m)$ space algorithm on the k-starlike graph of n vertices and m edges [27]. Furthermore, the edge search number problem is NP-complete for general graphs [21] and for planar graphs with maximum degree three [24]. It can be solved in linear time in trees [21], and given n the number of vertices and m the number of edges, in $O(mn^2)$-time in split graphs, in $O(m+n)$-time in interval graphs, in $O(mn^k)$-time in k-starlike graphs, for any fixed $k \geq 2$ [27].

One of the reasons for the interest in the search number problem is the fact that it is closely related to classical measures of graph complexity such as cutwidth, vertex separator, pathwidth, treewidth, interval-width. For example, it is first observed by I. H. Sudborough that the edge search number of a graph cannot exceed its cutwidth. It is showed in [23] by Makedon and Sudborough that the edge search number of G is identical to the cutwidth of G for all graphs of maximum degree 3. Similarly, Kirousis and Papadimitriou showed in [18] that $|es(G) - ns(G)| \leq 1$ and the node search number $ns(G)$ is equal to vertex separator plus 1; they also showed in [19] that the interval-width of a graph is equal to its node search number. Seymour and Thomas [29] show that the t-search number of a graph G is equal to the tree width of G plus one. Takahashi etc. [32] proved that for any simple graph G,the mixed-search number is equal to the proper pathwidth of G. Instead of solving the graph search problem (both edge and node search) directly, by proving its equivalent to some other graph problems (such as cutwidth, pathwidth, vertex separation), some results already obtained for those problems can apply to the graph search. For other results on graph search, please refer to [8, 9, 12, 30, 31].

In spite of the interest for the computability of the search number, very little is known in

terms of search strategies and on lower bounds for the search number. In fact, only the tree has been deeply investigated in [21] where an optimal strategy that uses $log_3\ n$ is described.

The graph search problem and its variants have many applications, including pursuit-evasion problems in a labyrinth [7, 25, 26], decontamination problems in a system of tunnels contaminated by toxic gas and network security problem [3, 15]. Moreover, the graph search problem appears in the VLSI design because it is equivalent to the gate matrix layout problem [22, 11].

1.2.2 Contiguous Search

An interesting variation of the graph search problem is called the *contiguous search problem* [2, 3]. The main difference in this setting is that the mobile agents are used as searchers; they cannot be removed from the network, i.e., they can only move from a node to a *neighbouring node*. More precisely, in [3] the *contiguous search problem* has been defined as a variation of graph search as follows: (1) the removal of agents is not allowed. (2) at any time of the process, the clean nodes and their edges form a connected subnetwork. (3) a clean node (or edge) cannot be recontaminated.

It is easy to see that contiguous search and intruder capturing are equivalent problems. We distinguish two types of contiguous search. One is *contiguous edge search*, which is a contiguous search with the intruder "hiding" only on edges. The other is *contiguous node search*, which is also a contiguous search with the intruder "hiding" only on nodes. Since the intruder can "see" the movement of the agents, it will move so to avoid them as much as possible; however, since agents cannot see the intruder, the only solution for them is then to "clean" the whole network, thus capturing the intruder in the last cleaned node or edge. Since the contiguous search and intruder capturing are equivalent problem, in the thesis, we will mostly use the terminology of contiguous graph search in the intruder capturing problem.

In the contiguous edge search, all edges are initially *contaminated*, which means the intruder might hide in any of these edges. Edges can become *clean*. The decontamination (clean) of an edge is defined exactly as in the edge search. A clean edge can be *recontaminated* if, after the edge is cleaned, due to the move of an agent, there exists a path from a contaminated edge

leading to this clean edge without an agent on a node that guards the clean path portion. A node can also be *clean, guarded or contaminated.* A node is clean when all its incident edges are clean; is guarded when an agent is on it; otherwise is contaminated. In the contiguous node search, edges do not have states, but nodes can be *contaminated, guarded, or clean.* Initially, all nodes are *contaminated*, which means the intruder might hide in any of these nodes. We say a node guarded if an agent is currently on it. A contaminated node can become clean if an agent passed by it and all its neighbors are either clean or guarded. However, a clean node might be recontaminated if, after the node is cleaned, due to the move of a guarding agent, there exists a path from a contaminated node leading to this clean node but without an agent on the way to isolate the clean node(s).

In the contiguous node search problem, the minimal number of agents required to capture the intruder is called the *contiguous node search number* ($cns(G)$) and in the contiguous edge search the minimal number of agents is called the *contiguous edge search number* ($ces(G)$).

Notice that in all the graph search variations studied in the literature the results are heavily based on the assumption that a searcher can be initially placed on an arbitrary node and can be arbitrarily moved to any other node. The contiguous assumption considerably changes the nature of the problem and the classical results on node and edge search do not generally apply. In addition, the contiguous search problem increases in difficulty with respect to the non-contiguous one. As mentioned in [21] there exist trees for which a minimal search strategy without removal require at least $\Omega(n \log n)$ steps, whereas if the removal is allowed there exists a strategy in each graph that requires at most $O(n)$ steps [20].

The problem of finding the optimal number of agents in the contiguous search is still NP-complete for arbitrary graphs, and has been shown to be solvable with a linear time algorithm for computing the contiguous search number only in trees [3] where the corresponding minimal contiguous search strategy is given. It is also shown in [3] that the contiguous edge search number of any tree with n nodes is at most $\lfloor \log_2 n \rfloor$. In contrast, note that it is known that $\log_3 n$ searchers suffice for the classical edge search in every tree with n nodes. No other topologies have been studied in this variation of the graph search problem. An investigation on the relationship between the number of searchers of different variations of this model and the

classical one for graph search has been studied in [2]. It is shown in [2] that for any graph G the contiguous edge search requires at least $es(G)$ agents. In other words, $es(G) \leq ces(G)$, the contiguous edge search number is no smaller than the classical edge search number.

1.3 Our Model – Contiguous Search

In this thesis we will be considering *contiguous node search* and *contiguous edge search.* For simplicity, from now on, when we say *node search*, we actually means contiguous node search instead of the node search studied in the literature and *edge search* actually means contiguous edge search. Mobile agents have to capture the intruder in the networks. To do so, following a common strategy, they have to "surround" the intruder so that it does not have an edge to escape through. A possible strategy is to let the agents move in the network starting from the common homebase so to visit nodes, checking the presence of the intruder, in such a way that no "corridor" (i.e., no way out) is created. In this way the intruder would not be able to escape in an already visited part of the network and thus it will be eventually discovered. The node search problem is solved when agents "surround" the last contaminated node, where the intruder must hide if it has not been captured during the strategy, anyone of these agents by moving to the last node will capture the intruder. We say agents surround a node if this node's neighbors are guarded by agents. The edge search problem is solved when the agents search all edges as in the classical edge search problem, i.e., by traversing and then protecting them.

Notice that according to the definition of contiguous search, the setting does not allow agents to search in the network starting from distinct and separate entrances (i.e., the entrance of the agents is unique); the setting also implies that a contiguous search cannot be applied to non-connected graphs, therefore from now on we will only consider connected graphs. Finally the fact that a clean node (or edge) cannot be recontaminated implies that our strategy is monotone; i.e., agents do not revisit part of the graphs.

In the following we define our model and our assumptions. As mentioned before, since the contiguous search and the intruder capturing are equivalent problems, in the thesis we will mostly use the terminology of contiguous graph search.

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.

Each node has a distinct id and a bounded amount of local storage, called *whiteboard* ($O(log\ n)$ bits suffice for all our algorithms). The state of the node is written on the node's whiteboard. In addition, in the contiguous edge search, the states of the edges incident on the node are also written on the whiteboard. Also written on the whiteboard of a node are: its Id (binary string), and the labels of the incident ports. Under some circumstance, the whiteboard can record some useful information for the agents; i.e., the number of agents presently on the node etc.

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 bounded computational storage, obey the same set of behavioral rules (the *protocol*). Initially, all agents are 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) and know the topology of the network.

Agents communicate with each other through the whiteboards, in fact they can read from and write on the whiteboards. Access to a whiteboard is gained fairly in mutual exclusion. In addition, we may assume that the agents are asynchronous or synchronous, have visibility and/or cloning capability.

- **Timing Assumption** :

  Local computation is considered instantaneous. In the thesis we consider both syn-

chronous and asynchronous models. In the asynchronous model, every movement of an agent takes a finite but otherwise unpredictable amount of time. In the synchronous model, we assume there is a global clock. When an agent leaves a node and moves along one of the edges, it will arrive at the other endpoint of the edge in one time unit.

- **Visibility** :

  When we consider the "visibility model" we assume that an agent located at a node can "see" the whiteboards of its neighboring nodes. Specifically, an agent want to "see" whether its neighboring nodes are clean, guarded or contaminated. 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.

- **Cloning** :

  The cloning capability allows agents to clone other identical agents if needed. With the cloning power, an agent during the cleaning process can have two operations: create and terminate. If agents do not have cloning power, then all agents are created at the beginning of the cleaning strategy and terminate at the end of the cleaning strategy.

The Intruder:

The intruder is also a software agent. In the node search, it can "hide" only on nodes. However in the edge search problem, it "hides" only on edges (e.g, like a toxic gas). The intruder always behaves in the most powerful way. In other words, the intruder moves as if it can "see" the whereabouts of the team of agents, thus avoiding them as much as possible. The intruder cannot move through a guarded node. In the node search problem, the intruder is captured when an agent arrives on the same node and there is no way for the intruder to escape; i.e., the neighbors are guarded by agents. In the edge search problem, the intruder is captured only when all the edges of the network are cleaned and no recontamination can occur.

Complexity Measures: Efficiency will be measured in terms of number of agents to be deployed, traffic (i.e., number of moves performed by the agents overall), time and storage. More precisely:

- **Number of agents:**

  In general, to compute the minimal number of agents required is a NP-complete problem. The number of agents given for our cleaning strategies is a constructive upper bound.

- **Moves**

  One agent moving along an edge is counted as one move performed by this agent.

- **Time**

  In case of asynchronous agents there is no notion of time since the time taken by the agents' movement is unpredictable. In other words, we cannot measure time. However, we can measure time assuming particular conditions. The measure usually employed is the *ideal time complexity*. That is, (in absence of failure), it takes one unit of time for an agent to traverse a link. Such assumption will be made *only* for the purpose of measuring time.

- **Memory**

  There are two storage areas: the whiteboard on each node and the local memory of the agents. For both, $O(log\ n)$ bits suffice for all our strategies.

1.4 Terminology for Our Strategy

In general, for all the strategies described in the thesis, we assume when an agent arrives at a node, it reads the state of the node from the whiteboard. If the state is *contaminated*, it changes the state to be *guarded* by writing to the whiteboard. Otherwise, it does nothing. Right before a group of agents leaves the node, one of them will change the state of the node to be *clean* by writing on the whiteboard. All our strategies are monotone, it means that once a node (edge) becomes clean, it will stay clean until the end of the algorithm. We also assume there are no failure in the system; i.e., an agent leaves a node and will arrive at its neighboring node eventually.

Group Movement:

Under some circumstances, our strategies will require a group of agents to move from a node to a neighboring node. Since agents are autonomous, in an asynchronous model, a group of

agents leave a node at the same time and head for the same destination but might arrive at different times. However, some strategies require all the expected agents be on the node in order to execute the next step. This can be easily accomplished. A counter indicating the number of agents presently on the node can be written on the whiteboard as well as the number of agents expected to be on the node (this depends on the strategy). Initially, the counter is 0. When an agent arrives at the node, it will increase the counter by writing on the whiteboard. So, when the last agent arrives, it will know no more agents will arrive.

In the thesis, when we talk about group movements, we will assume the agents move as described above.

Local Coordinator:

Under certain circumstances in our strategies, it will be required to elect a local leader among a group of agents present at a node. The leader might be selected by the local agents on a node simply by accessing the whiteboard; the first that gains access will become the local leader. The leader also might be the last agent which is expected to arrive at this node (if the total number of expected agents is known). This local leader will be responsible for all the processing on the node; i.e., create new agents, terminate agents, send out agents, change the state of a node to be clean before leaving; access the whiteboard for some needed information.

Global Coordinator (Synchronizer):

In some cases, we will need a global leader (synchronizer). The synchronizer acts as a coordinator for the entire cleaning process. It can be locally selected by the agents placed on the homebase by accessing the whiteboard; the first that gains access will become the synchronizer.

1.5 Our Results

In this thesis, we study the contiguous and monotone search problem in three common topologies for interconnection network: hypercube, butterfly and chordal ring. For each topology we are concerned with two variations of the contiguous and monotone search problem: node search and edge search.

The main objective of this thesis is to compare different variations of the model for node

search and edge search. More precisely, we consider asynchronous and synchronous strategies, we study agents with and without cloning capabilities and we consider agents with and without visibility power. We concentrate mostly on the hypercube, where we study all the above mentioned variations. We then study only some of the models for the butterfly and the chordal ring.

In Chapter 2 we study the search problem in the hypercube network. In the node search, five models are discussed depending on the agents' synchronicity, visibility and cloning power. **1** (AS) We first describe a cleaning strategy for the asynchronous node search model that requires $O(\frac{n}{\log n})$ agents, $O(n \log n)$ time steps and $O(n \log n)$ moves. In this strategy one agent acts as a coordinator and leads the other agents in a precise order and on a special spanning tree of the hypercube, in such a way that the whole network is cleaned and no recontamination occurs. **2** (ASCLO) In order to avoid moving the needed agents from the homebase, we add the cloning power to agents. The agents can create new agents, when needed. The cleaning strategy, that employs a synchronizer and agents with cloning power for the asynchronous node search, requires $\frac{n}{2}$ agents, $O(n \log n)$ time steps and $O(n \log n)$ moves. Although cloning power avoids moving the extra agents from the homebase to the needed nodes and reduces the number of moves by the agents, the number of moves by the synchronizer dominates. **3** (ASVI) To avoid using a synchronizer, we add the visibility capability to the agents (but no cloning). The agents are allowed to "see" the state of their neighbors. The cleaning strategy, that employs agents with visibility for the asynchronous node search, requires $\frac{n}{2}$ agents, $O(\log n)$ time steps and $O(n \log n)$ moves. The agents' visibility greatly speeds up the cleaning process to $O(\log n)$. This strategy has the great advantage of being totally *local*, meaning that the agents can make their decisions on the actions to take without the need of a coordinator. **4** (ASCLOVI) The cleaning strategy, that employs agents with cloning power and visibility for the asynchronous node search, requires $\frac{n}{2}$ agents, $O(\log n)$ time steps and $O(\log n)$ moves. The agents' visibility and cloning power combine to improve not only the process time to $O(\log n)$ but also to reduce the number of moves to $n - 1$, which is optimal for the node search. **5** (SYCLO) The cleaning strategy that employs agents with cloning power for the synchronous node search, requires $\frac{n}{2}$ agents, $O(\log n)$ time steps and $O(\log n)$ moves.

From the Strategy (AsCloVi and SyClo), we know that the agents' synchronous movement can replace agents' visibility to achieve the same complexity or vice versa. Compared to the other strategies, Strategy (AsCloVi and SyClo) have good performance in terms of time and number of moves, but require more agents. Strategy As uses less agents but more moves and times. Strategy AsVi, AsCloVi, SyClo have the advantage of being totally local, while strategy As and AsClo require a global coordinator.

In the edge search, three models are discussed. The cleaning strategy, that employs a synchronizer for the asynchronous edge search, requires $O(\frac{n}{\log n})$ agents, $O(n \log n)$ time steps and $O(n \log n)$ moves. This strategy has the same complexity as strategy As. The strategy developed for the asynchronous edge search model and agents with cloning power requires $O(n)$ agents, $\log n$ time steps and $\frac{n}{2} \log n$ moves. The last strategy developed for the synchronous edge search and agents with cloning power also requires $O(n)$ agents, $\log n$ time steps and $\frac{n}{2} \log n$ moves. Comparing the last two strategies, we see that agents' synchronicity does not help in terms of complexity in the last strategy.

In Chapter 3 we study the search problem in the d-dimensional butterfly network with n nodes and $d2^{d+1}$ edges. Since we employ the same strategies for both node search and edge search, we focus on developing the cleaning strategies for the edge search. The synchronous and asynchronous models are discussed in the edge search. All the cleaning strategies developed for the edge search or node search, have the same complexity and require $O(\frac{n}{d+1})$ agents, $3d-1$ time steps and $O(n \log n)$ moves. We observe that agents' cloning power can reduce the number of moves. We also observe that the agents' visibility is not helpful in the strategies we designed because they are devised in a way that agents on a node can implicitly know the state of this node's neighbors. So extra visibility power added to agents does not help in our strategies.

In Chapter 4 we study the search problem in the chordal ring network with n nodes and link structure $\langle 1, d_2, ..., d_k \rangle$. The asynchronous and synchronous models are discussed in the edge search. In the asynchronous edge search, the strategy requires $2d_k+1$ agents, $2kn+2d_k^2-2d_k+2$ moves and $2kn - d_k + 2$ time steps. The cleaning is inherent sequential, only one agent is cleaning all the links of the chordal ring while all the other agents are there to guard the nodes from recontamination. In the synchronous edge search, we study a special case of chordal ring,

the chorded ring $C\langle p, k\rangle$. To speed up the cleaning process a different strategy is devised. This strategy requires $2k + \frac{p^2+p}{2}$ agents, $(p+1)n + k^2 + \frac{p^3-p}{6} - 1$ moves and n time steps. The strategy is inherent concurrent, at one time, all the clockwise links of a node are clean. However, it employs many more agents. The asynchronous and synchronous models are also studied in the node search. Here the strategies are similar to the ones for the asynchronous edge search but without cleaning every link. The strategy for the asynchronous node search requires $2d_k + 1$ agents, $4n + 2d_k^2 - 5d_k$ moves and $4n - 4d_k$ time steps; the strategy for the synchronous node search requires $2d_k + 1$ agents, $2n + 2d_k^2 - 2d_k$ moves and $n + d_k - 1$ time steps. We observe that the number of moves can be reduced if the agents have cloning power.

Finally in Chapter 5 we conclude with some open problems.

Chapter 2

Hypercube

The hypercube is one of the most common topology for interconnection networks and the architecture of most parallel computers.

In this chapter, we are going to study the cleaning problem by mobile agents in the hypercube. Two variations of this problem (edge search and node search) are considered. In each variation different models are studied. For each model a strategy is devised and the complexity is analyzed. For the node search problem, five models are studied: Model AS, ASCLO, ASVI, ASVICLO, SYCLO. For the edge search problem, three models are discussed: EDAS, EDASCLO, EDSYCLO. Finally, we give some observations and discuss some open problems.

2.1 Definitions and Terminology

The d-dimensional hypercube H_d has $N = 2^d$ nodes and $d2^{d-1} = \frac{n}{2}log\ n$ edges. Each node corresponds to a d-bit binary string, and two nodes are linked with an edge if and only if their binary strings differ in precisely one bit. As a consequence, each node is incident to $d = logn$ other nodes, one for each bit position. For example, a 3-dimensional hypercube has $2^3 = 8$ nodes, the binary representations of nodes are 000, 001, 010, 110, 100, 101, 011, 111. (Please see Figure 2.1 (a))

Given the binary string representation x of a node v, we define an alternative name $\widehat{x}$ for node v as follows:

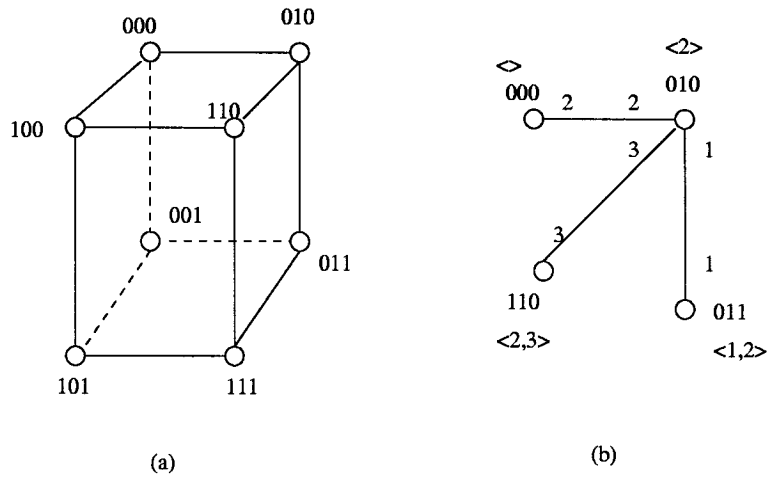


Figure 2.1: The 3-dimensional hypercube.

Definition 1. *Node v can be labeled alternatively by $\widehat{x} =< i_1, i_2, ..., i_l >$ where i_k are the positions of the 1 bits in x for $1 \leq k \leq l$ and $i_k > i_{k-1}$.*

Notice that the source (00...00) would be labeled as $< 0 >$. For example, node $x =$ (001110) would be alternatively labeled as $\widehat{x} =< 2, 3, 4 >$, while node $y =$ (001001) would be $\widehat{y} =< 1, 4 >$. We say lexicographically $\widehat{x} < \widehat{y}$. Since the binary representation x of a node is unique, by the definition, the $\widehat{x}$ is unique too. As a consequence, we can lexicographically order all nodes.

We define the rightmost bit of x is the first bit and in position 1. Let $m(x)$ denote the position of most significant bit of x. If $\widehat{x} =< i_1, i_2, ..., i_l >$, then $m(x) = i_l$. For example, for node (001110), $\widehat{x} =< 2, 3, 4 >$, and $m(x) = 4$.

The edges of the hypercube can be labeled according to their dimensions. In other words, the edges of a node can be labeled with the bit position in which their binary representation differ. For example, $\lambda_{000011}(000011, 000010) = 1$ because node (000011) and node (000010) differ in dimension 1; $\lambda_{000011}(000011, 001011) = 4$, because node (000011) and node (001011) differ in dimension 4.

Please see Figure 2.1 (b) for an example of the edge and node labels.

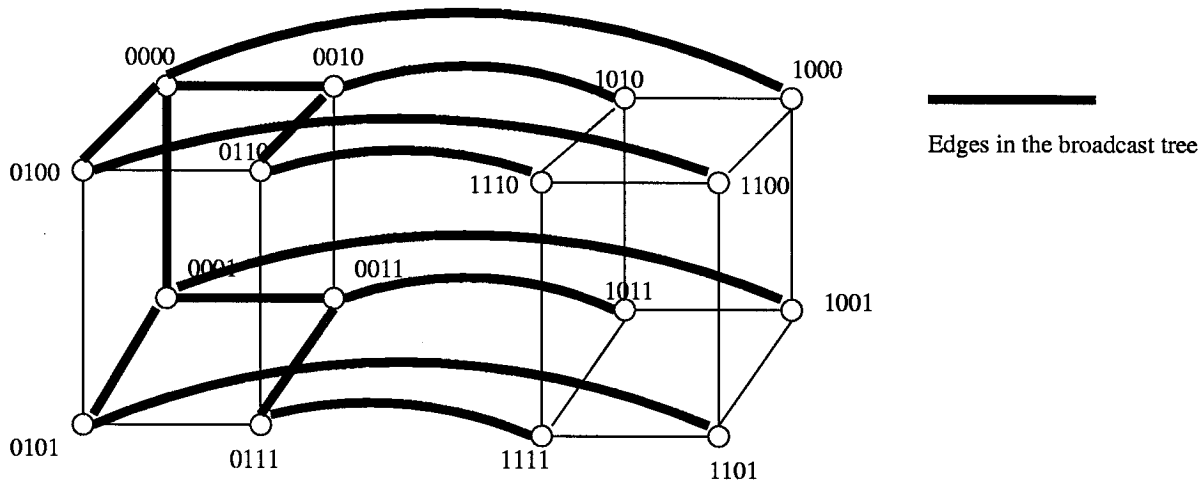


Figure 2.2: The 4-dimensional hypercube and its broadcast tree.

2.1.1 Broadcast Tree

We consider a special spanning tree of the hypercube rooted at the source (node (00..00)). The spanning tree is called *broadcast spanning tree* because it is commonly used to do optimal broadcast in the hypercube: a node x receiving a message from dimension i will forward it to all nodes connected by dimension $j > i$. Let h be the position of the highest 1 bit in the binary representation of x (ex: if $x = (0010)$, $h = m(x) = 2$ counting from the less significant bits) then x is connected to all the nodes in the next level whose bit in which they differ is in a position higher than h (in the above example (0110) and (1010)). Please see Figure 2.2.

Let us organize the hypercube H_d in $d+1$ levels and let level i consist of all the nodes whose binary representation contains i ones. Clearly, all the nodes at level i ($0 < i < d$)are connected only to nodes of level $i - 1$ and to those of level $i + 1$; the nodes at level 0 are connected only to nodes of level 1; the nodes at level d are connected only to nodes of level $d - 1$. For the following discussion, let us assume that the degree of the hypercube d is even. (see Figure 2.3). This assumption is made for ease of discussion; very minor modification would be required for our results to hold when d is odd.

The resulting spanning tree is also known as heap queue:

Definition 2. *A heap queue $T(d)$ is a rooted tree recursively defined as follows:*

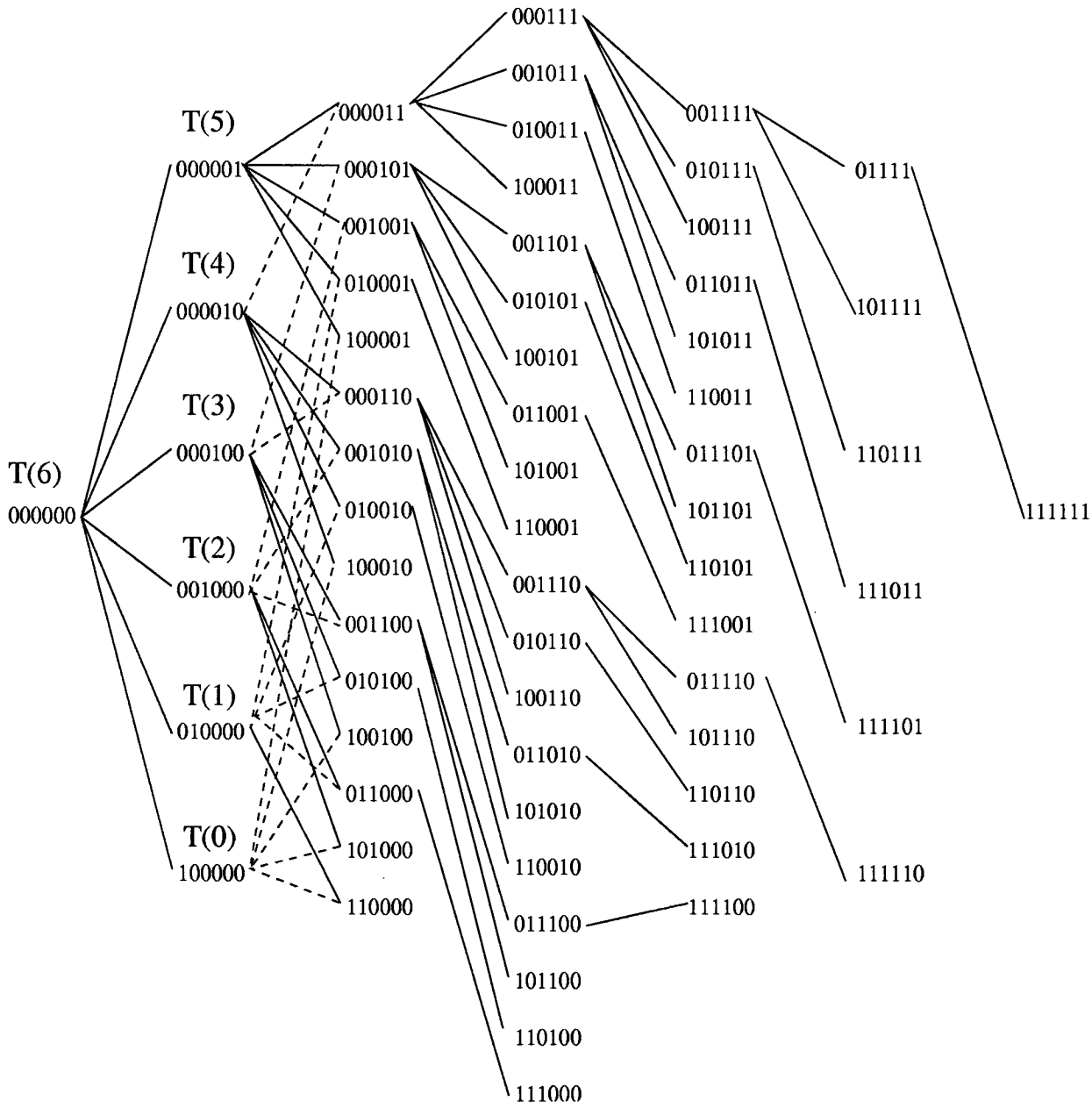


Figure 2.3: The broadcast tree $T(6)$ of the hypercube H_6. Normal lines represent edges in $T(6)$, dotted lines (only partially shown) the remaining edges of H_6.

$T(0)$ *is a leaf*

$T(1)$ *is a node with one child*

$T(k)$ *is a node with* k *children of type* $T(0), \ldots T(k-1)$.

In fact we have that:

Lemma 1. *A broadcast spanning tree of a hypercube of size* n *is a heap-queue* $T(log\ n)$.

Level 0 contains the source $(0, 0, \ldots, 0)$ of the tree (which is of type $T(d)$ where $d = log\ n$ is the degree of the hypercube), level l contains the nodes whose binary representation has l 1s.

Here are some useful properties of the constructed broadcast tree:

Lemma 2. *At level 0 there is one node of type* $T(d)$. *At level* $l > 0$ *there are* $\binom{d-k-1}{l-1}$ *nodes of type* $T(k)$ *with* $0 \leq k \leq d - l$.

PROOF Nodes of type $T(k)$ start with k 0s followed by a one. Since they are at level l they must contain $l - 1$ 1s in the last $d - k - 1$ positions. ∎

Lemma 3. *Let* $T(d)$ *be a broadcast tree for the hypercube. Level 0 has no leaves. The number of leaves at level* $l > 0$ *is* $\binom{d-1}{l-1}$.

PROOF By definition of broadcast tree, the leaves are those nodes whose d-bit representation start with 1 (i.e., the nodes that do not have bigger neighbour at the next level). Thus, they contain $l - 1$ 1s in the remaining $d - 1$ bits. ∎

Lemma 4. *Let* $T(d)$ *be a broadcast tree for the hypercube. The total number of leaves is* 2^{d-1}.

PROOF By definition of broadcast tree, the leaves are those nodes whose d-bit representation start with 1. Thus, they are half of the total number of nodes. ∎

Lemma 5. *Let* $T(d)$ *be a broadcast tree for the hypercube. The number of nodes at level* l *is* $\binom{d}{l}$.

PROOF By construction the nodes at level l are those containing l ones in their binary representation. Since the representation contains d bits, there are exactly $\binom{d}{l}$ such nodes. ∎

All nodes of the hypercube can be ordered lexicographically. Let $\widehat{x_1^l}, \ldots, \widehat{x_m^l}$ ($m = \binom{d}{l}$) be the nodes of level l lexicographically ordered. Notice that this order corresponds to the order in which the nodes are drawn in Figure 2.3. Given any node $\widehat{x}_j$ of the hypercube H_d, it is easy to compute the next node $\widehat{x}_{j+1}$.

Lemma 6. *Let* $\widehat{x}_j =< i_1, i_2, ..., i_l >$ *be a node at level* l *of the hypercube* H_d,
$\widehat{x}_{j+1} = < i_1, i_2, ..., i_l + 1 >$, *if* $i_l < d$;
$\widehat{x}_{j+1} = i_1 + 1, i_1 + 2, ..., i_1 + l$ *if* $i_l = d$ *and* $i_1 < d + 1 - l$;
$\widehat{x}_{j+1} = < 1, 2, ..., l + 1 >$ *at level* $l + 1$ *if* $i_l = d$ *and* $i_1 = d + 1 - l$;

For example, node $\widehat{x} =< 2, 5 >$, the next node in the lexicographical order is $< 2, 6 >$, while the next node of $< 2, 6 >$ is $< 3, 4 >$ and the next node of $< 5, 6 >$ is $< 1, 2, 3 >$.

2.1.2 Different Visualization of the Broadcast Tree

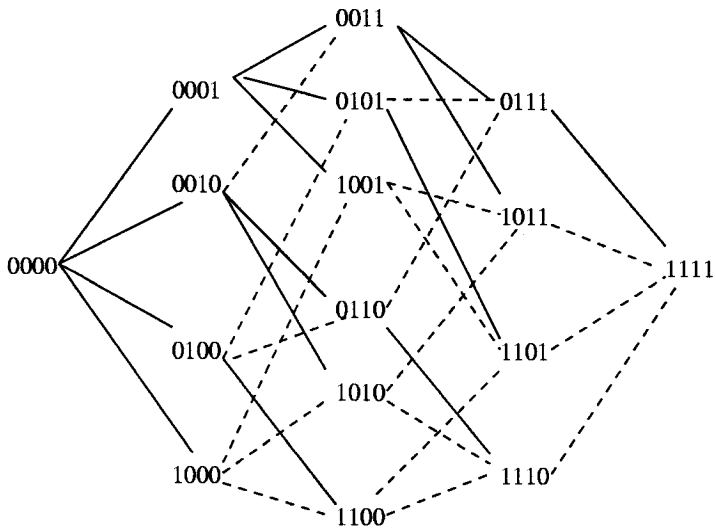
We now group nodes based on the position of the most significant bit of their labels, i.e., differently from the groups generated by the levels of the broadcast tree. Let C_i be the set of nodes whose most significant bit is in the i-th position. $C_0 = \{00...00\}$; $C_1 = \{00...001\}$; $C_2 = \{00...010, 00...011\}$ etc.

Let us redraw the broadcast tree in such a way that a node with the most significant bit in the i-th position will be shown to belong to group C_i. Please refer to Figure 2.4, in the top we have the broadcast tree of 4-dimensional hypercube ordered by levels; at the bottom each column corresponds to some C_i.

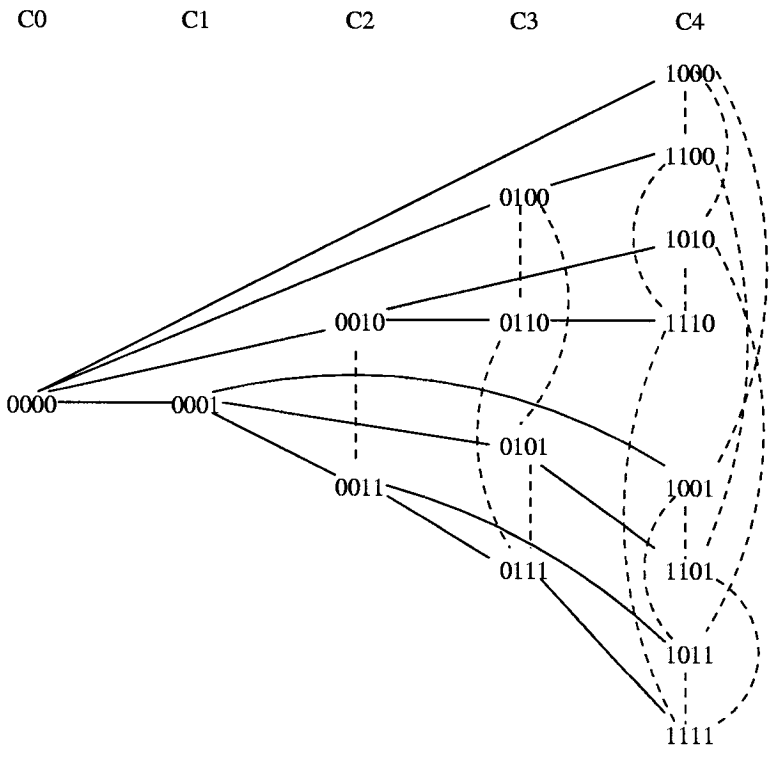
To easily refer to the neighborhood of a node, the neighbors are grouped into either smaller neighbors or bigger neighbors as defined below. Let x be a node and y be its neighbor.

Definition 3. *Node* y *is called a smaller neighbor of node* x *if* $\lambda_x(x, y) \leq m(x)$; y *is called a bigger neighbor of* x *if* $\lambda_x(x, y) > m(x)$.

By definition, the bigger neighbors of x are just the children of x in the broadcast tree. In other words, all the neighbors of node x whose most significant bit is smaller than or equal to that of x are *smaller neighbors* of x; all the neighbors of node x whose most significant bit is



(a)



(b)

Figure 2.4: (a)The broadcast tree of 4-dimensional hypercube; (b)the redrawing

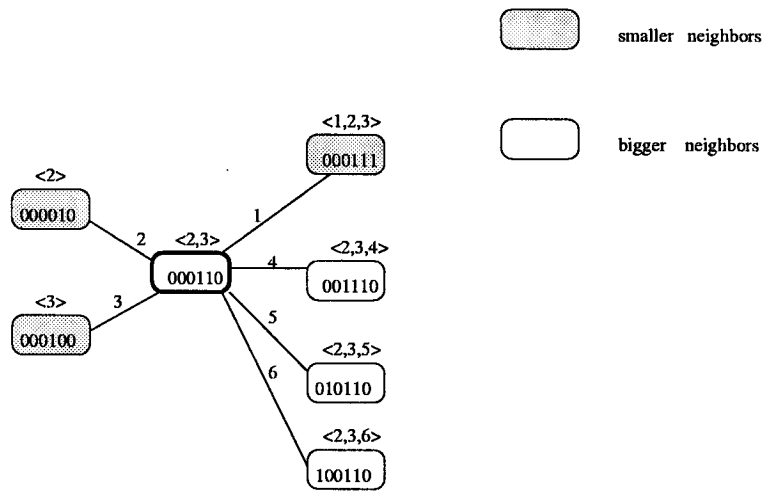


Figure 2.5: The smaller and bigger neighbors of node (000110).

higher than that of x are called the *bigger neighbors* of x. For example, the smaller neighbors of node (000110) are node (000111), (000010) and (000100); while node (001110), (010110), (100110) are its bigger neighbors as well as children in the broadcast tree of node (000110). (See Figure 2.5) Notice that node (00...00) does not have smaller neighbors; and nodes whose most significant bit is in the d-th position, which are leaves in the broadcast tree, do not have bigger neighbors.

Here are some useful properties of the new visualization of the hypercube:

Lemma 7. *Exactly one node is in C_0; 2^{i-1} nodes are in C_i for $0 < i \leq d$.*

PROOF By definition, only node (00..00) is in C_0. The nodes in C_i have the most significant bit in the i-th position. There are 2^{i-1} nodes with most significant bit in the i-th position. Hence C_i has 2^{i-1} nodes. ∎

It directly follows from the definition of the broadcast tree that

Lemma 8. *All the leaves of the broadcast tree are in the C_d.*

Let node x be any node in C_i. Since C_0 has only one node (00..00) which has no smaller neighbors, we only consider x in C_i where $i > 0$ for the following lemma.

Lemma 9. *Exactly one smaller neighbor of node x is in C_j (where $j < i$), the other smaller neighbors, if any, are in C_i and all its bigger neighbors, if any, are in C_k (where $k > i$).*

PROOF By definition, the most significant bit of node x is in the i-th position. Its neighbors are only one bit different from it. If a neighbor y is different from x in the l-th position for $l < i$, then y is a smaller neighbor of x and has a 1 bit in the i-th position. By definition, y is in C_i too. If the neighbor y is different from x in the i-th position, then the most significant bit of y must be in position j, $j < i$. So y is in C_j. By definition, y is a smaller neighbor of x too. Since there is only one neighbor different from x in the i-th position, we have one smaller neighbors of node x in C_j, $0 \leq j < i$ and the other smaller neighbors in C_i. If the neighbor y is different from x in the k-th position where $k > i$, then the k-th position of y must be 1 because the most significant bit of x is in the i-th position. By definition, y is the bigger neighbor of x and it is in C_k where $k > i$. ∎

Let node x be any node in C_i. Since node (00..00) in C_0 has no smaller neighbors and node (00..001) in C_1 has one smaller neighbor in C_0, but no smaller neighbor in C_1, we only consider node x in C_i where $i > 1$ for the following lemma.

Lemma 10. *There must exist at least one smaller neighbor y of node x such that y is in C_i and a smaller neighbor z of y is in C_{i-1}.*

PROOF By lemma 9, all smaller neighbors (except one) of node x are in C_i. Each of these neighbors is one bit different from x in the j-th position, where $j < i$. Depending on the $(i-1)$-th position of x, there are two cases.

Case 1: If the $(i-1)$-th position of x is a 0 bit, let y be the smaller neighbor which is different from x in the $(i-1)$-th position. Node y is in C_i and has a 1 bit in the $(i-1)$-th position. Node y has a smaller neighbor z such that z is different from y in the i-th position. Node z has a 0 bit in the i-th position and a 1 bit in the $(i-1)$-th position, which is the most significant bit of z. So node z is in C_{i-1}.

Case 2: If the $(i-1)$-th position of x is a 1 bit, let y be any smaller neighbor which is different from x in the j-th position, $j < i-1$. Node y is in C_i and has a 1 bit in the i-th and $(i-1)$-th position. Let z be a smaller neighbor of y which is different from y in the i-th position. z has a 0 bit in the i-th position and a 1 bit in the $(i-1)$-th position, which is the most significant bit of z. So z is in C_{i-1}.

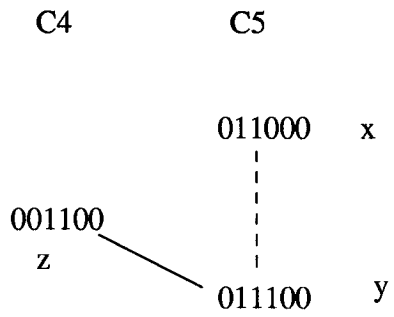


Figure 2.6: Node z is a smaller neighbor of y.

Hence, in any case there exists a smaller neighbor z of y such that $z \in C_{i-1}$. ∎

For an example of lemma 10, please refer to Figure 2.6.

2.2 Node Search

For the node search problem, we will consider both synchronous and asynchronous models. For the asynchronous model, we will study four variants, which are the model with a synchronizer, the model with a synchronizer and agents with cloning power, the model without synchronizer but agents with visibility, and the last model that agents have visibility and cloning power. We will also study the synchronous model where the agents have cloning power.

2.2.1 Asynchronous Model – AS

The agents are all identical; one of them, however, will act as coordinator for the entire cleaning process, we will call this agent *the synchronizer*. The synchronizer can be locally selected by the agents by accessing the whiteboard; the first that gains access will become the synchronizer. The synchronizer acts as a coordinator and follows an algorithm different from the other agents. Its task is to guide the movement of the agents on the broadcast tree so that they can correctly clean the hypercube.

Cleaning strategy

The main idea is to place enough agents in node $(00\ldots00)$ (the source) and then have them move level by level according to the strategy described below. The cleaning strategy is carried out on the broadcast tree. The agents will move on the broadcast tree and leaded by the synchronizer in such a way that during their moves, the intruder can not enter the nodes already clean. The intruder is allowed to move along any edge of the hypercube while the synchronizer can choose edges that are not of the broadcast tree for navigation. By lemma 6, the synchronizer knows which is the next node to go to clean.

At the beginning, all the agents are available at the root. We call the agents available at the root, *set of available agents*. The synchronizer is placed at the root too.

1. From the root to level 1

1.1 - One agent is sent to each of the root's children $T(d-1)\ldots T(0)$ guided by the *synchronizer*. The synchronizer makes sure the one agent arrives at its destination and then goes back to the root to guide another agent.

2. From level l to level $l+1$. [level l has one agent per node]

2.1 - Before starting to clean nodes in level $l+1$, the synchronizer goes to the root to collect the agents needed for completing the cleaning of level $l+1$ (i.e., $k-1$ agents per node of type $T(k)$ except for the nodes of type $T(1)$ and $T(0)$ which do not require any extra agent). So the root sends $k-1$ additional agents, in no specific order, to each node of type $T(k)$ at level l (none are sent to $T(1)s$ and $T(0)s$).

2.2 - When k agents are on a node of type $T(k)$ at level l, they are sent down on the broadcast tree to its children in level $l+1$ helped by the synchronizer. Let $\widehat{n_1^l},\ldots,\widehat{n_m^l}$ ($m=\binom{d}{l}$) be the nodes of level l lexicographically ordered. The synchronizer sequentially sends down the agents from level l to the level $l+1$, one per edge of the broadcast tree, following the lexicographical order of the $\widehat{n_i}$. It will send down all the agents on $\widehat{n_i}$, and will make sure that they arrive at their destination, before proceeding with node $\widehat{n_{i+1}}$.

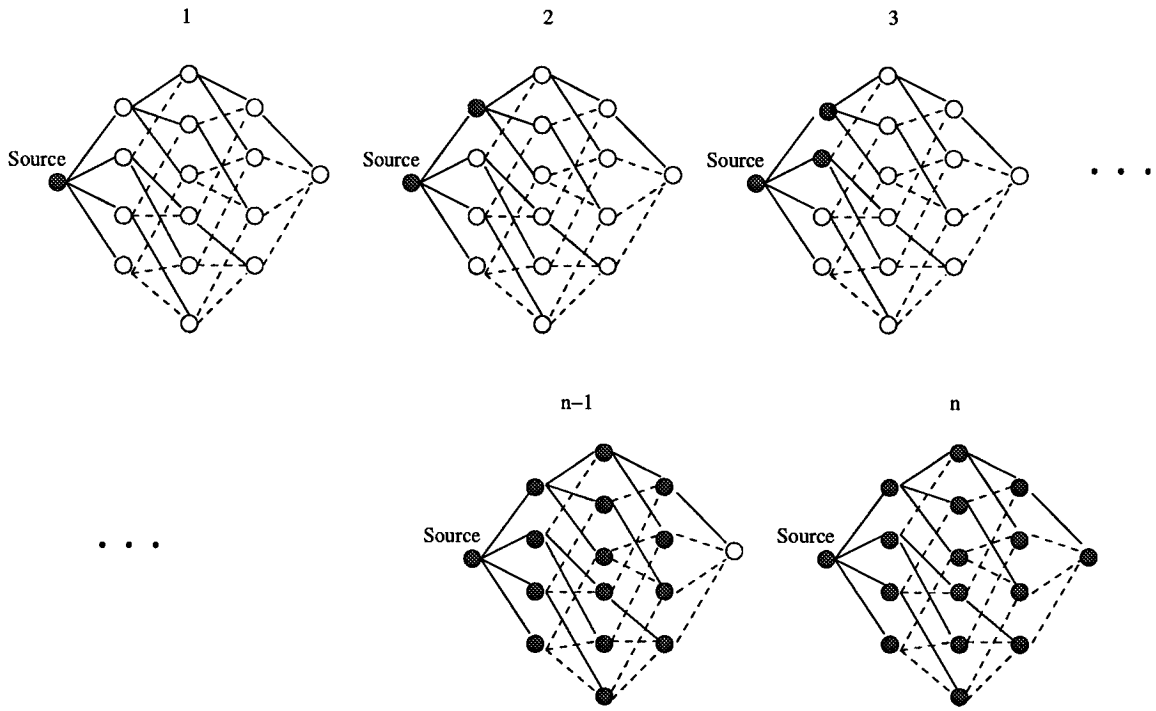


Figure 2.7: The order in which nodes are cleaned.

2.3 - When a leaf of level l is reached by the synchronizer, the agent on it becomes available and goes back to the root. Notice that when the synchronizer reaches the last node of level l, the only active agents are the ones covering level $l+1$.

Recall that the whiteboard is used for any communication between the synchronizer and the agents. In particular, during the strategy the whiteboard of a node stores the labels of the incident ports and the current number of agents present at the node; it is accessed by the synchronizer to decide when and where to send the agents to the next level. No more that $O(\log n)$ bits are required for this local storage.

Please see Figure 2.7 for the order in which nodes are cleaned by the strategy on the 4-dimensional hypercube.

Correctness and Complexity

Correctness. We now prove that our cleaning strategy is correct; i.e., that once a node has been cleaned, it will never be recontaminated and that all nodes will be cleaned.

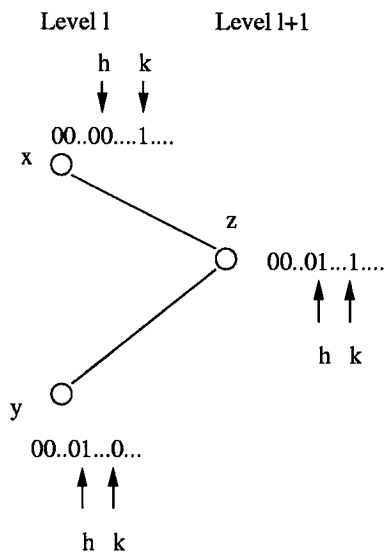


Figure 2.8: The relationship between x, y and z.

Let x be a node of level l, $N(x)$ denote the neighbors of x at level $l+1$ and $NT(x)$ denote the children of x at level $l+1$ in the broadcast tree. Notice that $N(x)$ includes $NT(x)$ and possible some other neighbours at level $l+1$.

Lemma 11. *If $z \in N(y) - NT(y)$ then $z \in NT(x)$ for some x such that $\widehat{x} < \widehat{y}$.*

PROOF Let h be the position of the most significant bit of y (highest 1 bit). Since $z \notin NT(y)$ we know that it does not contain a 1 bit in position higher than h. (see Figure 2.8.) However, z must have a 1 bit in position h; in fact if the bit in position h was 0, z would contain $l-1$ 1 bits and would belong to level $l-1$. So the new bit must be in a lower position. Let us assume, w.l.g, that $k < h$ is such a position (so the the k-th bit of y is 0 while the k-th bit of z is 1). Consider now a node x whose binary representation is the same as the one of y except for the h-th bit, which is 0 and the k-th bit, which is instead 1. Node x has the same number of 1s as node y. So node x and y are in the same level. Moreover, since the k-th bit in x is 1 but the k-th bit in y is 0, by the definition of lexicographical order, we have $\widehat{x} < \widehat{y}$. We have that $z \in NT(x)$ because they differ in just one bit (the h-th) and the most significant bit of z (the h-th) is higher than the most significant bit of x. ∎

We now prove the correctness of the strategy.

Lemma 12. *In our strategy, when agents leave a node in level i (leaving it unguarded) all its neighbours are either clean or guarded.*

PROOF This is clearly true for the one node of level 0. Assume it is true for all nodes of level $0 \leq j < i$ and consider level i.

Nodes at level i are only connected to nodes at level $i-1$ and $i+1$. When nodes at level i are sending agents to level $i+1$, by induction hypothesis, and the cleaning strategy, all nodes at level $i-1$ are clean. Let us assume that cleaning occurs at node y of level i now. For any $z \in N(y) - NT(y)$, by lemma 11, $\exists x$ such that $z \in NT(x)$ and $\widehat{x} < \widehat{y}$. According to the cleaning strategy, the synchronizer approaches the nodes in lexicographical order at each level. So agents on node x are already sent to level $i+1$ before the synchronizer reaches node y. Node z is then guarded by an agent sent down from node x. If y is a node of type $T(j)$ where $j > 1$, by step 2.1, from the the set of available agents on the homebase, enough agents are sent to node y to clean the children of y in the broadcast tree. If y is a node of type $T(1)$, the agent on it is enough to clean the only child at level $i+1$. So, when agents on y are sent to level $i+1$, all $z \in N(y) - NT(y)$ are already guarded by an agent each. After all agents on y are sent down to level $i+1$, each of $NT(y)$ is guarded by an agent. Now, all neighbors of y are either clean or guarded by an agent.

Notice that if node y is a leaf, $NT(y)$ is empty and $N(y) - NT(y) = N(y)$. For any $z \in N(y)$, by lemma 11, $\exists x$ such that $z \in NT(x)$, and $\widehat{x} < \widehat{y}$. So all y's neighbors at level $i+1$ are guarded by an agent when the synchronizer reaches it. Furthermore, by induction hypothesis, all its neighbors at level $i-1$ are clean. When the agent on y is set free, all neighbours of y are either clean or guarded. ∎

Theorem 1. *During the cleaning process all nodes are clean and a clean node will not be recontaminated.*

PROOF First of all, by the cleaning strategy, the cleaning is proceeding level by level. So when it reaches level d, all nodes are clean. The fact that a clean node will not be recontaminated directly follows from Lemma 12 ∎

Complexity. We first prove that the number of agents that are available at the beginning, are enough to perform our strategy up to the level d.

Let $\binom{d-1}{\frac{d}{2}-2} + \binom{d}{\frac{d}{2}} + 1$ be the number of initially available agents.

Lemma 13. *There are enough agents to perform the cleaning from the root to level* 1.

This is trivial because only d agents are needed.

We now calculate the number of agents that are taken from the set of available agents before performing the cleaning from level l to level $l+1$.

Lemma 14. *Before cleaning from level l ($l > 0$) to level $l+1$, $\binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$ extra agents are requested from the root by the synchronizer.*

PROOF In step 2.1, each node of type $T(k)$ needs k agents to be sent to the k links of the broadcast tree to level $l+1$. Since one agent is on each node of level l, $k-1$ extra agents for a node of type $T(k)$ will be sent from the root. By lemma 2, there are $\begin{pmatrix} d-k-1 \\ l-1 \end{pmatrix}$ nodes of type $T(k)$ at level $l > 0$. So totally $\sum_{k=2}^{d-l}(k-1)\begin{pmatrix} d-k-1 \\ l-1 \end{pmatrix}$ extra agents are sent from the root to level l while cleaning from level l to level $l+1$.

$\sum_{k=2}^{d-l}(k-1)\begin{pmatrix} d-k-1 \\ l-1 \end{pmatrix} = \sum_{i=1}^{d-l-1} i \begin{pmatrix} d-(i+1)-1 \\ l-1 \end{pmatrix}$ (i = k-1)

$= \sum_{i=1}^{d-l-1} \begin{pmatrix} i \\ 1 \end{pmatrix}\begin{pmatrix} d-i-2 \\ l-1 \end{pmatrix} = \sum_{i=1}^{d-L-2} \begin{pmatrix} i \\ 1 \end{pmatrix}\begin{pmatrix} d-i-2 \\ L \end{pmatrix}$ (L = l-1)

$= \sum_{i=1}^{d-2} \begin{pmatrix} i \\ 1 \end{pmatrix}\begin{pmatrix} d-i-2 \\ L \end{pmatrix} = \sum_{i=0}^{d-2} \begin{pmatrix} i \\ 1 \end{pmatrix}\begin{pmatrix} d-i-2 \\ L \end{pmatrix}$

Referring to ([13]) we have that: $\sum_{i=0}^{d-2} \begin{pmatrix} i \\ 1 \end{pmatrix}\begin{pmatrix} d-i-2 \\ L \end{pmatrix} = \begin{pmatrix} d-1 \\ L+2 \end{pmatrix}$;

thus, $\sum_{i=0}^{d-2} \begin{pmatrix} i \\ 1 \end{pmatrix}\begin{pmatrix} d-i-2 \\ L \end{pmatrix} = \begin{pmatrix} d-1 \\ l+1 \end{pmatrix} = \begin{pmatrix} d \\ l+1 \end{pmatrix} - \begin{pmatrix} d-1 \\ l \end{pmatrix}$

$= \begin{pmatrix} d \\ l+1 \end{pmatrix} - \begin{pmatrix} d \\ l \end{pmatrix} + \begin{pmatrix} d-1 \\ l-1 \end{pmatrix}$ ∎

Alternatively, it is simple to calculate the extra number of agents in another way. Each node of type $T(k)$ needs k agents to be sent to the k links of the broadcast tree to level $l+1$. In total, $\sum_{k=1}^{l} \binom{k}{1}\binom{d-k-1}{l-1} = \binom{d}{l+1}$ agents will be sent by the $\binom{d}{l} - \binom{d-1}{l-1}$ nodes of type $T(k)$ with $k \geq 1$. Each of these nodes has already an agent on it, so from the root $\binom{d}{l+1}$ - $\binom{d}{l}$ + $\binom{d-1}{l-1}$ extra agents will be requested by the synchronizer.
We now show that

Theorem 2. *There are enough agents to perform the cleaning from level l to level $l+1$.*

PROOF By induction. There are enough agents to clean level 1 (see Lemma 13). Let us assume that there are enough agents to clean level l, $l \geq 1$, using our strategy. At this point we have $\binom{d}{l}$ + 1 active agents (where the synchronizer is counted too), every node of level l is guarded by one agent; all the other agents are available. We show that there are enough agents to clean level $l+1$ using our strategy.

By lemma 14, before cleaning from level l to level $l+1$, extra $\binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$ agents are taken from the set of available agents. This number has been calculated so to give exactly k agents for each tree of type $T(k)$, which is just enough to proceed with the cleaning of level l. In fact, by induction hypothesis, each node of level l is already guarded by one agent before the extra agents come. So in total, $\binom{d}{l} + 1 + \binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1} = \binom{d}{l+1} + \binom{d-1}{l-1} + 1$ agents are active at level l. By our strategy, the $\binom{d-1}{l-1}$ agents on the leaves do not participate in the cleaning of level $l+1$, but the other $\binom{d}{l+1}$ are just enough to move to level $l+1$ on the broadcast tree.

In addition, it is well known that

$$\begin{aligned} & \max_{1\le l\le d-1}\left(\begin{array}{c} d \\ l+1 \end{array}\right)+\left(\begin{array}{c} d-1 \\ l-1 \end{array}\right) \\ = & \left(\begin{array}{c} d \\ \frac{d}{2}+1 \end{array}\right)+\left(\begin{array}{c} d-1 \\ \frac{d}{2}-1 \end{array}\right) \\ = & \left(\begin{array}{c} d \\ \frac{d}{2} \end{array}\right)+\left(\begin{array}{c} d-1 \\ \frac{d}{2}-2 \end{array}\right) \qquad for\ l=\frac{d}{2}-1\ or\ l=\frac{d}{2} \end{aligned}$$

Therefore, $\left(\begin{array}{c} d \\ \frac{d}{2} \end{array}\right)+\left(\begin{array}{c} d-1 \\ \frac{d}{2}-2 \end{array}\right)+1$ is the maximum number of agents required by the procedure and corresponds to the cleaning of the two central levels.

When the synchronizer sets free the agent on the last node (a leaf) of level l, level $l+1$ is cleaned and the only active agents are the ones covering level $l+1$. ∎

It is well known that $\left(\begin{array}{c} d \\ \frac{d}{2} \end{array}\right)+\left(\begin{array}{c} d-1 \\ \frac{d}{2}-2 \end{array}\right)=O(\frac{n}{log\ n})$ ([13]), thus we have that

Theorem 3. *In a d-dimensional hypercube, our strategy employs $O(\frac{n}{log\ n})$ agents to clean the network.*

We now calculate the total number of movements needed for the entire process.

Theorem 4. *The total number of moves performed by the agents and the synchronizer is $O(nlog\ n)$.*

PROOF To compute the global number of moves we have to take into account the ones performed by the agents and those performed by the synchronizer.

The number of moves performed by the agents :

It takes $2l$ moves for an agent to arrive at a leaf of level l from the root and go back to the root. By lemma 3, there are $\left(\begin{array}{c} d-1 \\ l-1 \end{array}\right)$ leaves at level l. So totally, there are $\sum_{l=1}^{d} 2l\left(\begin{array}{c} d-1 \\ l-1 \end{array}\right)$ moves performed by the agents. To compute this quantity first note that $\left(\begin{array}{c} d \\ l \end{array}\right)$ is the number of

nodes at level l, we have $\sum_{l=0}^{d} \begin{pmatrix} d \\ l \end{pmatrix} = 2^d = n$, which is also a known result [13]. It follows that $\sum_{l=1}^{d} \begin{pmatrix} d-1 \\ l-1 \end{pmatrix} = \sum_{l=0}^{d-1} \begin{pmatrix} d-1 \\ l \end{pmatrix} = 2^{d-1} = \frac{n}{2}$.

We have now to compute:

$\sum_{l=1}^{d} l \begin{pmatrix} d-1 \\ l-1 \end{pmatrix}$

$= 1 \begin{pmatrix} d-1 \\ 0 \end{pmatrix} + ... + (\frac{d}{2} - 1) \begin{pmatrix} d-1 \\ \frac{d}{2}-2 \end{pmatrix} + \frac{d}{2} \begin{pmatrix} d-1 \\ \frac{d}{2}-1 \end{pmatrix} + (\frac{d}{2}+1) \begin{pmatrix} d-1 \\ \frac{d}{2} \end{pmatrix} + (\frac{d}{2}+2) \begin{pmatrix} d-1 \\ \frac{d}{2}+1 \end{pmatrix} + ... + d \begin{pmatrix} d-1 \\ d-1 \end{pmatrix}$

Using the property that $\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} a \\ a-b \end{pmatrix}$ we may group terms in pair and obtain:

$\sum_{l=1}^{d} l \begin{pmatrix} d-1 \\ l-1 \end{pmatrix}$

$= (d+1) \begin{pmatrix} d-1 \\ 0 \end{pmatrix} + (d+1) \begin{pmatrix} d-1 \\ 1 \end{pmatrix} + ... + (d+1) \begin{pmatrix} d-1 \\ \frac{d}{2}-2 \end{pmatrix} + (d+1) \begin{pmatrix} d-1 \\ \frac{d}{2}-1 \end{pmatrix}$

$= (d+1) \sum_{l=0}^{\frac{d}{2}-1} \begin{pmatrix} d-1 \\ l \end{pmatrix}$

We already know $\sum_{l=0}^{d-1} \begin{pmatrix} d-1 \\ l \end{pmatrix} = 2^{d-1} = \frac{n}{2}$; we also know that $\sum_{l=0}^{d-1} \begin{pmatrix} d-1 \\ l \end{pmatrix} = 2 \sum_{l=0}^{\frac{d}{2}-1} \begin{pmatrix} d-1 \\ l \end{pmatrix}$. Thus, $(d+1) \sum_{l=0}^{\frac{d}{2}-1} \begin{pmatrix} d-1 \\ l \end{pmatrix} = (d+1)2^{d-2} = \frac{n}{4}(log\ n + 1)$.

The total number of moves performed by the agents is $\sum_{l=1}^{d} 2l \begin{pmatrix} d-1 \\ l-1 \end{pmatrix} = \frac{n}{2}(log\ n + 1) = O(nlog\ n)$.

The number of moves performed by the synchronizer :

1. Go to the root to get more agents. Totally, there are $\sum_{l=1}^{d-3} l = \frac{(d-3)(d-2)}{2} = O(log^2 n)$ moves.

Number of agents	$O(n/log\ n)$
Number of moves	$O(nlog\ n)$
Time Complexity	$O(nlog\ n)$

Table 2.1: Model As

2. Go to the first node of each level. Totally, there are $\sum_{l=1}^{d-2} l = \frac{(d-1)(d-2)}{2} = O(log^2 n)$ moves.

3. Navigate within each level to get to the next node. At level l, the synchronizer needs to navigate at most $2l$ edges if $l \leq \frac{d}{2}$, otherwise $2d - 2l$ edges to reach the next node at the same level. So totally, at most there are $\sum_{l=1}^{\frac{d}{2}} 2l \begin{pmatrix} d \\ l \end{pmatrix} + \sum_{l=\frac{d}{2}+1}^{d-2} (2d - 2l) \begin{pmatrix} d \\ l \end{pmatrix} = 4\sum_{l=1}^{\frac{d}{2}-1} l \begin{pmatrix} d \\ l \end{pmatrix} + d \begin{pmatrix} d \\ \frac{d}{2} \end{pmatrix} - 2d = O(nlog\ n)$.

4. Go down with each agent to clean a node at the next level in the broadcast tree and then come back. Each edge of the broadcast tree is traveled twice by the synchronizer. So totally there are $2(2^d - 1) = 2(n - 1)$ moves.

Totally, the number of moves performed by the cleaning process is $O(nlog\ n)$. ∎

We now consider the ideal time complexity of the cleaning strategy. We remind that for computing the ideal time, we assume that it takes 1 unit of time for an agent to traverse an edge and that computation starts at time 0.

Theorem 5. *The cleaning strategy takes $O(nlog\ n)$ time units.*

PROOF The cleaning process is carried out sequentially by the synchronizer. The time required is then equal to the number of moves of the synchronizer. ∎

Summary

Please refer to the table 2.1 for the general results for the cleaning strategy in this section. Notice that in the cleaning strategy, only the synchronizer is actually cleaning the network. All

the other agents are just there to guard the nodes to avoid any recontamination. Hence, the time cost is very expensive.

2.2.2 Asynchronous Model with Cloning Capability – ASCLO

This model is similar to the previous model. The synchronizer is defined in the same way as in the previous model. In addition, the agents have cloning power.

Cleaning strategy

This cleaning strategy is very similar to Strategy AS. However, when the synchronizer reaches a node which requires more than one agent to clean its children in the broadcast tree, the agent on it clones needed agents instead of having the synchronizer go to the root to get more agents.

Initially, one agent as well as the synchronizer are placed at the root.

1. From the root to level 1

1.1 - The agent cleans the node and clones $d-1$ new agents and then the synchronizer guides one agent to each child of the root, makes sure it arrives at the destination and then come back to the root.

2. From level l to level $l+1$ [level l has one agent per node]

2.1 - The synchronizer follows the lexicographical order of nodes at level l to process the cleaning. Let $\widehat{x_1^l}, \ldots, \widehat{x_m^l}$ $(m = \binom{d}{l})$ be the nodes of level l lexicographically ordered. When the synchronizer arrives at a node $\widehat{x_i^l}$ of type $T(k), k \geq 1$, the agent there clones $k-1$ new agents; and then k agents are sent to the k neighbors at level $l+1$ along the edges in the broadcast tree guided by the synchronizer; when an agent and the synchronizer arrive at a node at level $l+1$, the agent guards this node and the synchronizer goes back to the next node $\widehat{x_{i+1}^l}$.

2.2 - When a leaf is reached by the synchronizer, the agent is terminated. Notice that when the synchronizer reaches the last node of level l, the only active agents are the ones covering level $l + 1$.

Correctness and Complexity

Correctness. Since the cleaning strategy is very similar to that of the previous model, the correctness of this strategy is proved in the same way as the correctness of that strategy. Thus, we have that

Theorem 6. *During the cleaning process all nodes are clean and a clean node will not be recontaminated.*

Complexity. Since agents have cloning power, they can clone new agents whenever it is needed, thus there are always enough agents to clean the network. We now calculate the total number of agents used for the entire cleaning process.

Theorem 7. *The total number of agents used to perform the entire cleaning for the hypercube is* $\frac{n}{2}$.

PROOF The number of agents for the entire cleaning process is equal to the number of leaves in the tree because exactly one agent terminates at each leaf. The binary representation of all leaves starts with 1 in the most significant bits, where that of none leaves starts with 0. We know there are 2^{d-1} of each. So the number of leaves is $\frac{n}{2}$. ∎

Theorem 8. *The number of moves performed by the synchronizer and the agents is* $O(nlog\ n)$.

PROOF The number of moves performed by the agents is reduced comparing to the model where the agents have no cloning power. Exactly one agent traverses an edge in the broadcast tree T. There are $n - 1$ edges in T. So the number of moves by the agents is $n - 1$ (Recall that without cloning power $O(nlog\ n)$ moves were needed). The reason for the reduction in the number of moves is that cloning allows to create the agents when they are needed. So unnecessary moves from the root are avoided.

Number of agents	$\frac{n}{2}$
Number of moves	$O(nlog\ n)$
Time Complexity	$O(nlog\ n)$

Table 2.2: MODEL ASCLO

The number of moves performed by the synchronizer is reduced too. The synchronizer does not need to go to the root to get more agents. However, it still has to navigate from one node to the next node at the same level while cleaning. This navigation takes at most $\sum_{l=1}^{\frac{d}{2}} 2l\binom{d}{l} + \sum_{l=\frac{d}{2}+1}^{d-1}(2d-2l)\binom{d}{l} = 4\sum_{l=1}^{\frac{d}{2}-1} l\binom{d}{l} + d\binom{d}{\frac{d}{2}}$ moves, which is $O(nlog\ n)$ of moves. The number of moves performed by the synchronizer to accompany the agents to the next level is $2(n-1)$, the same as that in the previous model.

Totally, the number of moves by the entire process is reduced, it takes $O(nlog\ n)$ moves. ∎

Theorem 9. *The cleaning process takes $O(nlog\ n)$ times.*

PROOF Since we are considering ideal time, it takes an agent one time to perform one move. Hence, the total times for the entire process is the same as the number of moves, which is $O(nlog\ n)$. ∎

Summary

The general results for the cleaning strategy in this model are given in Table 2.2. Basically, the cleaning strategy is the same as Strategy AS, but uses more agents. Since the needed agents are cloned in place instead of getting them from the root, the number of moves performed by the agents is reduced; however the overall cost is still $O(nlog\ n)$ because the number of moves of the synchronizer dominates.

2.2.3 Asynchronous Model with Visibility – ASVI

In this model, we assume agents have visibility power. This extra capability enables agents to correctly act without the need of being coordinated by the synchronizer in the asynchronous

environment; agents make their own decision regarding the action to take solely on the basis of their local knowledge.

Cleaning strategy

The agents are still moving on the broadcast tree, but they do not have to follow the order imposed by the coordinator like in the model with a synchronizer. In fact, the agents on node x can proceed to clean the children of x in the broadcast tree when they "see" that the other neighbours of x are either clean or guarded. In other words, the agents on node x can proceed to clean the bigger neighbours of x when they "see" that the smaller neighbours of x are either clean or guarded.

All the agents are initially located at the source; they are identical and autonomous, and they all follow the same local rule.

Rule 1. *Rule for the agents on node x of type $T(k)$ ($0 \leq k \leq d$):*

- *If the number of agents on x is less that 2^{k-1}, then the agents wait on x.*
- *When 2^{k-1} agents are on x: when all the smaller neighbours of x are clean or guarded, one agent moves to the bigger neighbour of type $T(0)$; 2^{i-1} agents move to each of the bigger neighbours of type $T(i)$ for $0 < i < k$; if there are no bigger neighbours, terminate.*

Recall that the whiteboard is used for storing the status of the node (*contaminated*, *clean*, *guarded*), the current number of agents present at the node. During the strategy the agents can access the local whiteboard and the whiteboards of the neighbours. Which agent goes to which node is also determined by accessing the whiteboard. Also in this case, no more that $O(\log n)$ bits are required for the whiteboard.

Correctness and Complexity

Correctness. We first compute the number of agents needed to clean a node of type $T(d)$ by our strategy.

Theorem 10. *The total number of agents needed to clean the d-dimensional hypercube is* $\frac{n}{2}$.

PROOF By definition the strategy is sending from level 0 to level 1 one agent for $T(0)$ and 2^{i-1} agents for each $T(i)$ for a total of $1 + \sum_{i=1}^{d-1} 2^{i-1} = 1 + \sum_{i=0}^{d-2} 2^i = 2^{d-1} = \frac{n}{2}$ agents. Moreover, a node of type $T(k)$ receives 2^{k-1} agents and 2^{k-1} is exactly the number of agents needed to continue the cleaning strategy, in fact, $2^{k-1} = 1 + \sum_{i=1}^{k-1} 2^{i-1}$. Thus, with $\frac{n}{2}$ agents the strategy can be completed. ∎

We now prove that the strategy is correct; i.e., that the network is clean and once a node has been cleaned, it will never be recontaminated.

Lemma 15. *In our strategy, when agents leave a node in* $C(i)$ *(leaving it unguarded) all its smaller neighbours are either clean or guarded.*

PROOF Let us consider a node x in C_i. By the cleaning strategy, when an agent arrives at node x, it cleans the node. By theorem 10 every node will have eventually enough agents to continue the cleaning process, by definition of the cleaning rule, the agents on x move to the bigger neighbours only when all other neighbours are clean or guarded. ∎

Theorem 11. *During the cleaning process the agents clean all nodes and a clean node will not be recontaminated.*

PROOF First of all, by the cleaning strategy, the edges and nodes traversed by the agents form the broadcast tree. All the nodes are visited by an agent. The fact that a clean node will not be recontaminated directly follows from lemma 15. ∎

Complexity. We now consider the time complexity of the cleaning strategy.

Theorem 12. *Cleaning the entire network takes* $O(\log n)$ *time units.*

PROOF We will prove it by showing that at time i, all nodes in C_i are clean; only the agents in C_i can move to clean the bigger neighbors, which are in C_j for $j > i$. We prove it by induction.

Base case: at time $i = 0$, all the agents are placed on node (00...00) (the source). First notice that, since there is no agent on any other node at time 0, only the agents in C_0 might move at this

time. By the cleaning strategy, the agents clean the source, then they move to clean the d bigger neighbors, which are in C_j for $0 < j \leq d$. Node (00..00) becomes clean at time 0; obviously, no recontamination can occur to it. The claim then holds for $i = 0$.

Assume the claim is true up to time i, $i \geq 0$. We show that it holds at time $i + 1$.

By induction hypothesis and theorem 11 all the nodes in C_k are clean for any $0 \leq k \leq i$. The agents ever on them have left. Let node x be an arbitrary node in C_{i+1}. By lemma 9, exactly one smaller neighbour of x is in C_k for $k \leq i$. By induction hypothesis, at time k, the agents arrive at x and clean it upon arrival. So at time $i + 1$, every node in C_{i+1} is guarded by at least an agent; and, thus, all smaller neighbours of any node in C_{i+1} are clean or guarded. So at time $i + 1$ every agent in C_{i+1} execute the algorithm; they go to clean the bigger neighbors which, by lemma 9, are in C_j with $j > i + 1$. Since the nodes in C_{i+1} are already cleaned by their guarding agents upon arrival, together with the fact that a clean node will not be recontaminated by theorem 11, we know that at time $i + 1$ the nodes in C_{i+1} become clean after the agents on them move.

Notice that, by our cleaning algorithm, the other agents in C_j for $i + 1 < j \leq d$ cannot move because one or more of their smaller neighbours are not guarded. Let us consider any node y in C_j on which there are agents. By lemma 10, there exists one smaller neighbor z of y which is in C_j too and a smaller neighbor w of z is in C_{j-1}, $j - 1 \geq i + 1$. We know that the agents on z come from its smaller neighbor w which is in C_{j-1}. If $j - 1 = i + 1$, in other words, w is in C_{i+1}, the agents on w move at time $i + 1$. So at time $i + 1$ no agent is on z yet. If $j > i + 1$, even if there are agents on w, by induction hypothesis, they have not moved before time $i + 1$. Hence, in any case, z is not guarded at time $i + 1$ and the agents on y cannot move because at least one of its smaller neighbors is not guarded. ∎

We now calculate the total number of moves performed by the agents.

Theorem 13. *The number of moves performed by the agents for the entire cleaning is* $O(n \log n)$.

PROOF All the agents start from the source and each terminates on a leaf. There are $\binom{d-1}{l-1}$ leaves at level $l > 0$, thus the total number of moves is: $\sum_{l=1}^{d} l\binom{d-1}{l-1} = O(n \log n)$ (the calculation is similar to the one of the proof of theorem 4). ∎

Number of agents	$n/2$
Number of moves	$O(n \log n)$
Time Complexity	$log\ n$

Table 2.3: Model AsVi

Summary

The general results for the cleaning strategy in this model are given in Table 2.3. Compared to Strategy As, AsClo, Strategy AsVi works in a more powerful model where the agents are allowed to "see" the states of their neighboring nodes. It has the advantage that computation is local, i.e., there is no need of a coordinator. In addition, it is much faster ($log\ n$) and requires the same number of moves ($O(n \log n)$). However it used more agents ($\frac{n}{2}$).

2.2.4 Asynchronous Model with Visibility and Cloning – AsViClo

In this model, we assume agents have both visibility and cloning power. Due to the agents' visibility, the synchronizer becomes unnecessary.

Cleaning strategy

No synchronizer exists to centrally control the cleaning. All the agents are autonomous and they all follow the same rule.

Rule 2. *For an agent on node x:*

- *If I see that not all the smaller neighbors of x are clean or guarded, I wait on x;*
- *When all the smaller neighbors of x are clean or guarded: I clone enough agents to clean the bigger neighbors of x and then I send one agent per edge to clean each of the bigger neighbors. If I see that all the smaller neighbors are guarded or clean and there are no bigger neighbors, I terminate.*

Initially, one agent is placed at node (00...00). It cleans node (00...00) and clones $d-1$ new agents. In this way d agents are available on the node and then one agent per edge is sent to

clean each of the d neighbors. When an agent arrives at a node, it cleans this node and then executes Rule 2.

Correctness and Complexity

Correctness. We now prove that our cleaning strategy is correct; i.e., that the network is clean and once a node has been cleaned, it will never be recontaminated.

Theorem 14. *During the cleaning process the agents clean all nodes and a clean node will not be recontaminated.*

PROOF First of all, by the cleaning strategy, the edges and nodes traversed by the agents form the broadcast tree. All the nodes are clean. Next we need to show that a clean node will not be recontaminated. Let us consider a node x in C_i. By the cleaning strategy, when an agent arrives at node x, it cleans the node. By Rule 1, the agent on x clone and go only when its smaller neighbors are clean and guarded. Because of the cloning power, an agent can always clone enough agents, if needed, to clean the bigger neighbors. After the agents arrive at the bigger neighbors, if any, all neighbors of x are either clean or guarded. No recontamination can occur to node x. However, it is possible that the agents on several nodes clone and go at the same time. That means the agent on one of x's smaller neighbor might clone and go at the same time when the agent on x clones and goes. However, this concurrent cleaning won't introduce recontamination because after the agent on the smaller neighbor leaves, the smaller neighbor becomes clean instead of guarded. The clean smaller neighbor y of x remains clean because all y's neighbors are either clean or guarded or changing from guarded to clean. So, no recontamination occurs. If an agent is on a node without any contaminated neighbor, it terminates. No recontamination can occur in this case. ∎

Complexity. Since agents have cloning power, they can clone new agents whenever they are needed, thus there are always enough agents to clean the network. We now calculate the total number of moves performed by agents and the number of agents used for the entire process.

Theorem 15. *The total number of agents used to perform the cleaning is* $\frac{n}{2}$.

PROOF Since all agents have to terminate, instead of counting how many agents ever created, we count how many agents terminate. By the cleaning strategy, we know that the edges and nodes traversed by the agents form the broadcast tree; every node is visited by exact one agent. By Rule 2, the agent on a node terminates only if the node does not have bigger neighbors. It is easy to see there are 2^{d-1} such nodes. Each of these 2^{d-1} nodes is visited by exactly one agent, which later terminates on it. Hence the total number of agents used is equal to $2^{d-1} = \frac{n}{2}$. ∎

Theorem 16. *The number of moves performed by agents for the entire cleaning is* $n - 1$.

PROOF The proof is trivial because by the cleaning strategy, we know that the edges and nodes traversed by the agents form the broadcast tree; each edge in the broadcast tree is traversed by one agent only. ∎

We now consider the ideal time complexity of the cleaning strategy. We remind that the ideal time is computed with the assumption that it takes 1 unit of time for an agent to traverse an edge and that computation starts at time 0.

Theorem 17. *Cleaning the entire network takes log* n *time units.*

PROOF We will prove it by showing that at time i, all nodes in C_i are clean; only the agents in C_i can clone and go to clean the bigger neighbors which are in C_j for $j > i$. By induction.
Base case: at time $i = 0$, one agent is placed on node (00...00). By the cleaning strategy, the agent cleans the node; and then clones $d - 1$ new agents and each agent is sent down to clean one of the d bigger neighbors which are in C_j for $0 < j \leq d$. Node (00..00) becomes clean at time 0; obviously, no recontamination can occur to it. Since there is no agent on any other node at time 0, only the agent in C_0 can clone and go. The claim holds for $i = 0$.
Assume the claim is true up to time i, $i \geq 0$. We show that it holds at time $i + 1$.
By induction hypothesis and Theorem 14 all the nodes in C_k are clean for any $0 \leq k \leq i$. The agents ever on them have cloned and gone. Let node x be an arbitrary node in C_{i+1}. By lemma 9, exactly one smaller neighbor of x is in C_k for $k \leq i$. By induction hypothesis, at time k, exactly one agent arrives at x and cleans it upon arrival. So at time $i + 1$, every node in C_{i+1} is guarded by an agent; and thus all smaller neighbors of any node in C_{i+1} are clean or guarded.

Number of agents	$n/2$
Number of moves	$n-1$
Time Complexity	$log\ n$

Table 2.4: Model AsViClo

So Rule 1 applies to every agent in C_{i+1} at time $i+1$; they clone and go to clean the bigger neighbors which, by lemma 9, are in C_j with $j > i+1$. Since the nodes in C_{i+1} are cleaned by their guarding agents upon arrival and the fact that a clean node will not be recontaminated by theorem 14, at time $i+1$, when the agents on them move and the nodes in C_{i+1} become clean. Notice that other agents in C_j for $i+1 < j \leq d$ do not satisfy the condition of Rule 1. Let us consider any node y in C_j on which there is an agent. By lemma 10, there exists one smaller neighbor z of y which is in C_j too and a smaller neighbor w of z is in C_{j-1}, $j-1 \geq i+1$. The agent on z has to come from its smaller neighbor w in C_{j-1}. If $j-1 = i+1$, in other words, w is in C_{i+1}, the agent on w clones and moves at time $i+1$. So no agent is on z at time $i+1$. If $j > i+1$, even if there is an agent on w, by induction hypothesis, it has not cloned before time $i+1$. Hence in any case, z is not guarded at time $i+1$. Rule 1 cannot apply to the agent on y. ∎

Summary

The general results for the cleaning strategy in this model are given in Table 2.4. It is easy to see that this model is very similar to Model AsVi. Instead of placing all agents needed for the entire cleaning on the homebase and moving them around, agents are created when needed. The cloning power of agents reduces the number of moves performed by the agents from $O(nlog\ n)$ to $n-1$, which is optimal in terms of number of moves in the contiguous search because it takes at least $n-1$ moves to visit every node in the hypercube.

Remark Since every agent is autonomous, it is very difficult if not impossible to reuse an agent when it is not needed, instead to let it terminates.

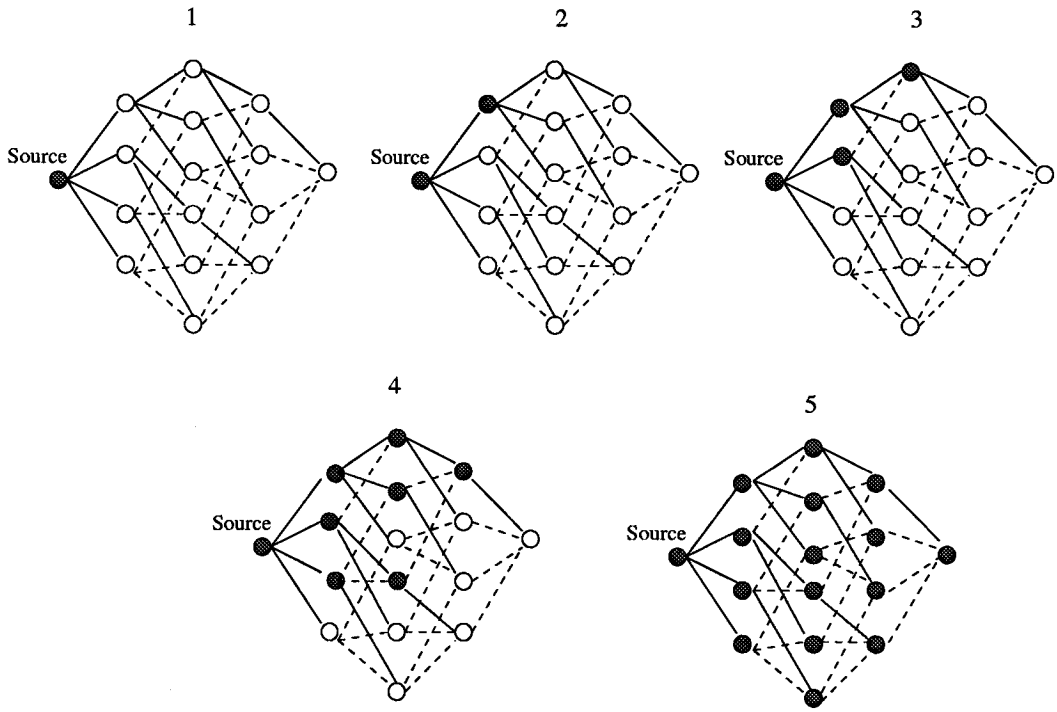


Figure 2.9: The execution of Strategy SYCLO on a 4-dimensional hypercube.

2.2.5 Synchronous Model with Cloning – SYCLO

We now examine the cleaning strategy in a synchronous model where the agents have only cloning capability. No synchronizer is necessary.

Cleaning strategy

All agents are autonomous and they all follow the same rule. Recall that $m(x)$ is the position of the most significant bit of x.

Rule 3. *For an agent on node x:*
If time $t = m(x)$, I clone enough agents to be able to clean the bigger neighbors and then one agent per edge is sent to clean each of these bigger neighbors; If x does not have bigger neighbors, I terminate; Otherwise, for $t < m(x)$, I wait on x.

At time 0, one agent is placed at node (00...00); it cleans node (00...00) and clones $d-1$ new agents; and then one agent per edge is sent to clean each of the d neighbors. At time i, an

agent arrives at a node, it cleans the node and then executes Rule 3.

Figure 2.9 shows the order in which the nodes get cleaned with our strategy. Notice that nodes are not cleaned sequentially; several nodes, in fact, could be cleaned independently.

Correctness and Complexity

Correctness. We now prove that, with this strategy, the hypercube is correctly cleaned. We first give some useful properties of the cleaning strategy.

Lemma 16. *At time i, there is one agent on every node in C_i.*

PROOF By Induction. Base case, at time $i = 0$, one agent is placed on node (00..00) by the strategy. Since there is only one node, node (00..00), in C_0, the claim is true.
Assume the claim is true up to time i for $i \geq 0$. We show that at time $i + 1$, there is one agent on every node in C_{i+1}.
Let node x be any node in C_{i+1}. By lemma 9, exactly one smaller neighbor of x is in C_j, for some $j \leq i$. Let us call this smaller neighbor z. By induction hypothesis, at time j, there is one agent on z. Hence, by Rule 3, at time j, one agent from z is sent to clean x. Since $j < i+1$ and by Rule 2, only at time $t = i + 1$ the agent on x can clone and go, so the agent waits on x from time $j + 1$ to $i + 1$. Hence, at time $i + 1$, the agent is still on x. Since x is any node of C_{i+1}, the claim holds for time $i + 1$. ∎

Lemma 17. *At time i, for $0 \leq i < d$, all the agents on the nodes of C_i clone and they go to clean the bigger neighbors; at time d, all the agents on the nodes of C_d terminate.*

PROOF By lemma 16, at time i, there is one agent on every node of C_i. By definition, every node in C_i has the most significant bit in the i-th position. Hence, by Rule 2, at time i, for $0 \leq i < d$, every agent on nodes in C_i clones enough new agents and then each agent goes to clean one of its bigger neighbors. By definition, the nodes in C_d do not have bigger neighbors. Hence, by Rule 2, at time d, every agent on nodes in C_d terminates. ∎

We now prove that our cleaning strategy is correct; i.e., that the network is clean and once a node has been cleaned, it will never be recontaminated.

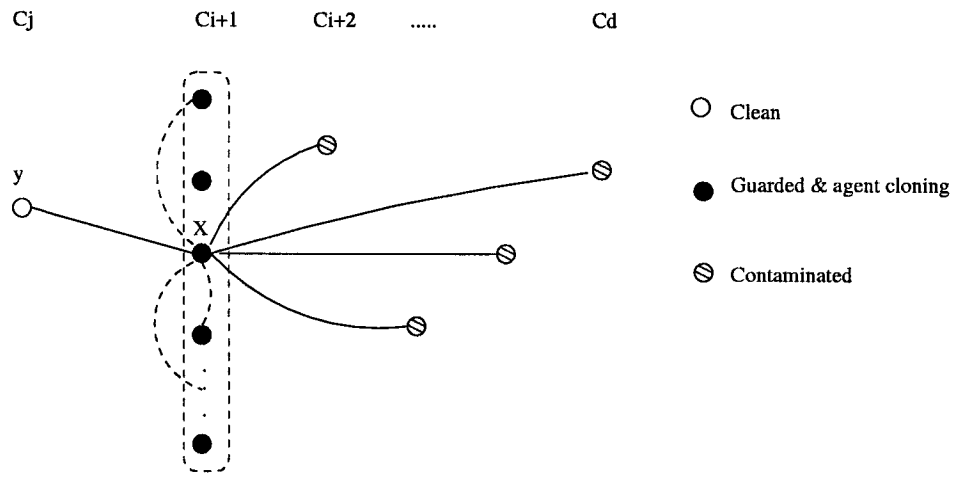


Figure 2.10: The agents in C_{i+1} cloning at time $i+1$.

Lemma 18. *At time i, all nodes in C_j for $0 \leq j \leq i$ are clean.*

PROOF By induction. Base case: it is trivial for $i = 0$.

Assume the claim holds at time i, $i \geq 0$. Let us show that it is true at time $i+1$.

By lemma 16, at time $i+1$, every node in C_{i+1} is guarded by an agent. (see Figure 2.10) Let x be an arbitrary node of C_{i+1}. By lemma 9, one smaller neighbor of x is in C_j, $j \leq i$, let us call it y; and all other smaller neighbors of x, if any, are in C_{i+1} and guarded by agents at time $i+1$. By induction hypothesis, at time i, nodes in C_j, $j \leq i$ are clean. Thus, y is clean. The only contaminated neighbors of x are the bigger neighbors. By the cleaning strategy, at time $i+1$, the agent on x clones enough new agents and then each of them goes to clean one of the bigger neighbors; x becomes clean. Notice that the guarded smaller neighbors of x which are in C_{i+1} become clean too at time $i+1$. Since all contaminated neighbors of x are now guarded by agents (one agent for each contaminated neighbor), even though no agent is left to guard x, no recontamination can occur to it. Hence, all nodes in C_{i+1} are clean simultaneously at time $i+1$, and no recontamination can occur to them.

We show at time $i+1$, no recontamination can occur to clean nodes in C_j, $j \leq i$. Let z be any clean node. (see Figure 2.11) At time $i+1$, z's neighbors in group C_l for $l < i+1$ are clean and all neighbors in group C_h for $h \geq i+1$ are guarded by agents. Let us call the neighbor in C_{i+1} w. By the cleaning strategy, at time $i+1$, the agent on w is cloning and then agents go to

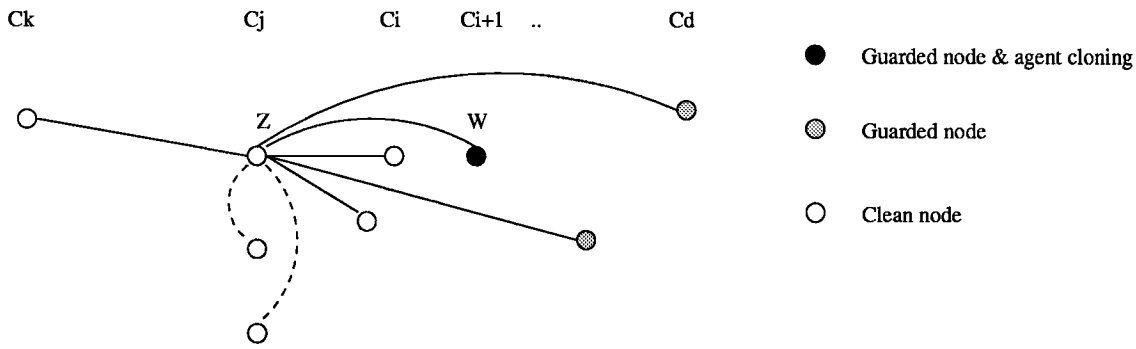


Figure 2.11: The clean node x at time $i+1$.

clean all contaminated neighbors of w; w becomes clean; from the above proof w will not be recontaminated. At time $i+1$, among all z's neighbors, only w's state changes from guarded to clean. So no recontamination can occur to z. ∎

It follows from lemma 18 that

Theorem 18. *During the cleaning process the agents clean all nodes and a clean node will not be recontaminated.*

Complexity. Since the agents have cloning power, they can clone new agents whenever it is needed. As a consequence that there are always enough agents to clean the network. We now calculate the total number of agents used for the entire process.

Theorem 19. *The total number of agents used to perform the cleaning is* $\frac{n}{2}$.

PROOF Since all agents have to terminate, instead of counting how many agents are created in each step, we can calculate how many agents terminate. By lemma 17, agents in C_i for all $0 \leq i < d$ are cloning some agents; only at time d, every agent on each node of C_d terminates. By lemma 16, at time d there is one agent on each node of C_d. Hence the total number of agents used is equal to the number of nodes in C_d, which is $2^{d-1} = \frac{n}{2}$. ∎

Theorem 20. *The number of moves performed by the agents for the entire cleaning process is* $n-1$.

PROOF Let us consider a node x in C_i for any $0 \leq i < d$. By Rule 3, at time i, the agent on x clones; and then one agent per edge goes to clean one of the bigger neighbors of x. By

Number of agents	$n/2$
Number of moves	$n-1$
Time Complexity	$log\ n$

Table 2.5: Model SYCLO

definition of bigger neighbor, it is easy to see that the number of bigger neighbor of every node in C_i is $(d-i)$. So the agents on x perform $d-i$ moves. By lemma 7, there are 2^{i-1} nodes in C_i. By Rule 3, at time i, only agents on nodes in C_i clone and go. Hence, at time i, the number of moves performed by the agents is $(d-i)2^{i-1}$. At time 0, the number of moves is d by the agent and its clone on node (00..00); at time d, the agents terminate, they don't perform any move. So the total number of moves performed by the agents is

$$\begin{aligned} & d+ \quad \sum_{i=1}^{d-1}(d-i)2^{i-1} \\ = \quad & d+ \quad d\sum_{i=1}^{d-1}2^{i-1} - \sum_{i=1}^{d-1}i2^{i-1} \\ = \quad & d+ \quad d(2^{d-1}-1) - (d-2)2^{d-1} - 1 \\ = \quad & n-1 \quad \blacksquare \end{aligned}$$

Alternatively, theorem 20 can be proved by noticing that the set of edges and nodes traversed by the agents forms the broadcast tree of the hypercube. One edge in the broadcast tree is traversed by only one agent. Hence the number of moves performed by the agents is $n-1$, which is the number of edges in the broadcast tree.

It follows from lemma 18 that

Theorem 21. *The cleaning strategy requires exact log n time steps.*

Summary

The general results for the cleaning strategy in this model are given in Table 2.5.

The cleaning strategy in this model is similar to Strategy ASVICLO. In these two strategies, agents are autonomous and have to wait until all their smaller neighbors are clean or guarded.

With the synchronous assumption, the movement of agents depends on time. We have proved that at the time for agents to move, all its smaller neighbors are clean or guarded. However without the synchronous assumption, agents' visibility can help to determine if the smaller neighbors are clean or guarded. So either the synchronous assumption of the model or visibility of agents serves the same purpose: determining when an agent can clone and go.

We have seen three models where the agents have cloning power. In Model AS, agents are not autonomous; in fact their moves are sequenced by the synchronizer. The synchronizer acts like a central controller to sequence the entire cleaning process. In the other two models, agents are autonomous and both cleaning strategies perform a distributed cleaning, which can introduces concurrent process.

Remark Another variation of the model that should now be studied is when the agents have cloning power only in an asynchronous environment. We do not have a synchronizer to control the entire cleaning process, neither do the agents have visibility power or are synchronous. Strategy EDASCLO in the next section (Edge Search), with little modifications, can be employed for this model.

2.3 Edge Search

Initially, all the edges of the hypercube are contaminated. We want to obtain a clean network by using a cleaning strategy. Three edge search models are discussed in this section: Asynchronous model with a synchronizer, Synchronous model where the agents have cloning power, Asynchronous model where the agents have cloning power.

2.3.1 Asynchronous Edge Search – EDAS

The synchronizer is the same as the one defined in Model AS of node search. All nodes of the hypercube are organized level by level; level i contains all the nodes whose binary representation contains i ones.

Cleaning strategy

In the edge search, all the edges between level l and level $l+1$ have to be cleaned while cleaning from level l to level $l+1$. With minor modification, the cleaning strategy for the asynchronous node search with a synchronizer can be employed obtaining the same complexity for solving the Edge Search problem. The main idea is before cleaning the edges in the broadcast tree, the synchronizer can first clean the non-tree edges leading to the next level one by one; if the synchronizer reaches a leaf, it cleans all the edges leading to the next level and then set the agent there free. The differences are in step 2.2 and 2.3

2.2 - When k agents are on a node of type $T(k)$ at level l, they are sent down in the broadcast tree to its children in level $l+1$ helped by the synchronizer. Let $\widehat{n_1^l}, \ldots, \widehat{n_m^l}$ $(m = \binom{d}{l})$ be the nodes of level l lexicographically ordered. The synchronizer sequentially sends down the agents from level l to the level $l+1$, one per edge of the broadcast tree, following the lexicographical order of the $\widehat{n_i}$. *For each node $\widehat{n_i}$, the synchronizer will first clean the non-tree edges, if any, leading to level $l+1$ to clean them;* it will then send all the agents from $\widehat{n_i}$. It will make sure that they arrive at their destination, before proceeding with node $\widehat{n_{i+1}}$.

2.3 - When a leaf of level l is reached by the synchronizer, *the synchronizer cleans all the edges leading to the next level* and then the agent on the leaf can be set free and becomes available. We assume that as soon as an agent becomes available, it will go to the root. Notice that when the synchronizer reaches the last node of level l, the only active agents are the ones covering level $l+1$.

Correctness and Analysis

Correctness. Similarly, We can prove that, with this modified strategy, the hypercube is correctly cleaned.

Theorem 22. *During the cleaning process all the edges are clean and a clean edge will not be recontaminated.*

Number of agents	$O(n/log\ n)$
Number of moves	$O(nlog\ n)$
Time Complexity	$O(nlog\ n)$

Table 2.6: Model EDAS

PROOF The proof is very similar to the proof of Theorem 1. The only new part that needs to be proved is that the non-tree edges cleaned by the synchronizer before sending the agents from level l to level $l+1$ will not be recontaminated. This is clearly true because the non tree edges are leading to a node of level $l+1$ that is already guarded by an agent previously sent by the synchronizer (this is a consequence of Lemma 11). Thus, after the synchronizer has cleaned a non-tree edge, the intruder will not be able to recontaminate it. ∎

Complexity. The number of agents required is obviously the same as the number employed by Strategy AS.

As for the number of moves, we have to add two moves by the synchronizer per non-tree edge, for a total of $O(nlog\ n)$ extra moves. However, the increment of the number of moves by the synchronizer does not change the order of magnitude of the move complexity. Thus, the overall number of moves is still $O(nlog\ n)$.

Finally, also for the time complexity, we have to add $O(n \log\ n)$ units of time since the synchronizer is moving sequentially.

Thus,

Theorem 23. *Our strategy uses $O(\frac{n}{log\ n})$ agents to clean the network. The total number of moves performed by the agents is $O(nlog\ n)$. The cleaning strategy takes $O(nlog\ n)$ units of time.*

Summary

The general results for the cleaning strategy in this model are given in Table 2.6.

2.3.2 Synchronous Edge Search with Cloning – EDSYCLO

In this model, all nodes of the hypercube are organized level by level; level i contains all the nodes whose binary representation contains i ones. The synchronizer is not necessary.

Cleaning strategy

The idea is similar to flooding. The cleaning proceeds level by level starting from level 0. For a node x at level l, it has l neighbors at level $l-1$ and $d-l$ neighbors at level $l+1$. Since all edges have to be cleaned, by our strategy, the agents from the l neighbors at level $l-1$ clean the l edges of x and arrive at x at time $l-1$. At time $l-1$, all edges between level $l-1$ and l are cleaned.

Initially, one agent is placed at node (00...0).

1 - At time 0, the agent clones $d-1$ new agents and the agents can now clean the d contaminated edges and arrive at nodes at level 1.

2 - At time i, if not enough agents are on a node to clean the next level, then one agent clones the needed agents; if more agents are on a node than required to clean the next level $l+1$, only the needed number of agents stay alive, the others terminate; if exactly $(d-l)$ agents are on a node, they move one per edge to the next level.

Notice that since all agents are identical, in step 2, we need to elect a leader among the agents on each node. The leader will clone or ask some other agents to terminate and then send the agents to clean the contaminated edges, it moves too.

Correctness and Analysis

Correctness. We now prove that our cleaning strategy is correct; i.e., that all edges are clean and once an edge has been cleaned, it will never be recontaminated.

Lemma 19. *At time l, all the edges before level l are clean.*

PROOF By induction. The base case (time 0, 1) are trivial. Assume at time $l-1$ all edges before level $l-1$, $l-1 \geq 1$ are clean. We show that at time l, all edges before level l are clean. By induction hypothesis, at time $l-1$ all edges before level $l-1$ are clean. At time l, by lemma 20, every node at level l is guarded by l agents. So no recontamination can occur to any clean edge before level $l-1$. By the cleaning strategy, exact one agent per each edge goes to level $l+1$ at time l. Hence the edges between level $l+1$ and l are clean at time l; and each node at level $l+1$ is guarded by $l+1$ agents. So it is not possible to recontaminate the just clean and the originally clean edges. So at time l, all edges before level l are clean. The claim holds. ∎

It follows from lemma 19 that

Theorem 24. *During the cleaning process all the edges are clean and a clean edge will not be recontaminated.*

Complexity. Since agents have cloning power, there are always enough agents to perform the cleaning from level l to level $l+1$. We first show how many agents are required for the cleaning strategy.

Lemma 20. *At time l, all agents are at level l and exactly l agents are on each node.*

PROOF Each node at level l has a path of length l to node $(00...0)$. It takes the agents 1 time step to travel on an edge. By the cleaning strategy, agents will arrive at level l at the end of time step $l-1$ and will be there at the beginning of time step l. Since a node at level l has l edges connecting to nodes at level $l-1$, one agent comes from each edge, totally l agents arrive at it. ∎

Theorem 25. *In a d-dimensional hypercube, $\frac{d}{2}\binom{d}{\frac{d}{2}}=O(n)$ agents suffice to perform the edge cleaning by the above cleaning strategy.*

PROOF By the cleaning strategy, if a node has less agents than required to clean the next level, then agents clone themselves. For a node at level l, $l < \frac{d}{2}$, by lemma 20, there are l agents on it, but the required number of agents to clean the next level is $d-l$, which is larger than l. So agents have to clone themselves. Only for nodes at level $\frac{d}{2}$, there are exactly $\frac{d}{2}$ agents on each,

Number of agents	$\frac{d}{2}\binom{d}{\frac{d}{2}} = O(n)$
Number of moves	$\frac{n}{2} log\ n$
Time Complexity	$log\ n$

Table 2.7: Model EDSYCLO

so no clone is created and no agent terminates. For a node at level l, $l > \frac{d}{2}$, more agents are on it than required, no clone is created and extra agents terminate. Therefore, the maximal number of agents required to perform the cleaning strategy is $\frac{d}{2}\binom{d}{\frac{d}{2}}$, where $\binom{d}{\frac{d}{2}}$ is the number of nodes at level $\frac{d}{2}$. ∎

We now calculate the total number of movements and the time needed for the entire process.

Theorem 26. *The number of moves performed by the agents during the cleaning process is* $\frac{n}{2} log\ n$

PROOF By the cleaning strategy, every edge is cleaned by exactly one agent. Totally, there are $\frac{d}{2}n$ edges. So, the number of moves performed by agents is $\frac{d}{2}n = \frac{n}{2} log\ n$. ∎

Finally, we calculate the ideal time complexity of the cleaning strategy. It directly follows from lemma 19 that

Theorem 27. *The cleaning process requires* $log\ n$ *time steps.*

Summary

The general results for the cleaning strategy in this model are given in Table 2.7.

2.3.3 Asynchronous Edge Search with Cloning – EDASCLO

In this model, all nodes of the hypercube are organized level by level; level i contains all the nodes whose binary representation contains i ones. The synchronizer is not necessary.

Cleaning strategy

The cleaning starts from level 0 and proceeds level by level. The agents from the root are sent to all the neighboring nodes, then the cleaning strategy is like flooding in a hypercube. The edges

via which agents arrive at a node are clean. So the agents are only sent to clean those edges from which no agents arrived. In an asynchronous environment, agents might arrive at a node at different time. So it is important to allow the agents to know when no more agents arrive. Fortunately, we know l agents from nodes at level $l-1$ arrive on a node at level l. Equivalently, we know that the l edges of a node at level l are clean. The whiteboard of a node contains a variable indicating the number of incident clean edges. Each agent increases the variable upon arrival at a node; after that it checks if the number of clean edges incident on the node is equal to l; if not, more agents are expected to come, so the agents have to wait; if yes, l agents have arrived and l edges are clean, the last arrival agent can act as a leader to see if any clone is needed or to terminate some agents that are not needed for cleaning the edges of the next level. All agents are autonomous. They execute the following rule.

Rule 4. *For an agent that arrives at a node x at level l:*

I count the number of clean edges (m) leading to nodes at level $l-1$; then I compare m with l: If $m \neq l$, I wait for an order to move or terminate; otherwise, I compare m with the number of contaminated edges of x which is $d-l$. If $m > d-l$, I ask any $m-d+l$ agents to terminate,(possibly also myself); else if $m < d-l$, I clone enough agents such that $m = d-l$; if I am still alive, I send all $d-l$ agents (including myself) to the contaminated edges (one agent per edge).

Initially, one agent is placed at the node $(00..00)$. It clones $d-1$ new agents and then they move along the edges, one per edge to the d neighbors. All agents follow Rule 4.

Notice that by the cleaning strategy, the cleaning is progressing from level 0 to d.

Correctness and Analysis

Correctness. We now prove that our cleaning strategy is correct; i.e., that all edges are clean and once a edge has been cleaned, it will never be recontaminated.

Theorem 28. *During the cleaning process the agents clean all edges and a clean edge will not be recontaminated.*

PROOF First of all, the hypercube is connected, by the cleaning strategy, all the edges are cleaned. Next, we prove that a clean edge will not be recontaminated.
Let us consider a node x at level l. A node at level l is only connected to nodes at level $l-1$ and $l+1$. It is easy to see that x has l neighbors at level $l-1$ and $d-l$ neighbors at level $l+1$. By the cleaning strategy, when l edges of x leading to nodes at level $l-1$ are clean, we know l agents arrive on it. Only then, by cloning or terminating agents, exactly $d-l$ agents are sent (one agent per contaminated edge) to level $l+1$. So the $d-l$ contaminated edges are clean and thus all edges of x are clean. Since each neighbor of x at level $l+1$ is guarded by an agent, no recontamination can occur to the just clean edges as well as the originally clean edges. ∎

Complexity Since agents have cloning power, there are always enough agents to perform the cleaning. We now calculate the total number of agents used for the entire process and the number of moves performed by them.

Theorem 29. *The total number of agents used for the cleaning is* $\frac{d}{2}\binom{d}{\frac{d}{2}}=O(n)$.

PROOF For a node at level l, the number of its neighbors is l at level $l-1$ and $d-l$ at level $l+1$. For $0 \leq l < \frac{d}{2}$, $l < d-l$; for $l = \frac{d}{2}$, $l = d-l$; and others, $l > d-l$. So agents on nodes at level l create agents by cloning for $0 \leq l < \frac{d}{2}$. When $l = \frac{d}{2}$, the number of agents is maximal. Since every node at level $\frac{d}{2}$ has $\frac{d}{2}$ agents and there are $\binom{d}{\frac{d}{2}}$ nodes at level $\frac{d}{2}$, the total number of agents is $\frac{d}{2}\binom{d}{\frac{d}{2}}$, which is $O(n)$.

Theorem 30. *The total number of moves performed by the agents is* $\frac{n}{2}log\ n$.

PROOF By theorem 28, no recontamination can occur, each edge is cleaned by exact one agent. There are $\frac{n}{2}log\ n$ edges totally. So the total number of moves performed by the agents is $\frac{n}{2}log\ n$. ∎

We now consider the ideal time complexity of the cleaning strategy. The computation starts at time 0. It immediately follows by the cleaning strategy that

Theorem 31. *Cleaning the entire network takes* $log\ n$ *time units.*

Number of agents	$\frac{d}{2}\binom{d}{\frac{d}{2}} = O(n)$
Number of moves	$\frac{n}{2} log\ n$
Time Complexity	$log\ n$

Table 2.8: Model EDASCLO

Summary

The general results for the cleaning strategy in this model are given in Table 2.8.

The cleaning strategy is very similar to Strategy EDSYCLO. Both cleaning strategies require the same number of agents, have the same number of moves performed by the agents and take the same time.

2.4 Overall Summary

In this chapter, we have looked at two variants of the cleaning problem in hypercube. In each variant, different cleaning strategies are developed and analyzed in depth. We summarize the differences among the several models of the cleaning strategies we have studied. The comparisons are shown in Table 2.9.

For the node search, if the synchronizer is employed, the cleaning process is sequential and thus rather slow because only the synchronizer is cleaning and all the other agents are just there to guard the nodes to avoid recontamination. We can add more power to the agents such as cloning and visibility. In general, agents' cloning power reduces the number of moves because instead of fetching agents from the root, agents are cloned on the spot when needed. Agents' visibility is useful when no synchronizer exists and agents have to cooperate to clean the network while avoiding recontamination. Both in Model ASVICLO and Model SYCLO, the number of moves performed by the agents is optimal; the time complexity of both cleaning strategies is optimal, which is $O(log\ n)$. However, the total number of agents used in these two models is $(\frac{n}{2})$, which is higher than $O(\frac{n}{\log n})$ in Model AS.

For the edge search, if the synchronizer is employed, with minor modification, Strategy AS can work for the edge search. Thus, these two cleaning strategies have the same complexity

	Model	Agents	Moves	Time
Node Search	AS	$O(\frac{n}{\log n})$	$O(nlog\ n)$	$O(nlog\ n)$
	ASCLO	$\frac{n}{2}$	$O(nlog\ n)$	$O(nlog\ n)$
	ASVI	$\frac{n}{2}$	$O(nlog\ n)$	$log\ n$
	ASVICLO	$\frac{n}{2}$	$n-1$	$log\ n$
	SYCLO	$\frac{n}{2}$	$n-1$	$log\ n$
Edge Search	EDAS	$O(\frac{n}{\log n})$	$O(nlog\ n)$	$O(nlog\ n)$
	EDASCLO	$O(n)$	$\frac{n}{2}log\ n$	$log\ n$
	EDSYCLO	$O(n)$	$\frac{n}{2}log\ n$	$log\ n$

Table 2.9: Comparison of different models.

in terms of number of agents as well as number of moves and time. In the other two models, the cleaning strategies are very similar. The number of moves in these two models are $\frac{n}{2}log\ n$, which is optimal; and the time complexity is $O(log\ n)$, optimal too.

Open Problems

Although we have developed some cleaning strategies in different models for cleaning the hypercube, there are still many open problems. The main open question is the following:

What is the minimal number of agents that suffice to clean the hypercube? What is its corresponding cleaning strategy?

Chapter 3

Butterfly

The butterfly is a regular network with constant degree and is one of the most common interconnection networks and architectural basis for parallel computers.

In this chapter, we are going to study the cleaning problem by mobile agents in the butterfly. Two variations of this problem are considered and the corresponding cleaning strategies are developed. In both cases, we are only interested in developing monotone strategies. Since our node search strategies depend on the edge search strategies, we first develop and analyze the strategies of the edge search in detail and then briefly discuss the node search. At the end, we will briefly give some observations about the agents' visibility and cloning power.

3.1 Definitions and Terminology

The d-dimensional butterfly has $n = 2^d(d+1)$ nodes and $d2^{d+1}$ edges. The nodes are arranged in 2^d rows numbered from 0 to $2^d - 1$ and in $d+1$ levels numbered from 0 to d. Each row is represented by a d bit binary number $b_0 b_1 ... b_{d-1}$. Notice that b_0 is the most significant bit. Each node in the network is labeled distinctly as (r, l), where r is an d-bit binary number and denotes the row of the node, and l is the level of the node ($0 \leq l \leq d$). A node labeled (r, l) (where $l < d$) is connected to nodes $(r, l+1)$ and $(r^l, l+1)$ where r^l denotes only bit b_l of r is complemented. Clearly, all the nodes at level i ($0 < i < d$) are connected only to nodes of level $i-1$ and level $i+1$; the nodes at level 0 are connected only to nodes of level 1; the nodes at

level d are connected only to nodes of level $d-1$.

We call the edge connecting node (r,l) and $(r,l+1)$ $(0 \leq l < d)$ the straight edge and the edge between node (r,l) and $(r^l,l+1)$ the cross edge. Every node has two straight edges and two cross edges, except the nodes at level 0 and d haves one straight edge and one cross edge. Consider a node (r,l) $(0 < l < d)$. The straight edge connecting to a node at level $l+1$ is labeled as edge 1; the cross edge leading to a node at level $l+1$ is labeled as edge 2; the other straight edge connecting to a node at level $l-1$ is labeled as edge 3 and the last edge is labeled as edge 4. A node at level 0 has only edge 1 and 2; a node at level d has only edge 3 and 4.

An example of a 5-dimensional butterfly is illustrated in Figure 3.1. In the example, the two nodes $(00001,0)$ and $(10001,1)$ are connected via a cross edge. 10001 is 00001 with the b_0 complemented. The labels of edges are given in Figure 3.2.

The butterfly and the hypercube are quite similar in structure. The hypercube is just a folded up butterfly. A d-dimensional hypercube can be obtained from a d-dimensional butterfly by merging all butterfly nodes that are in the same row and then removing the extra copy of each edge.

The butterfly network has a recursive structure. A d-dimensional butterfly contains two $(d-1)$-dimensional butterfly. By removing the level 0 nodes of the d-dimensional butterfly and the edges incident on them, we get two $(d-1)$-dimensional butterfly. These two $(d-1)$-dimensional butterfly are identical but to distinguish them, the upper one is called block B_u and the lower one is called block B_l. In addition, the nodes at level 0 of the butterfly belong to block B. Block B_u is connected to block B_l via B.

One of the nice features of the d-dimensional butterfly is that there is a unique path of length d that allows a message to route from any node of level 0 to any node of level d. The algorithm to find the path is based on bit flipping, which is similar to the routing in the hypercube. Upon reaching node (r,l), $l < d$, the message with destination (R,d) takes the straight edge to $(r,l+1)$ if the bit b_l of both r and R are the same. Otherwise, it takes the cross edge to node $(r^l,l+1)$. In the example of Figure 3.1, Let us see how a message is routed from node $(00001,0)$ to $(10000,5)$. First, the b_0s of 00001 and 10000 are different, so the cross edge is taken and the message reaches node $(10000,1)$. Next, compare the b_1 of 10000 and 10000,

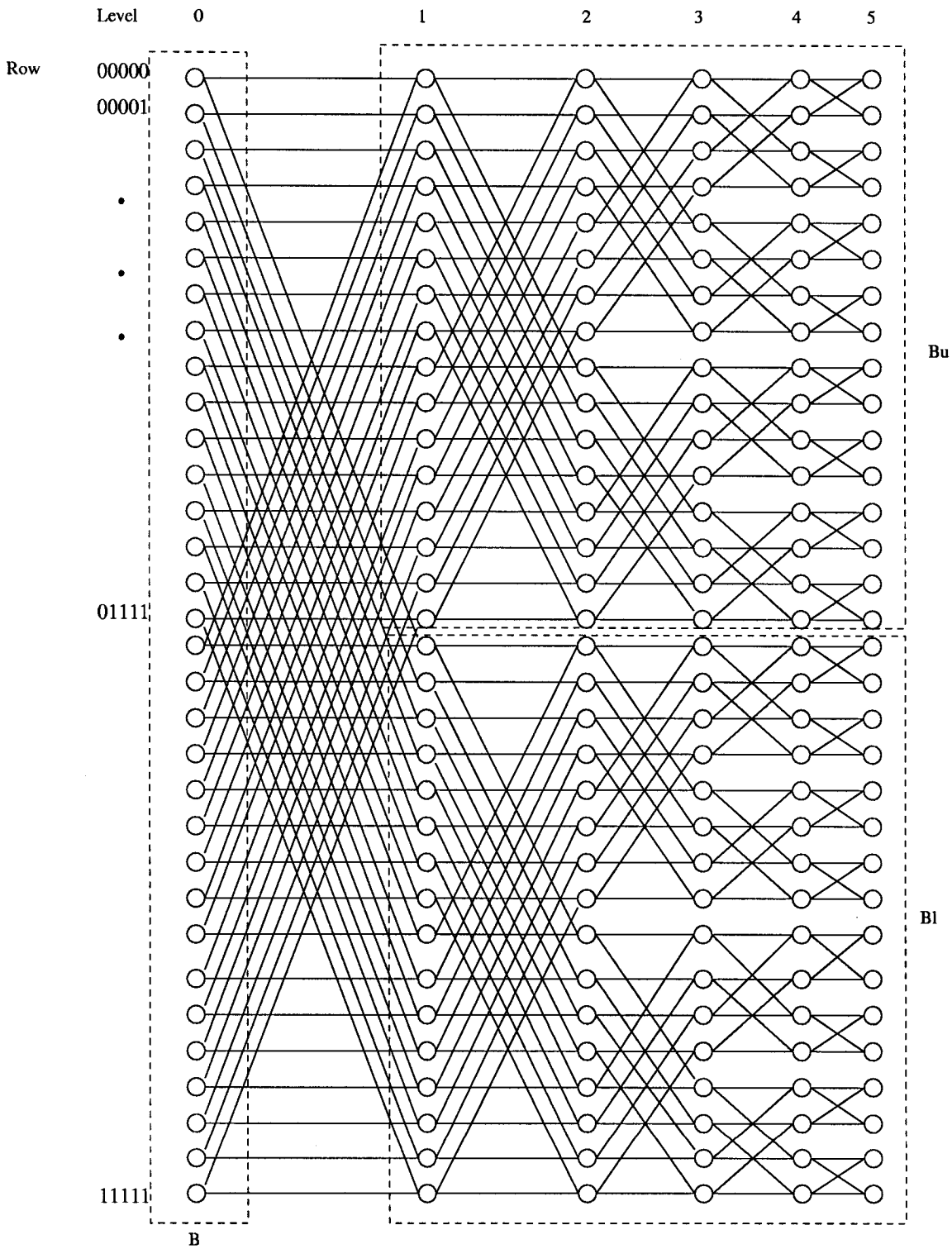


Figure 3.1: The 5-dimensional Butterfly.

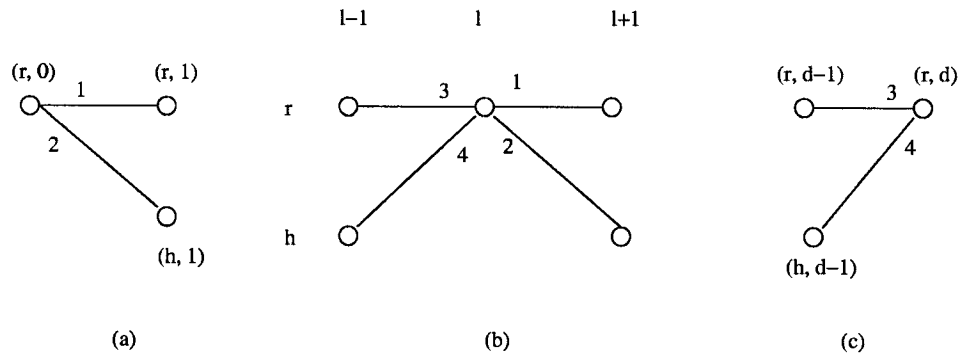


Figure 3.2: The edge labeling.

which are the same, so the straight edge is taken and the message reaches node $(10000, 2)$; etc. until the message reaches $10000, 5)$, the destination.

3.2 Edge Search

All edges of the butterfly are initially contaminated. A team of identical and autonomous agents are deployed to clean the network. An agent on a node (r, l) can distinguish the edges leading to level $l-1$ and/or level $l+1$ by looking at the labels. While an agent moves across an edge, it cleans the edge. When it arrives at a node, it writes to the white broad to indicate it has cleaned which edge. In this section, we will consider both the synchronous and asynchronous models.

3.2.1 Synchronous Edge Search

In this model, we first develop a cleaning strategy; then we show that our strategy is correct; i.e, that the butterfly is clean; we also analyze the complexity of the strategy.

Cleaning strategy

Due to the recursive structure of a butterfly, we can employ enough agents to clean block B_u and then reuse these agents to clean block B_l. We want our cleaning to start from level d of block B_u and then proceed level by level until agents reach level 0 and then the cleaning strategy can proceed to clean block B_l using the same team of agents. However, if initially we deploy

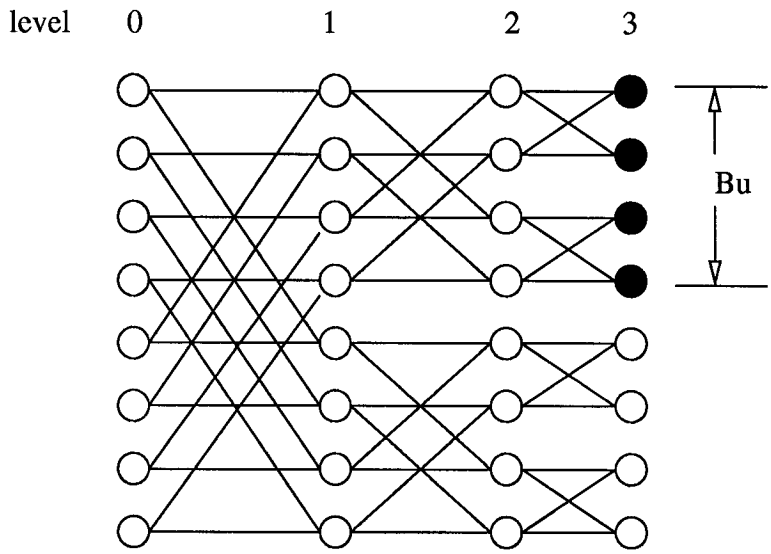


Figure 3.3: The 3-dimensional butterfly cleaning starts at level 3.

the agents at level d of block B_u, the cleaning strategy would not be contiguous because the clean edges would not form a connected subgraph at the beginning(see Figure 3.3). Thus, we choose node $(0, 1)$ as the home base for the agents and deploy enough agents on it and then have them move from the home base to level d of block B_u. We call this process *deployment stage*. However, in order to avoid recontamination and keep the strategy monotone, we have to leave agents on the visited nodes to guard the clean edges. So while the agents move toward level d, some agents are left to guard the visited nodes. In general, one agent is enough to guard one endpoint of a clean edge. At the end of the deployment stage, all visited nodes and the clean edges form a full binary tree T and every node of T is guarded by one agent except the home base, which is guarded by two.

In a synchronous system a group of agents arrive at a node simultaneously. The agents are identical, one of them will have to act as a coordinator for this local process, we will call this agent *the local leader*, which is defined as in our model. *During the deployment stage*, the local leader divides the local agents into two equal size groups and sends to the next level one group via the straight edge and the other group via the cross edge; then the local leader is left to guard the node. Two agents are left on the homebase. *During the cleaning stage*, if there are two agents on a node, the local leader sends the other agent to clean one of the contaminated edges,

and it goes to clean the other contaminated edge; if there is only one agent on a node, the agent is the local leader and it goes to clean the only contaminated edge.

We now describe the cleaning strategy more precisely.

Initially, enough agents are placed on the home base $(0, 1)$.

1. **From the home base to the level d of block B_u - Deployment Stage** [at the end of this stage level d of B_u has one agent per node at time $d-1$]

   1.1 - At time 0, two agents are left to guard the home base; the rest are divided into two equal size groups and one group moves across the straight edge and the other group moves across the cross edge to the level 2.

   1.2 - At time l $(0 < l < d)$, a group of agents are on a node of level $l+1$, one agent is left to guard the node, all the others are divided into two equal size groups, one group across the straight edge and the other across the cross edge arrive at level $l+2$.

2. **From the level d of block B_u to the level 0 of block B- Cleaning Stage**

   At time k $(d \leq k \leq 2d-1)$, the agents on the level $2d-k$ move to level $2d-k-1$. If one agent is on the node of level $2d-k$, the agent goes to clean the only contaminated edge; if two agents are on the node of level $2d-k$, the two agents go to clean the two contaminated edges independently.

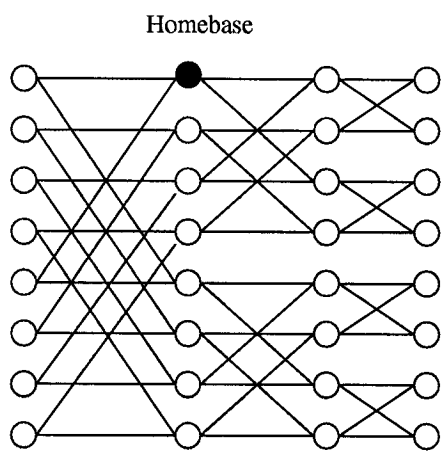
3. **From the level 0 of block B to the level d of block B_l - Cleaning Stage**

   At time j $(2d \leq j \leq 3d-1)$, the agents on the level $j-2d$ move to level $j-2d+1$. There are two agents on the node of level $j-2d$ for $j > 2d$, each of them goes to clean the two contaminated edges independently.

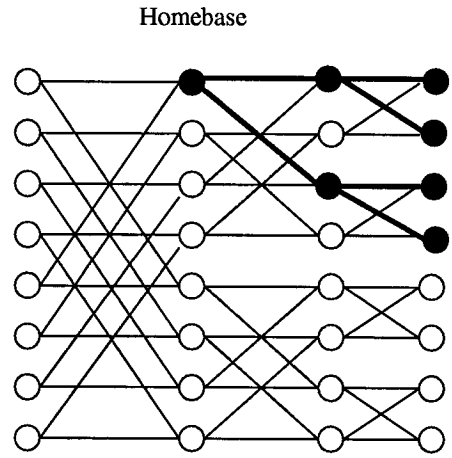
Please see Figure 3.4 for an execution of the cleaning strategy on the 3-dimensional butterfly.

Correctness and Analysis

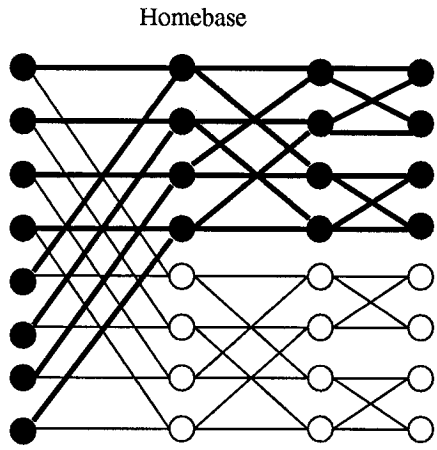
Correctness. We now prove that our cleaning strategy is correct; i.e., that all edges will be cleaned and that once an edge has been cleaned, it will never be recontaminated.



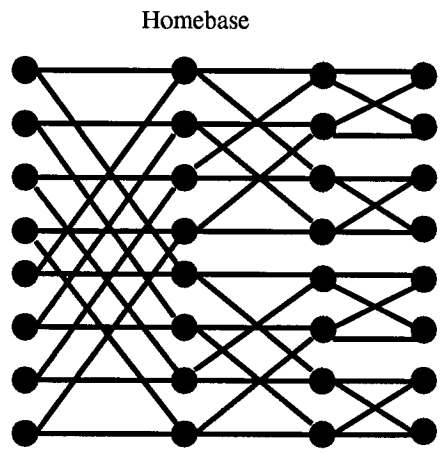
(a) Initially



(b) Full binary tree at time 2



(c) At time 5



(d) The network is clean at time 8.

Figure 3.4: An execution on the 3-dimensional butterfly.

Lemma 21. *At time $d-1$, the clean edges and the guarded nodes form a full binary tree T with height of $d-1$; an edge is cleaned and will not be recontaminated.*

PROOF By the cleaning strategy of step 1, the edges and the nodes visited by agent(s) form a binary full tree; at time $d-1$, the full binary tree rooted at the home base has height of $d-1$. Except the home base, which is guarded by two agents, every node visited by agent(s) is guarded by one agent; and at time $d-1$, every node of the full tree is guarded by one agent, except the root is by two. By the strategy, one endpoint of an edge is guarded by an agent and then a group of agent(s) clean it while moving across it; right after the cleaning, the other endpoint of the edge is guarded by another agent. Hence no recontamination can occur to the clean edge. ∎

Lemma 22. *At time k ($d \leq k < 2d-1$), 1) all the edges between level $2d-1-k$ and d of block B_u are clean; 2) every node at level $2d-k-1$ is guarded by one agent if it is a node of T (except the root), otherwise it is guarded by two agents.*

PROOF By Induction. Base case: time $k = d$. By step 1, at time $d-1$, every node of level d is guarded by one agent; one of the two edges is clean. At time d, every agent at level d cleans the only contaminated edge and then guards the other endpoint. No recontamination can occur to the clean edges. All edges between level d and $d-1$ are clean. 1) holds. Every node of T at level $d-1$ is guarded by one agent from step 1. By step 2, at time d, no agent moves to a node of T at level $d-1$. Thus, it is still guarded by one agent. Now let us consider a node x at level $d-1$ that is not a node of T. At time $d-1$, x has two contaminated edges leading to level d. By step 2, at time d, one agent moves across a contaminated edge and arrive at a node at level $d-1$; thus, two agents arrives at x at time d. Claim 2) holds.

Assume the claim 1) and 2) hold up to time $i-1$ for $d \leq i-1 \leq 2d-1$. We want to show they are true at time i.

At time i, by step 2, the agents from level $2d-i$ of B_u move to level $2d-1-i$. Let y be any node of level $2d-i$ of B_u. Let us first consider y is a node of T. By induction hypothesis, y is guarded by one agent; and y has two clean edges leading to level $2d-i+1$. The edge between y and its parent of T is clean too because it is cleaned at step 1 and both endpoints are guarded

by some agents. So, y has one contaminated edge. By step 2, the agent goes to clean the only contaminated edge and then guards the other endpoint. Thus, no recontamination can occur to the clean edges. Now let us consider that y is not a node of T. Obviously, y's neighbors at level $2d-i-1$ cannot be nodes of T. Thus, y has two contaminated edges leading to level $2d-i-1$ at time $i-1$. By induction hypothesis, y is guarded by two agents; and y has two clean edges leading to level $2d-i+1$. By step 2, at time i, the two agents go to clean the two contaminated edges independently. No recontamination to the clean edges can occur. All edges between level $2d-i$ and $2d-i-1$ are clean at time i. Since no recontamination occurs, all edges between level d and $2d-i-1$ are clean. Claim 1) holds.

We know that at time $i-1$ any node of T at level $2d-i-1$ is guarded by one agent from step 1. By the above proof, we also know that no agent moves along a clean edge. Thus, at time i no agent will arrive at a node of T at level $2d-i-1$. Every node of T at level $2d-i-1$ is still guarded by one agent at time i. Let us now consider a node z at level $2d-i-1$ that is not a node of T. Node z has two contaminated edges leading to level $2d-i$. By step 2, at time i, one agent moves across one contaminated edge; thus, two agents via these two contaminated edges arrive at z. Claim 2) holds. ∎

Lemma 23. *At time $2d-1$, every node of level 0 is guarded by one agent; all edges between B and B_u are clean.*

PROOF By lemma 22, at time $2d-2$, all edges between level 1 and d of B_u are clean; every node of B_u at level 1 is guarded by two agents. Notice that the root is guarded by two agents and it is the only node at level 1 of T. By step 2, at time $2d-1$, the two agents on each node clean the two contaminated edges independently and arrive at nodes at level 0. Thus, all edges between B and B_u are clean; no recontamination can occur. Since every node of level 0 is connected to one node of B_u, only one agent arrives at it at time $2d-1$. ∎

Now we want to show our strategy cleans B_l.

Lemma 24. *At time j ($2d \leq j \leq 3d-1$), all the edges between level 0 and level $j-2d+1$ are clean; two agents are on each node of level $j-2d+1$ of B_l.*

PROOF By lemma 23, at time $2d-1$, every node of level 0 is guarded by one agent. At time $2d$, by step 3, every agent of level 0 moves to level 1 of B_l. So, the edges between B and B_l are clean. By lemma 23, at time $2d-1$, all edges between B and B_u are clean. Therefore, all edges between level 0 and level 1 (both of B_u and B_l) are clean. Since a node at level 1 of B_l is connected to two nodes at level 0, two agents arrive at it from level 0 at time $2d$.
Assume the claim is true at time $i-1$, for $2d \leq i-1 < 3d-1$. We show it is true at time i. By induction hypothesis, at time $i-1$ two agents are on each node of level $i-2d$ of B_l; and all edges between level 0 and level $i-2d$ are clean. By step 3, two agents on any node of level $i-2d$ clean the two contaminated edges independently and then guard the other endpoints at level $i-2d+1$ of B_l. No recontamination can occur to the clean edges. All edges between level $i-2d$ and $i-2d+1$ of B_l are clean. By lemma 22 and induction hypothesis, all edges between level 0 and $i-2d+1$ are clean at time i. Since a node at level $i-2d+1$ is connected to two nodes at level $i-2d$, two agents arrive it from level $i-2d$ of B_l at time i. ∎
It directly follows from lemma 24 that

Theorem 32. *Our strategy cleans the butterfly and a clean edge will not be recontaminated.*

Complexity. We first calculate the number of agents needed to perform the cleaning of the butterfly with our strategy.

Theorem 33. *In a d-dimensional butterfly, our strategy employs 2^d agents to clean the network.*

PROOF By lemma 21, the agents guard a full binary tree T with height of $d-1$. Except the root is guarded by two agents, every node of T is guarded by one agent. There are 2^d-1 nodes of T. So the total number of agents is $1+2^d-1=2^d$. ∎

We now calculate the total number of movements needed for the entire process.

Theorem 34. *The total number of moves performed by the agents is $O(n\log n)$.*

PROOF By step 1, an agent guarding a node at level i of T travels totally $i-1$ edges from the home base. There are 2^{i-1} nodes at level i of T. So, the total number of moves by step 1 is

$\sum_{i=1}^{d}(i-1)2^{i-1} = d2^d - 2^{d+1} + 2$.

By step 2 and 3, since every contaminated edge is cleaned once, the total number of moves by the agents is the total number of edges of the butterfly minus the clean edges from step 1, which is $d2^{d+1}$ - $(2^d - 2)$. So the total number of moves is $d2^d - 2^{d+1} + 2 + d2^{d+1}$ - $(2^d - 2)$ = $(3d-3)2^d + 4$, which is $O(nlog\ n)$. ∎

Finally we consider the time complexity of the cleaning strategy. It follows directly from lemma 24 that

Theorem 35. *Our strategy take* $3d-1$ *time units to clean a* d*-dimensional butterfly.*

3.2.2 Asynchronous Edge Search

In this model, agents are also autonomous. The main idea of the cleaning strategy here is similar to the one of synchronous model. Similarly, in the Deployment Stage, agents move towards level d of block B_u and agents are left to guard the visited nodes. However, since it is an asynchronous model, in the Cleaning Stage, agents follow two local rules for actions instead of strictly following the global time.

Cleaning strategy

Enough agents are placed on the home base $(0,1)$. They move towards level d of B_u, one agent is left to guard each visited node (except for the root where two agents are left to guard it) and eventually every node of level d of B_u is guarded by one agent; and then cleaning starts from level d of B_u to level 0 of B and then to level d of B_l. The flow of strategy execution here is similar to that of the synchronous model.

Initially, 2^d agents are placed on the home base $(0,1)$.

1. **Deployment Stage** [Every node of level d of B_u is guarded by one agent]

   The agents on a node are divided into two equal size groups and move towards level d of B_u. Meanwhile, one agent is left to guard this node, except the home base, which is guarded by two agents.

2. Cleaning Stage

2.1 If an agent is on a node that has only one contaminated edge, this agent cleans the contaminated edge and arrives at the other endpoint;

2.2 If two agents are on a node with only two contaminated edges, these two agents clean the contaminated edges independently; otherwise, the agent has to wait on a node.

In the Deployment Stage, the group of agents move as a whole towards level d of B_u. Remind that movement is defined as *group movement* in our model of Chapter 1. In the Cleaning stage, agents follow the rule 2.1 and 2.2 for actions and an agent can know the number of contaminated edges and which edge is contaminated by accessing the whiteboard of the node.

The agents can distinguish the deployment stage and the cleaning stage if each is designed to carry a counter c. Initially $c = 1$ for all agents, which indicates that the agents are in the deployment stage; when an agent is left to guard the node as described by the strategy, it update $c = 0$, which indicates that now it is in the cleaning stage. Notice that since the agents are autonomous and asynchronous, some agents might be in the deployment stage and the others in the cleaning stage. But this fact does not affect our analysis of correctness and complexity.

Correctness and Analysis

Correctness. We now prove that our cleaning strategy is correct; i.e., that all edges will be cleaned and that once an edge has been cleaned, it will never be recontaminated.

Lemma 25. *Our strategy constructs a clean full binary tree T during the deployment stage.*

PROOF By step 1, during the deployment stage, our strategy constructs a full binary tree T. For any edge (u, v) of T, by step 1, one agent is left to guard the endpoint u, and a group of agent(s) clean this edge and right after that v is guarded by another agent. Edge (u, v) is cleaned and no recontamination can occur to it. ∎

Theorem 36. *Our strategy cleans the butterfly during the cleaning strategy and a clean edge will not be recontaminated.*

PROOF We first show that the edges of B_u are clean. By 1) of step 2, the agents from level d of B_u move to level 1; all the contaminated edges between level d and $d-1$ of B_u are clean; no recontamination can occur. Two agents arrive at every node $x \notin T$ at level $d-1$ of B_u because x has two contaminated edges leading to level d. For a node of T at level $d-1$ of B_u, it has two clean edges from Lemma 25; no agents arrive at it from level d, only one agent from the deployment stage is left on it. The cleaning is then proceeding towards level 0 of B. At each level $i > 0$ of B_u, a node is either a node of T or not. If it is a node of T, then by 1) of step 2, the agent on it cleans the only contaminated edge (except the two agents on the root, which clean the two contaminated edge); if it is not a node of T, by 2) of step 2, the two agents on it clean the two contaminated edges independently. All edges between level i and $i-1$ are clean; no recontamination can occur.

We now show that the edges between level 0 and level 1 are clean. Two agents will arrive at every node not in T at level 1 of B_u. The root (home base) has two agents too. By 2) of step 2, every agent moves on different edges to arrive at a node of level 0. In other words, every node of level 0 is guarded by one agent. All the edges between level 0 and level 1 of B_u are clean; no recontamination can occur. By 1) of step 2, every agent of level 0 moves to level 1 of B_l. Two agents will arrive at a node of level 1 of B_l. All the edges between level 0 and level 1 of B_l are clean; no recontamination can occur.

Finally, we show that the edges of B_l are clean. While cleaning the edges of B_l, two agents arrive at a node from two different edges; thus, this node has two clean edges. So, when a node has two agents, it has only two contaminated edges. By 2) of step 2, the two contaminated edges are clean and no recontamination can occur to it. ∎

Complexity. Our strategy is very similar to the one of synchronous model. It is immediate to see that both require the same number of agents and have the same number of moves.

Finally, for the time complexity, if we consider ideal time, our strategy takes the same time as that of the synchronous model. Thus,

Theorem 37. *Our strategy uses 2^d agents to clean the network. The total number of moves performed by the agents is $O(n\log n)$. The cleaning strategy takes $3d-1$ units of time.*

3.3 Node Search

Any Edge Search strategy solves also the Node Search problem. In the case of the butterfly we have not devised specific strategy for the Node Search problem. We leave as an open problem to find a strategy better than the one designed for Edge Search.

3.4 Conclusion and Observation

In this chapter, we have looked at the cleaning problem in the butterfly, focusing on the edge search. In the edge search, we study both the synchronous and asynchronous models. The cleaning strategies for them are very similar. In the synchronous model, the cleaning is proceeding strictly level by level. In the asynchronous model, some agents might be in the deployment stage and the others are in the cleaning stage. Our strategy for the synchronous edge search is employed for the synchronous node search; and the strategy for the asynchronous edge search is employed for the asynchronous node search. Please refer to Table 3.1 for the general results.

Observation

In all the cleaning strategies developed in this chapter, we deploy 2^d agents on the home base. Moving the agents from the home base to the level d of B_u introduces extra moves. If we want to reduce this extra moves, we can add the cloning power to agents as we did for the hypercube. With this capability, one agent is placed on the home base. In the deployment stage, it clones three new agents, one is sent across the straight edge to a node of level 2 and another across the cross edge to a node of level 2. Both the original agent and one new agent are left to guard the root. Hereafter, in the deployment stage, an agent arriving on a node creates two agents and sends each of them across different edges to the next level and it is left to guard the node. The agents stop cloning when they reach level d of B_u. The cleaning stage will be the same, and so is the number of agents. However, the number of moves performed by the agents with cloning power is reduced to $d2^{d+1}$, which is optimal in the edge search.

Agents' visibility is not useful in these cleaning strategies. When a node has two agents in the cleaning stage, by our strategy, these two agents arrive from different edges. Since recon-

Model	Agents	Moves	Time
Synchronous Edge Search	$2^d = \frac{n}{(d+1)}$	$O(n \log n)$	$3d-1$
Asynchronous Edge Search	$2^d = \frac{n}{(d+1)}$	$O(n \log n)$	$3d-1$
Synchronous Node Search	$2^d = \frac{n}{(d+1)}$	$O(n \log n)$	$3d-1$
Asynchronous Node Search	$2^d = \frac{n}{(d+1)}$	$O(n \log n)$	$3d-1$

Table 3.1: Comparison of different models

tamination is not possible in our strategy, the fact that two agents are on a node in the cleaning stage implies that two neighbors (or two edges) of the node are clean; if one agent is on a node of the full binary tree, it implicitly knows that the node's three other neighbors (or edges) are clean. Hence, agents' visibility would not add any extra power for the agents for these strategies.

Open Problems

Although we have developed some cleaning strategies for cleaning the butterfly, there are still many open problems.

What is the minimal number of agents that suffice to clean the butterfly? What is its corresponding cleaning strategy?

Is there any better strategy for the node search instead of using the same strategy for the edge search?

Chapter 4

Chordal Ring

Chordal ring topologies have frequently been used to implement local area networks as well as some highly coupled parallel computer systems because they possess a very high degree of fault tolerance.

In this chapter, we are going to study the cleaning problem by mobile agents in the chordal ring topology. Two variants of this problem (edge search and node search) are considered and the cleaning strategies are developed respectively. In both the edge search and node search, the asynchronous and synchronous models are studied. In the synchronous edge search, we only study the cleaning strategy in a special chordal ring: the chorded ring.

4.1 Definitions and Terminology

A circulant graph with n nodes and link structure $\langle d_1, d_2, ..., d_k \rangle$, where $1 \leq d_i \leq d_{i+1} \leq n-1$, is a graph with n nodes denoted as $x_0, x_1, ..., x_{n-1}$ where each node x_i $(0 \leq i \leq n-1)$ is adjacent to all the nodes $x_{i \pm d_j}$ for $1 \leq j \leq k$. Here $i \pm d_j$ means $(i \pm d_j)$ mod n. In the following discussion, we assume *all nodes' index operations are modulo* n. Every node of a circulant graph has degree $2k$.

A *chordal ring* is any circulant graph with $d_1 = 1$. In other words, a chordal ring is an augmented ring. A *chorded ring*, denoted by $C\langle p, k \rangle$ where $p < k$, is a chordal ring with link connection $\langle 1, 2, ..., p, k \rangle$. Every node of the chorded ring $C\langle p, k \rangle$ has degree $2p+2$. For an

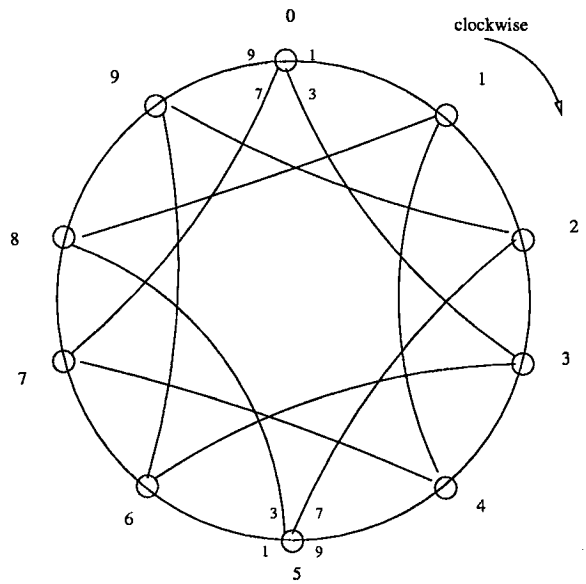


Figure 4.1: A chordal ring with 10 nodes and link structure $< 1, 3 >$; partial link labels.

example of a chordal ring, see Figure 4.1.

The links are labeled with chordal sense of direction, i.e., the label of a link is the distance between the two nodes in clockwise direction. Let x_i be a node of the chordal ring and x_j connected to x_i via a link. $\lambda_{x_i}(x_i, x_j) = d(x_i, x_j)$ where $d(x_i, x_j)$ is $(j - i)$ modulo n, the distance between node x_i and x_j. For the labels of links, see Figure 4.1.

We call the set of neighbors $\{x_{i+1}, x_{i+d_2}, ..., x_{i+d_k}\}$ of a node x_i the *clockwise neighbors* of x_i, and all the others the *counterclockwise neighbors*. For example, in Figure 4.1, node $\{2, 4\}$ are the clockwise neighbors of node 1; node $\{0, 8\}$ are the counterclockwise neighbors.

4.2 Edge Search

In this section, we will consider both synchronous and asynchronous edge search. In the asynchronous model, we will consider the cleaning in a chordal ring. However in the synchronous model, we will consider a special chordal ring: the chorded ring $C\langle p, k\rangle$.

In both strategies a team of identical and autonomous agents are deployed on the homebase to clean the network. Since the chordal ring is regular and vertex symmetric, to avoid the

recontamination, we have to deploy some agents to separate the clean portion of the ring from the contaminated portion. This becomes the key point of our strategies.

For simplicity of description, in both the edge search and node search, we consider the node where the agents are initially located node x_0, which is also called the *homebase*. We refer to the nodes of the ring in the clockwise order as x_0, x_1, x_2,..., x_{n-1}.

4.2.1 Asynchronous Edge Search

In this model, we consider a chordal ring with n nodes and link structure $\langle 1, d_2, ..., d_k \rangle$, $1 \leq d_i \leq d_{i+1} \leq n-1$. In the following, we first develop a cleaning strategy; then we show that our strategy is correct; i.e, that the circulant graph is clean; we also analyze the complexity of the strategy.

Cleaning strategy

The main idea is to deploy enough agents on the ring to avoid the recontamination of the clean links from the contaminated ones. To do that the agents completely cover a "window" of the chordal ring (i.e., a portion of the chordal ring of d_k consecutive nodes) and then the cleaning of the entire network starts. The strategy has two stages: the deployment stage and the cleaning stage. At the end of the deployment stage, node x_0 to x_{2d_k-1} are guarded by one agent except node x_{d_k}, which is guarded by two agents. In the cleaning stage, agents still guard node x_0 to x_{d_k-1} forming a "window" of consecutive d_k agents until they are notified to terminate; during the cleaning, another "window" of agents of size d_k moves forwards to guard the clean links.

Initially, $2d_k + 1$ agents are placed on the homebase x_0.

1. Deployment Stage

The group of agents start from x_0 and move in the clockwise direction to x_{2d_k-1} on the external ring (i.e., via link 1 of each visited node). On the way, one agent is left to guard every visited node. When two agents arrive at x_{2d_k-1}, one agent is left to guard the node, the other goes back to x_{d_k}. (Notice that this can be done in two hops, first going from x_{2d_k-1}

to x_{d_k-1} then from x_{d_k-1} to x_{d_k}.) Now all the nodes $x_0, x_1, ..., x_{d_k-1}, x_{d_k+1}, ..., x_{2d_k-1}$ are each guarded by one agent and x_{d_k} is guarded by two agents.

2. Cleaning Stage

The cleaning starts from x_{d_k} and proceeds in the clockwise direction until x_{n-1} is reached. Let x_i $(k \leq i \leq n-1)$ be the node with two agents on it.

2.1 While one of the two agents is guarding x_i, the other agent cleans all the links incident on x_i in an arbitrary order keeping the link leading to x_{i+d_k} as the last link to clean. After the agent cleans a link, it comes back to x_i and starts cleaning the next link; when the agent arrives at x_{i+d_k}, it stops.

2.2 When the links incident on x_i are clean, the agent guarding x_i moves to x_{i+1}; if this agent arrives at x_0, it notifies all the agents on x_0, x_1,..., x_{d_k-1} to terminate.

2.3 The agent on any node x_j where $i \neq j$ has to wait on x_j for another agent to arrive.

Please see Figure 4.2 for an execution of the cleaning strategy on the chordal ring with 10 nodes and link structure $C\langle 1, 3\rangle$.

Correctness and Analysis

Correctness. We now prove that our cleaning strategy is correct; i.e., that all links will be cleaned and that once an link has been cleaned, it will never be recontaminated.

Lemma 26. *A clean link will not be recontaminated in our strategy.*

PROOF In the deployment stage, link 1 of node x_i for $0 \leq i \leq d_{k-1}$ are cleaned. Since one agent is guarding x_i during the entire strategy, these clean links will not be recontaminated. We now show that a clean link in the cleaning strategy will not be recontaminated. We prove it by showing that all clean links of x_j for $d_k \leq j \leq n-1$ can not be recontaminated. By Induction. $j = d_k$. The cleaning starts from x_{d_k}. By the strategy, while one agent is guarding x_{d_k}, the other agent is cleaning all the contaminated links of x_{d_k}. By the definition of link structure, all neighbors of x_{d_k} are within x_0 to x_{2d_k-1}, except one neighbor, which is x_{2d_k}. By the strategy,

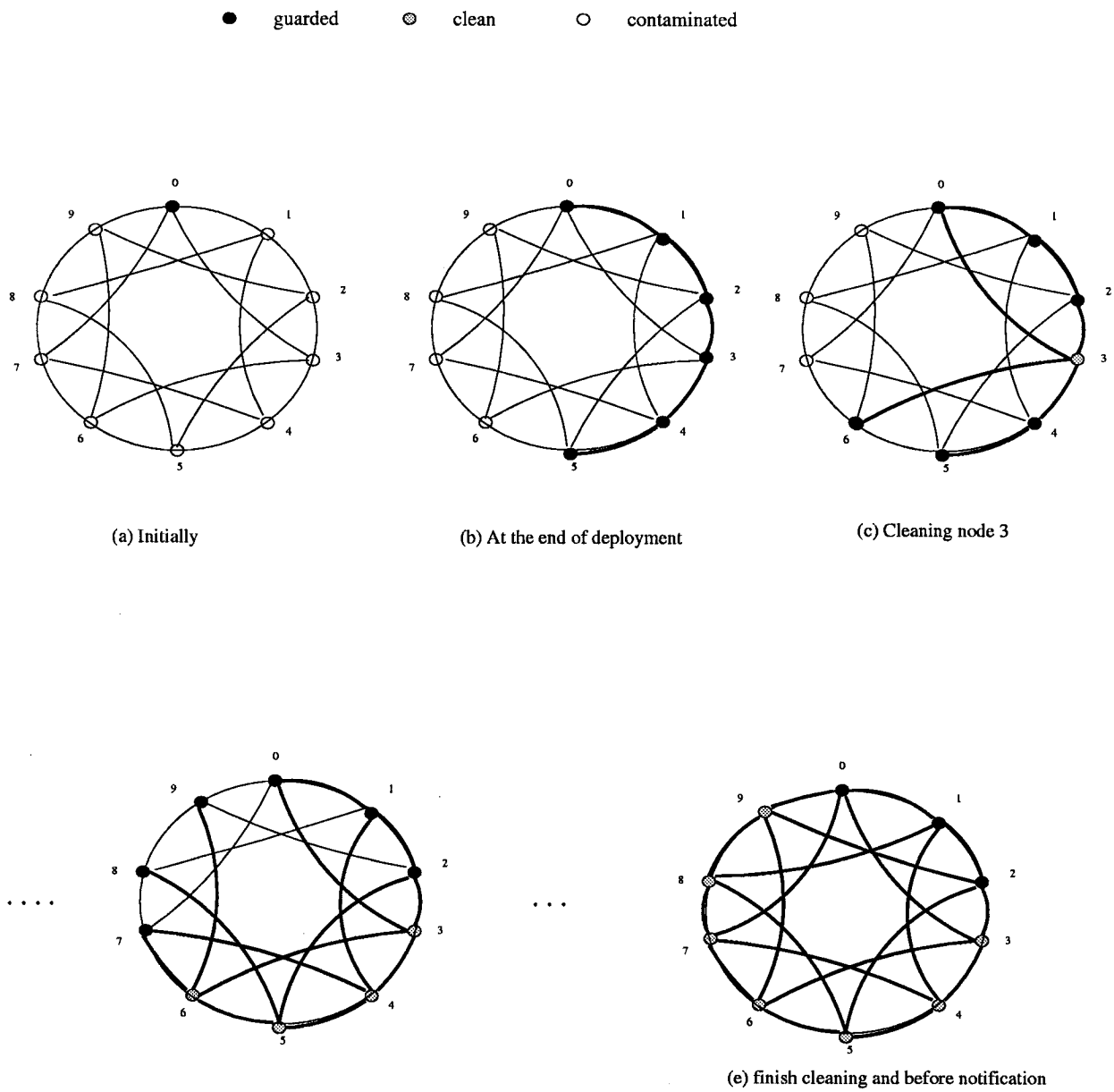


Figure 4.2: An execution on the chordal ring with 10 nodes and link structure $< 1, 3 >$.

after cleaning all other contaminated links, the cleaning agent cleans the last link to x_{2d_k} and stops there. Only after all links of x_{d_k} are clean, the guarding agent on it moves to x_{d_k+1}. Now no agent is on x_{d_k}, however all its links are clean and all neighbors are guarded by one agent, the clean links of x_{d_k} can not be recontaminated.

We assume it is true for x_{d_k}, ..., x_{m-1} ($d_k \leq m-1 < n-1$) and we show it is true for x_m.

By the induction hypothesis, we know that all links of x_{d_k}, ..., x_{m-1} are clean, these nodes are clean and no agents are on them; node x_m is guarded by two agents and x_{m+1} to x_{m-1+d_k} are guarded by one agent. We also know from the strategy that x_0, x_1, ..., x_{d_k-1} are still guarded by one agent during the cleaning.

Now we consider the cleaning at x_m. By the definition of link structure, all neighbors of x_m are within x_{m-d_k} to x_{m+d_k}. Since $m > d_k$, by induction hypothesis, except neighbor x_{m+d_k}, which is not guarded by an agent yet, all counterclockwise neighbors of x_m are either guarded by one agent or clean and the clockwise neighbors are guarded by one agent. By the cleaning strategy, while one agent is guarding x_m, the other agent cleans all the contaminated links and after cleaning the last link to x_{m+d_k}, it stops. Only after all links of x_m are clean, the guarding agent on it moves to x_{m+1}, which now has two agents on it. Now neighbor x_{m+d_k} is guarded by one agent too. Since all neighbors of x_m are either guarded or clean, all clean links of x_m can not be recontaminated after the guarding agent leaves for x_{m+1}. ∎

We now prove the correctness of the strategy.

Theorem 38. *The strategy cleans the chordal ring.*

PROOF First of all, in the deployment stage, link 1 of node x_i for $0 \leq i \leq d_{k-1}$ is cleaned; in the cleaning stage all links of node x_{d_k} to x_{n-1} are cleaned. Notice that all links (except link 1) of node x_0 to x_{d_k-1} are also links of some nodes from x_{d_k} to x_{n-1}; so these links are cleaned during the cleaning stage. The strategy cleans all the links of the chordal ring. By lemma 26, a clean link will not be recontaminated. Hence, the strategy cleans the chordal ring. ∎

Complexity. We first calculate the number of moves performed by the agents.

Theorem 39. *The total number of moves performed by the agents is* $2kn + 2d_k^2 - 2d_k + 2$.

PROOF In the deployment stage, a group of agents are moving towards x_{2d_k-1}. So the number of moves in this stage is $\sum_{i=1}^{2d_k-1} i + (2d_k - 1 + 2) = d_k(1 + 2d_k) + 1$ where $2d_k - 1 + 2$ is the number of moves by the agent back to x_{d_k}.

In the cleaning stage, while cleaning the nodes from x_{d_k} to x_{2d_k-1}, by the strategy, all contaminated links of each node are cleaned. In other words, all clockwise links of each node from x_0 to x_{2d_k-1} are traversed. Since each node has k clockwise links, the number of moves performed by the agents in the cleaning stage is $2kn - 2(2d_k - 1)$ where $(2d_k - 1)$ is the number of the links cleaned in the deployment stage. In addition, it takes $d_k - 1$ moves for the agent to notify all agents on node x_0 to x_{d_k-1} to terminate. Hence the total number of moves is $d_k(1 + 2d_k) + 1 + 2kn - 2(2d_k - 1) + d_k - 1 = 2kn + 2d_k^2 - 2d_k + 2$. ∎

We now consider the ideal time complexity of the cleaning strategy. We remind that the ideal time complexity is computed by assuming that it takes one unit of time for an agent to traverse an edge. The computation starts at 0.

Theorem 40. *The cleaning strategy takes* $2kn - d_k + 2$ *time units.*

PROOF It takes $2d_k + 1$ time units for the deployment stage. The cleaning process in the cleaning stage is carried out sequentially by the cleaning agent on each node. The time required is then equal to the number of moves of this stage, which is $2kn - 2(2d_k - 1) + d_k - 1$. So, totally it takes $2d_k + 1 + 2kn - 2(2d_k - 1) + d_k - 1 = 2kn - d_k + 2$. ∎

It directly follows from the strategy that

Theorem 41. *Our strategy employs* $2d_k + 1$ *agents.*

4.2.2 Synchronous Edge Search

In this section, we study the cleaning strategies for synchronous edge search in a special chordal ring: the chorded ring $C\langle p, k\rangle$.

The strategy of the previous section can be used for the synchronous edge search. However, it is inherently sequential and its execution in a synchronous environment would result in a very

high time complexity. In fact, since only one agent is cleaning at a time and it takes the cleaning agent one time unit to clean an edge and another unit to come back, the time required for the entire process will be at least twice the total number of the contaminated edges. To speed up the cleaning process in a synchronous environment, a different cleaning strategy can be developed by using more agents. We first devise a strategy for cleaning the chorded ring $C\langle p, k\rangle$ and then prove our strategy is correct and analyze the complexity.

Chorded Ring

We consider a chorded ring of size n with a simple link structure, $\langle 1, 2, ..., p, k\rangle$.

Cleaning Strategy

The idea of the strategy is to clean all clockwise links of node x_i at time i to achieve the n time complexity. The strategy works as follow:

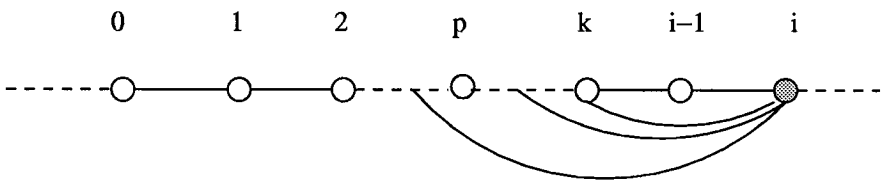
> Initially enough agents are placed at x_0. At time i, all clockwise incident links on node x_i are cleaned by one agent per link simultaneously; if $0 \leq i \leq k-1$, one agent is left to guard x_i and the rest if any move to x_{i+1}. When an agent moves from x_{n-1} to x_0, it notifies all the agents on node x_0 to x_{k-1} to terminate.

Correctness and Complexity

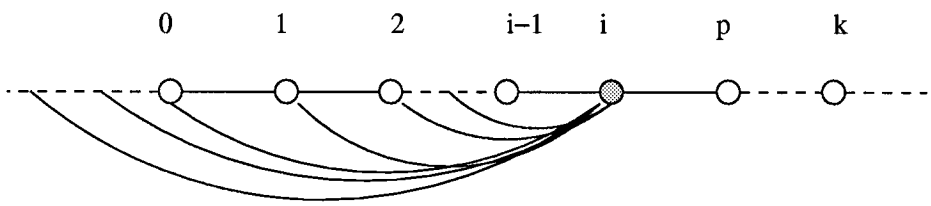
We first calculate the number of agents required by our strategy.

Theorem 42. $2k + \frac{p^2+p}{2}$ *agents suffice to clean the chorded ring by our strategy.*

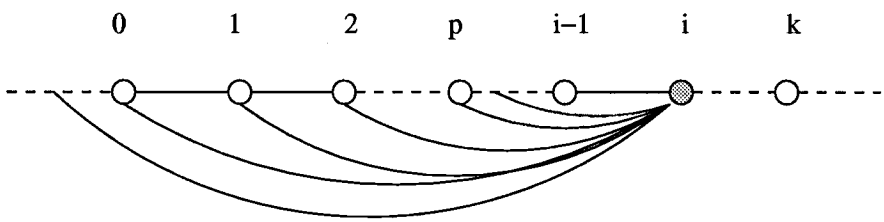
PROOF By the strategy, one agent is left to guard x_0 to x_{k-1}. Obviously the number of guarding agents is k. For each node x_j where $k \leq j \leq n-1$, (see Figure 4.3 (a)) because all counterclockwise neighbors of x_j are within x_0 to x_{j-1}, $p+1$ agents arrive at it via the $p+1$ counterclockwise links before time j, which is the time for the agents on it to clean the clockwise links of x_j. No extra agents are required. However, for a node x_i where $0 \leq i < k$, since only some of the counterclockwise neighbors of x_i are within x_0 to x_{i-1}, less than $p+1$ agents arrive via the counterclockwise links before time i. To clean the $p+1$ clockwise links, extra agents from the homebase are required. There are two cases.



(a) p+1 counterclockwise neighbors of node i are within node 0 and i–1.



(b) i (< p + 1) counterclockwise neighbors of node i are within node 0 to i–1



(c) p counterclockwise neighbors of node i are within node 0 to i–1

Figure 4.3: Node *i*'s counterclockwise neighbors.

Case 1: $0 \leq i \leq p$. (see Figure 4.3 (b)) Since i neighbors of x_i are within x_0 to x_{i-1}, which are cleaned before time i, i agents arrive at x_i. In addition, $p+1-i$ agents are required from the homebase. By the strategy, these extra agents have arrived at x_i at time i. So, the nodes from x_0 to x_p require $\sum_{i=0}^{p}(p+1-i)$ agents.

Case 2: $p < i < k$. (see Figure 4.3 (c)) Node x_i has p neighbors within x_0 to x_{i-1}, p agents arrive at x_i before time i. Only one extra agents is required in order to clean the $p+1$ clockwise links of x_i. So, the nodes from x_{p+1} to x_{k-1} require $\sum_{i=p+1}^{k-1} 1$.

Thus, the total number of agents required by the strategy is :

$\sum_{i=0}^{p}(p+1-i) + \sum_{i=p+1}^{k-1} 1 + k = \sum_{i=0}^{p}(p+1)$ - $\sum_{i=0}^{p} i + (k-p-1) + k$

$= (p+1)^2 - \frac{p(p+1)}{2} + 2k - p - 1 = p^2 + p - \frac{p(p+1)}{2} + 2k$

$= \frac{p^2+p}{2} + 2k$ ∎

We now calculate the number of moves performed by the agents.

Theorem 43. *The number of moves performed by the agents is* $(p+1)n + k^2 + \frac{p^3-p}{6} - 1$.

PROOF By the strategy, every link is traversed by at least one agent, so $(p+1)n$ moves are performed by the agents for the entire process. The number of moves performed by the k guarding agents is $\sum_{i=0}^{k-1} i = \frac{k(k-1)}{2}$. In addition, link 1 of node x_i for all $0 \leq i \leq k-1$ is traversed by more than one agent. We now consider the moves performed by the extra agents moving from the homebase to the needed node x_i. From the proof of theorem 42, we know that $(p+1-i)$ extra agents are needed if $0 \leq i \leq p$ and one extra agent is needed if $p < i < k$. So, the number of moves performed by the extra agent(s) for x_i is $i(p+1-i)$ or i; hence, the number of moves performed by all extra agents for node x_0 to x_{k-1} is $\sum_{i=0}^{p} i(p+1-i) + \sum_{i=p+1}^{k-1} i$. It takes $k-1$ moves to notify the agents, which are guarding node x_0 to x_{k-1}, to terminate. Totally, the number of moves performed by the agents is $(p+1)n + k - 1 + \frac{k(k-1)}{2} + \sum_{i=0}^{p} i(p+1-i) + \sum_{i=p+1}^{k-1} i$

$= (p+1)n + k - 1 + \frac{k(k-1)}{2} + (p+1)\sum_{i=1}^{p} i - \sum_{i=1}^{p} i^2 + \sum_{i=p+1}^{k-1} i = (p+1)n + k^2 + \frac{p^3-p}{6} - 1$. ∎

It directly follows from the strategy that

Theorem 44. *The strategy takes n time units to clean the $C\langle p, k\rangle$ chorded ring.*

We now prove that our cleaning strategy is correct; i.e., that all links will be cleaned and that once a link has been cleaned, it will never be recontaminated.

Theorem 45. *Our strategy cleans all the links of the $C\langle p, k\rangle$ chorded ring and a clean link will never be recontaminated.*

PROOF First of all, by the strategy, at time i, all the clockwise links incident on x_i are cleaned. So, at time $n-1$, all links of the chorded ring are cleaned. Now we prove that a clean link will not be recontaminated. We prove it by showing all clockwise links incident on any node x_i will not be recontaminated. By induction.

At time 0, by the strategy, each clockwise link of x_0 is cleaned by one agent and then the other endpoint of the link is guarded by the arrival agent. Since one agent is left to guard x_0 by the strategy, although all counterclockwise links of x_0 are contaminated, no recontamination can occur to the clean links of x_0.

Assume the clockwise links incident on node x_0 to x_{i-1} where $i-1 \geq 0$ are clean at time $i-1$. We show that at time i the clockwise links incident on node x_i are clean and will not be recontaminated.

Depending on where x_i is, there are three cases.

Case 1: $0 < i \leq k-1$. By the strategy, at time i each clockwise link of x_i is cleaned by one agent and then the other endpoint of the link is guarded by the arrival agent. Since one agent is left to guard x_i by the strategy, although some counterclockwise link(s) incident on x_i are contaminated, no recontamination can occur to the clean links of x_i. By induction hypothesis, all clockwise links incident on node x_0 to x_{i-1} are clean. At time i, nothing has changed to the clean links incident on node x_0 to x_{i-1} (The guarding agents are still guarding). Hence, no recontamination can occur.

Case 2: $k = i$. By the strategy, at time k each clockwise link of x_k is cleaned by one agent and then the other endpoint of the link is guarded by the arrival agent; no agent is left to guard x_k. By induction hypothesis, all clockwise links incident on node x_0 to x_{k-1} are

clean; and by the strategy, node x_0 to x_{k-1} are guarded by one agent. So at time k, all the links incident on x_k are clean and all neighbors of x_k are guarded by one agent. Even though no agent is left to guard x_k, no recontamination can occur to the clean links of x_k as well as to other clean links. Node x_k becomes clean.

Case 3: $k < i \leq n-1$. By the strategy, at time i each clockwise link of x_i is cleaned by one agent and then the other endpoint of the link is guarded by the arrival agent; no agent is left to guard x_i. By induction hypothesis, all clockwise links incident on node x_0 to x_{i-1} are clean. Since $i \geq k$, we know all counterclockwise links of x_i are clean before time i, and its clockwise links become clean at time i. So all links of x_i are clean. Since the clockwise neighbors of x_i are guarded by one agent and the counterclockwise neighbors are either guarded or clean, no recontamination can occur to the clean links of x_i as well as other clean links. Node x_i becomes clean.

Hence, no recontamination can occur to any clean link of x_i. ∎

4.3 Node Search

In this section, we are going to discuss briefly both the asynchronous node search and the synchronous node search in a chordal ring with n nodes and link structure $\langle 1, d_2, ..., d_k \rangle$, $1 \leq d_i \leq d_{i+1} \leq n-1$.

4.3.1 Asynchronous Node Search

Our goal is to clean all the nodes of the chordal ring. However, since each node of the chordal ring has more than two links incident on it, our strategy has to make sure that the intruder cannot recontaminate a clean node via any link. The agents we employ are autonomous and identical.

Cleaning strategy

With minor modification, the cleaning strategy developed for the asynchronous edge search can be used for the asynchronous node search. The deployment stages are the same in both

strategies. The only difference is in the cleaning stage. More precisely, the cleaning agent goes directly to the only contaminated node instead cleaning all the contaminated links; the guarding agent goes with the cleaning agent to the contaminated node and then comes back. In the node search,

2. Cleaning Stage

2.1 The cleaning starts at x_{d_k} and proceeds in the clockwise direction until x_{n-1-d_k} is reached. Let x_i $(k \leq i \leq n-1-d_k)$ be the node with two agents on it. These two agents move along link d_k of x_i; and only when the two agents arrive at node x_{i+d_k}, one is left to guard x_{i+d_k} and the other goes back to x_i; when the agent arrives at x_i, it then moves along link 1 of x_i to arrive at node x_{i+1}. The agent on any node x_j where $i \neq j$ has to wait on x_j for another agent to arrive.

2.2 When two agents are on node x_{n-d_k}, one agent terminates and the other agent goes to notify all the agents on node x_{n-d_k+1} to x_{d_k-1} to terminate.

Notice that in the asynchronous system, it is necessary for the two agents from node x_i to go to node x_{i+d_k} together. The agent, which goes back to x_i later, makes sure that the other agent is guarding node x_{i+d_k} and only then cleaning node x_{i+1} can start.

Correctness and Complexity

Correctness We first prove that our cleaning strategy is correct; i.e., that all nodes will be cleaned and that once a node has been cleaned, it will never be recontaminated.

Theorem 46. *Our strategy cleans all the nodes of the chordal ring and a clean node will never be recontaminated.*

PROOF At the end of the deployment stage, node x_0 to x_{2d_k-1} are guarded by one agent. By the strategy, in the cleaning stage, node x_0 to x_{d_k-1} are still guarded by an agent while node x_{d_k} to x_{n-1-d_k} are cleaned; now node x_{n-d_k} to x_{n-1} are guarded by one agent too. So after the agent notifies all the agents on node x_{n-d_k} to x_{d_k-1} to terminate, all the nodes of the chordal

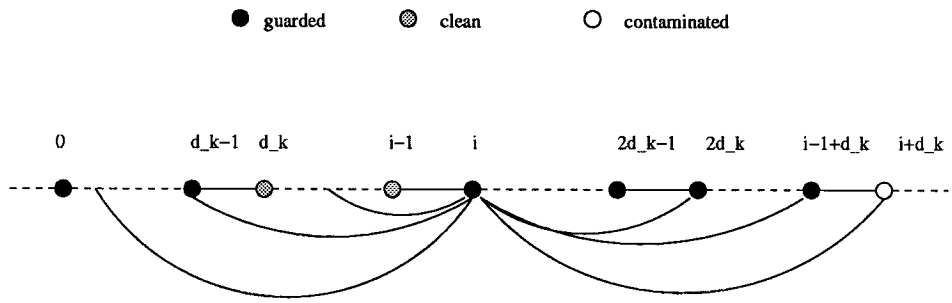


Figure 4.4: States of node i's neighbors.

ring are cleaned by our strategy. We now prove that a clean node will not be recontaminated. By induction.

The cleaning starts at node x_{d_k}. Except the neighbor x_{2d_k} of x_{d_k}, which is not guarded by an agent yet, all the other neighbors are guarded. By the strategy, the two agents on it move to x_{2d_k}, the only contaminated neighbor. Then one agent is left to guard this neighbor and the other goes back to x_{d_k}. So all neighbors of x_{d_k} are guarded; when the agent arrive at x_{d_k} and then move to x_{d_k+1}, node x_{d_k} becomes clean and no recontamination can occur.

Assume node x_{d_k} to x_{i-1} where $d_k \leq i-1 \leq n-2-d_k$ are clean and the cleaning is at node x_i. We show that x_i becomes clean and no recontamination can occur. Depending on the location of x_i, there are two cases.

Case 1: $d_k < i \leq 2d_k - 1$. By the strategy, node x_0 to x_{d_k-1} are guarded. By induction hypothesis, node x_{d_k} to x_{i-1} are clean. So the counterclockwise neighbors of x_i are either guarded or clean. From the induction hypothesis together with the strategy, we know node x_{2d_k} to x_{i-1+d_k} are guarded. Since $d_k < i \leq 2d_k - 1$, node x_{i+1} to x_{2d_k-1} are guarded too. So the clockwise neighbors are guarded except x_{i+d_k}. (see Figure 4.4) By the strategy, when the two agents move to the only contaminated neighbor x_{i+d_k}, no recontamination can occur. One agent is left to guard x_{i+d_k}. So all neighbor of x_i are either guarded or clean; when the agent goes back to x_i and then moves to x_{i+1}, no agent is left to guard x_i and x_i becomes clean. No recontamination can occur to x_i.

Case 2: $2d_k \leq i \leq n-2-d_k$. By induction hypothesis, node x_{d_k} to x_{i-1} are clean. So the counterclockwise neighbors of x_i are clean. From the induction hypothesis together

with the strategy, we know node x_{2d_k} to x_{i-1+d_k} are ever guarded. But Since the cleaning is at node x_i now and $2d_k \leq i \leq n-2$, only node x_{i+1} to x_{i-1+d_k} are guarded. So the clockwise neighbors are guarded except x_{i+d_k}. By the strategy, when the two agents move to the only contaminated neighbor x_{i+d_k}, no recontamination can occur. One agent is left to guard x_{i+d_k}. So all neighbor of x_i are either guarded or clean; when the agent goes back to x_i and then moves to x_{i+1}, no agent is left to guard x_i and x_i becomes clean. No recontamination can occur to x_i.

At step 2.2, when the agent from x_{n-1-d_k} arrives at x_{n-d_k}, all the nodes from x_{n-d_k} to x_{d_k-1} are guarded and the others are clean from step 2.1. There is no contaminated node anymore. So it is impossible to recontaminate a clean node. ∎

Complexity

We first calculate the number of moves performed by the agents.

Theorem 47. *The total number of moves performed by the agents is* $4n + 2d_k^2 - 5d_k$.

PROOF In the deployment stage, a group of agents are moving towards x_{2d_k-1}. So the number of moves in this stage is $\sum_{i=1}^{2d_k-1} i + (2d_k - 1 + 2) = d_k(1 + 2d_k) + 1$ where $2d_k - 1 + 2$ is the number of moves by the agent back to x_{d_k}.

In the cleaning stage, for cleaning one node x_i, four moves are performed by the agents. It takes one move for each of the two agents to arrive at x_{i+d_k}, one move back to x_i and then one move to x_{i+1}. So totally, $4(n - 2d_k)$ moves are performed in the cleaning stage. It takes $2d_k - 1$ moves for the agent from x_{n-d_k} to notify agents on x_{n-d_k} to x_{d_k-1} to terminate.

So totally, the number of moves for the entire process is $d_k(1+2d_k) + 1 + 4(n-2d_k) + 2d_k - 1$ $= 4n + 2d_k^2 - 5d_k$. ∎

We now consider the ideal time complexity of the cleaning strategy.

Theorem 48. *The cleaning strategy takes* $4n - 4d_k$ *time units.*

PROOF The deployment stage takes $2d_k + 1$ time units. The cleaning process is carried out sequentially by the cleaning agent on each node. The time required is then equal to the number

of moves of this stage, which is $4(n-2d_k)+2d_k-1$. So totally, it takes $2d_k+1+4(n-2d_k)+2d_k-1=4n-4d_k$. ∎

It directly follows from the strategy that

Theorem 49. *Our strategy employs* $2d_k+1$ *agents.*

4.3.2 Synchronous Node Search

Cleaning strategy

The idea is similar to the asynchronous node search. Agents are deployed on node x_0 and then in the deployment stage, agents are scattered from x_0 to x_{2d_k-1} to guard the future clean nodes from recontamination. The cleaning starts from node x_{d_k} and proceeds in the clockwise direction until it reaches node x_{n-1}. Since they are synchronous, the two agents on a node x_i can go along link 1 and link d_k independently at the same time.

Initially, $2d_k+1$ agents are placed on the homebase x_0.

1. Deployment Stage

At time i (where $0 \leq i \leq 2d_k-1$) one agent is left to guard node x_i and the rest move along link 1 of x_i to node x_{i+1}. However, at time d_k, two agents instead of one are left to guard node x_{d_k}.

2. Cleaning Stage

The cleaning starts at x_{d_k} and proceeds in the clockwise direction until x_{n-1}.

2.1 Let x_i $(d_k \leq i \leq n-1)$ be the node with two agents on it. At time $i+d_k$, the two agents on x_i move along link 1 and link d_k to arrive at node x_{i+1} and x_{i+d_k} independently. If the agent arrives at x_{i+d_k} and finds it is already guarded by an agent, these two agents on x_{i+d_k} terminate; if the agent arrives at x_{i+1} and finds it is x_0 which is clean, it terminates.

2.2 The agent on any node x_j where $i \neq j$ has to wait on x_j for another agent to arrive.

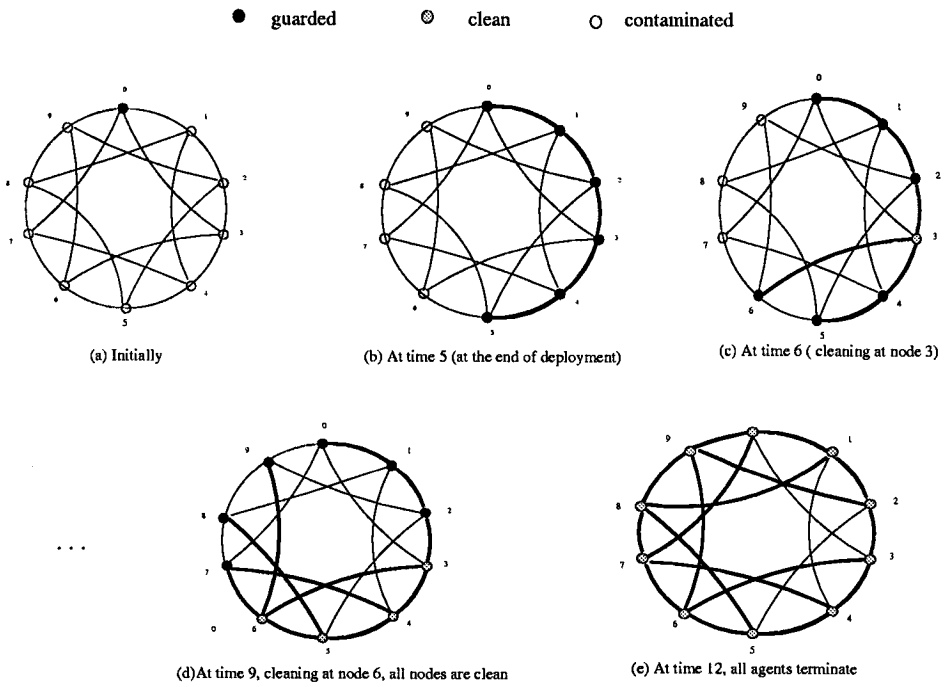


Figure 4.5: An execution on the chordal ring with 10 nodes and link structure $< 1, 3 >$.

Please see Figure 4.5 for an execution of the cleaning strategy on the chordal ring with 10 nodes and link structure $< 1, 3 >$.

Correctness and Complexity

Correctness We first prove that our cleaning strategy is correct; i.e., that all nodes will be cleaned and that once a node has been cleaned, it will never be recontaminated.

Theorem 50. *Our strategy cleans all the nodes of the chordal ring and a clean node will never be recontaminated.*

PROOF At time $2d_k - 1$, the end of the deployment stage, node x_0 to x_{2d_k-1} are guarded by one agent, except node x_{d_k} is guarded by two agents. All these nodes are cleaned. By the strategy, in the cleaning stage, node x_{2d_k} to x_{n-1} are cleaned. So, all the nodes of the chordal ring are cleaned by our strategy. Now we prove that a clean node will not be recontaminated. By induction.

The cleaning starts at time $2d_k$ from node x_{d_k}. At time $2d_k$, except the neighbor x_{2d_k} of x_{d_k} is contaminated or not guarded by an agent yet, all the other neighbors are guarded. By the strategy, at time $2d_k$ one of the two agents moves via link d_k to clean the only contaminated neighbor and then guards it; and the other agent moves via link 1 to node x_{d_k+1}. Now all neighbors of x_{d_k} are guarded and no agent is on it. Node x_{d_k} becomes clean and no recontamination can occur to it.

Assume from time $2d_k$ to $d_k + i - 1$ where $d_k \leq i - 1 \leq n - 2$, node x_{d_k} to x_{i-1} are cleaned. At time $d_k + i$, the cleaning is at node x_i. We show that at node x_i becomes clean and no recontamination can occur.

Depending where x_i is, there are two cases.

Case 1: $d_k < i \leq 2d_k - 1$. By the strategy, node x_0 to x_{d_k-1} are guarded. By induction hypothesis, node x_{d_k} to x_{i-1} are clean. So the counterclockwise neighbors of x_i are either guarded or clean. From the induction hypothesis together with the strategy, we know node x_{2d_k} to x_{i-1+d_k} are guarded. Since $d_k < i \leq 2d_k - 1$, node x_{i+1} to x_{2d_k-1} are still guarded from the deployment stage. So the clockwise neighbors of x_i except x_{i+d_k} are guarded. By the strategy, at time $d_k + i$, one of the two agents moves to clean the only contaminated neighbor x_{i+d_k} and then guards it; the other moves to node x_{i+1}. So all neighbor of x_i are either guarded or clean; no agent is left to guard it. At time $d_k + i$, node x_i becomes clean. Obviously, no recontamination can occur to it.

Case 2: $2d_k \leq i \leq n - 2$. By induction hypothesis, node x_{d_k} to x_{i-1} are clean. So the counterclockwise neighbors of x_i are clean. From the induction hypothesis together with the strategy, we know node x_{2d_k} to x_{i-1+d_k} are ever guarded. But Since the cleaning is at node x_i now, only node x_{i+1} to x_{i-1+d_k} are guarded. So the clockwise neighbors of x_i are guarded except x_{i+d_k}. By the strategy, at time $i + d_k$, one of the two agents moves to clean the only contaminated neighbor x_{i+d_k} and then guards it; the other agent moves to node x_{i+1}. Now all neighbor of x_i are either guarded or clean; no agent is left on x_i. Node x_i becomes clean at time $i + d_k$. No recontamination can occur to it.

At time $n - 1$, one of the two agents from x_{n-1-d_k} moves via link d_k to clean the contaminated

node x_{n-1}. All the nodes of the chordal ring are guarded or clean. No recontamination is possible anymore. ∎

Complexity

We first calculate the number of moves performed by the agents.

Theorem 51. *The total number of moves performed by the agents is* $2n + 2d_k^2 - 2d_k$.

PROOF In the deployment stage, a group of agents are moving towards x_{2d_k-1}. So the number of moves in this stage is $\sum_{i=1}^{2d_k-1} i + d_k = 2d_k^2$ where d_k is the number of moves performed by the extra agent at x_{d_k}.

In the cleaning stage, it takes 2 moves by the agents to clean one node. So totally, $2(n - d_k)$ moves are performed by the agents in the cleaning stage. By step 2.1, while cleaning from node x_{n-d_k} to x_{n-1}, the agents on node x_0 to x_{d_k} terminate. So there is no need to notify those agents to terminate.

Totally, the number of moves performed by the agents is $2d_k^2 + 2(n - d_k) = 2n + 2d_k^2 - 2d_k$. ∎

We now consider the time complexity of the cleaning strategy.

Theorem 52. *The cleaning strategy takes* $n + d_k - 1$ *time units.*

PROOF The deployment stage takes $2d_k - 1$ time units; and the cleaning stage starts at time $2d_k$ and ends at time $n - 1 + d_k$. ∎

It directly follows from the cleaning strategy that

Theorem 53. *The cleaning strategy employs* $2d_k + 1$ *agents.*

4.4 Summary

In this chapter, we have studied two variants (edge search and node search) of the cleaning problems in the chordal ring with n nodes and link structure $\langle 1, d_2, ..., d_k \rangle$. In each variant, we study both the asynchronous and synchronous models. In the Synchronous Edge Search, we study a special chordal ring: the chorded ring $C\langle p, k \rangle$. Different cleaning strategies in each

	Model	Agents	Moves	Time
Edge Search	Asynchronous	$2d_k + 1$	$2kn + 2d_k^2 - 2d_k + 2$	$2kn - d_k + 2$
	Synchronous $C\langle p, k\rangle$ [1]	$2k + \frac{p^2+p}{2}$	$(p+1)n + k^2 + \frac{p^3-p}{6} - 1$	n
Node Search	Asynchronous	$2d_k + 1$	$4n + 2d_k^2 - 5d_k$	$4n - 4d_k$
	Synchronous	$2d_k + 1$	$2n + 2d_k^2 - 2d_k$	$n + d_k - 1$

Table 4.1: Comparison of different models

variant are developed and analyzed in depth. We summarize the differences among the several models of the cleaning strategies in Table 4.1.

In the asynchronous edge search, the cleaning is inherent sequential, only one agent is cleaning all the links of the chordal ring while all the other agents are there to guard the nodes from recontamination. In the synchronous edge search, the strategy developed for the asynchronous edge search can be employed with same complexity; but to speed up the cleaning process, a different strategy is devised. In this strategy, it takes n time units for the entire cleaning. However, more agents have to be employed. In the asynchronous and synchronous node search, the strategies are similar to the one of asynchronous edge search but without cleaning every link.

Observation

All cleaning strategies developed in this chapter employ a *deployment stage*. Moving the agents from the homebase to node x_{2d_k-1} introduces extra moves. If we want to reduce this extra moves, we can add the cloning power to agents as we did for the hypercube. With this capability, initially one agent is placed on the homebase. In the deployment stage, the agent on a node x_i creates one new agent and then one is sent to node x_{i+1} and the other is left to guard x_i. Totally, the number of moves performed by the agents is greatly reduced to $2d_k - 1$.

Open Problems

We conjecture $2d_k + 1$ is the optimal number of agents to clean the chordal rings with link structure $\langle 1, d_2, ..., d_k\rangle$ in both edge search and node search. But we have not yet proved it.

[1] Notice that in this case the result is for a $C\langle p, k\rangle$ chorded ring.

Chapter 5

Conclusions

In the thesis, we have studied the intruder capture problem in Hypercube, Butterfly and Chordal Ring. In particular, we have compared strategies under different assumptions on the capabilities of the agents. Our goal was to better understand the impact that additional assumptions (synchronicity, cloning, visibility) have on the complexity of our solutions. Many interesting problems are still open.

- It is known that a linear time algorithm exists to compute the minimal contiguous search number only in trees [3]. Actually the only topology where a lower bound (and a matching upper bound) on the number of agents is known is the tree [3]. In this thesis, we have developed some strategies for cleaning the network of hypercube, butterfly and chordal ring, it is unknown what is the minimal number of agents required to clean these topologies. We conjecture $\Omega(\frac{n}{log\ n})$ is the lower bound on the number of agents required to clean the hypercube and $2d_k + 1$ for chordal rings. As an ongoing work we are trying to prove our conjecture.
- We have studied the power of cloning and visibility in the hypercube and given some observations about them in the butterfly and chordal ring. From the observations here it seems that the power of cloning and/or visibility does not help in reducing the number of agents required to clean the network. We leave as an open problem to prove cloning and/or visibility do not help or show examples where they really help.

- We studied the power of visibility only in the hypercube and gave some observations about it in the butterfly. Agents' visibility allows the strategy being totally local, meaning that in the asynchronous model the agents can make their decisions on the actions to take without the need of a coordinator. How can the power of visibility be exploited in developing strategies for chordal rings and other topologies?
- A cleaning strategy with a synchronizer for the asynchronous node search was presented in Chapter 2. The strategy requires $\binom{d-1}{\frac{d}{2}-2} + \binom{d}{\frac{d}{2}} + 1$ agents. By the strategy during the cleaning of level $l+1$, the $\binom{d-1}{l-1}$ agents on the leaves of level l do not participate in the cleaning of level $l+1$, while the other $\binom{d}{l+1}$ are just enough to move to level $l+1$ on the broadcast tree guided by the synchronizer. Can we make use of the agents on the leaves; i.e., when the agent on the leaf is set free, instead of going back to the root, it could move with the synchronizer to help in cleaning the next node, thus reducing the total number of agents required. If the agents on leaves is reused in the cleaning of the next level, how many agents are actually needed?
- The strategies for the edge search in the butterfly are employed also for the node search. Can we find a better strategy specific for the node search instead of using the same strategy for the edge search?
- We presented a cleaning strategy for the synchronous edge search in the chorded ring $C\langle p, k\rangle$. This strategy requires $2k + \frac{p^2+p}{2}$ agents and n time steps. Is there any strategy that requires only n time steps but employs less agents for this synchronous edge search in the chorded ring $C\langle p, k\rangle$?

Bibliography

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

[2] L. Barrière and P. Fraignaud and N. Santoro and D.M. Thilikos. Searching is not jumping. 29th Workshop on Graph Theoretic Concepts in Computer Science (WG), Elspeet, the Netherlands, Springer Verlag, LNCS **2880**, 34-45, 2003.

[3] L. Barrière and P. Flocchini and P. Fraignaud and N. Santoro. Capture of an intruder by mobile agents. *Proc. 14-th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Winnipeg, Manitoba, Canada, 2002.

[4] H.L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and Treewidth of Graphs. *Journal of Algorithms*, **21**, 358-402, 1996.

[5] H.L. Bodlaender and T. Kloks and D. Kratsch. Treewidth and pathwidth of permutation graphs. *Proc. of International Conference in Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, **700**, 114-125, 1993.

[6] H.L. Bodlaender and R.H. Moehring. The pathwidth and treewidth of cographs. *SIAM Journal of Discrete Mathematics*, **6** (2), 181-188, 1993.

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

[8] R. Chang. Single step graph search problem. *Information Processing Letters*, 40(2)107-111, 1991.

[9] N. Dendris, L. Kirousis, and D. Thilikos. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science*, 172(1-2):233-254, 1997.

[10] J.A. Ellis and I.H. Sudborough and J.S. Turner. The vertex separation and search number of a graph. *Information and Computation*, **113**, 50-79, 1994.

[11] M. Fellows and M.langston. On search, decision and the efficiency of polynomial time algorithm. In 21st ACM *Symp. on Theory of Computing* (STOC '89), pp. 501-512, 1989.

[12] F. Fomin and P. Golovach. Graph searching and interval completion. *SIAM Journal on Discrete mathematics* 13(4), 454-464, 2000.

[13] R.L. Graham and D.E. Knuth and O. Oren Patashnik. Concrete Mathematics, Second Edition, Reading, Massachusetts, Addison-Wesley, 1994.

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

[15] S. Hansen and M. Eldredge. Intruder isolation and monitoring. In 1st USENIX Security Workshop, pages 63-64, 1988.

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

[17] T. Kloks and H. Bodlaender and H. Muller and D. Kratsch. Computing treewidth and minimum fill-in: All you need are the minimal separators. *Proc. 1st Annual European Symposium on Algorithms (ESA)*, in T. Lengauer, editor, Springer Verlag, Lecture Notes on Computer Science, vol. 726, 260-271, Berlin, 1993.

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

[19] L. M. Kirousis and C. H. Papadimitriou. Interval graphs and searching. Discrete Math. 55 (1985), 181-184.

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

[21] 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.

[22] 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.

[23] F. Makedon and H. Sudborough. Minimizing width in linear layout. In 10th *Int. Colloquium on Automata, Languages, and Programming* (ICALP '83), LNCS 154, Springer-Verlag, 478-490, 1983.

[24] B. Monien and I.H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theoretical Computer Science*, **58**, 209-229, 1988.

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

[26] 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.

[27] S. Peng and M. Ko and C. Ho and T. Hsu and C. Tang. Graph searching on chordal graphs. *Algorithmica*, **27**, 395-426, 2000.

[28] 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.

[29] P.D. Seymour and R. Thomas, Graph searching, and a minmax theorem for tree width. *Journal of Combinatoral Theory*, Series B 58, 22-53, 1993.

[30] J. Smith. Minimal trees of given search number. *Discrete mathematics* 66(1987)191-202.

[31] Y. Stamatiou and D. Thilikos. Monotonicity and inert fugitive search games. In 6th *Twente Workshop on Graphs and Comb. Opt.*, Elsevier, 1999.

[32] A. Takahashi, S. Ueno, and Y. Kajitani. Mixed searching and proper path width. *Theoretical Computer Science*, 137(2):253-268, 1996.