Decontamination of Hypercubes by Mobile Agents*

Paola Flocchini† Miao Jun Huang† Flaminia L. Luccio‡

Abstract

In this paper we consider the decontamination problem in a hypercube network of size n. The nodes of the network are assumed to be contaminated and they have to be decontaminated by a sufficient number of agents. An agent is a mobile entity that asynchronously moves along the network links and decontaminates all the nodes it touches. A decontaminated node that is not occupied by an agent is re-contaminated if it has a contaminated neighbour.

We consider some variations of the model based on the capabilities of mobile agents: *locality*, where the agents can only access local information; *visibility*, where they can "see" the state of their neighbours; and *cloning*, where they can create copies of themselves. We also consider *synchronicity* as an alternative system requirement.

For each model, we design a decontamination strategy and we make several observations. For agents with locality, our strategy is based on the use of a coordinator that leads the other agents. Our strategy results in an optimal number of agents, $\Theta(\frac{n}{\sqrt{\log n}})$, and requires $O(n \log n)$ moves and $O(n \log n)$ time steps. For agents with visibility, we assume that the agents can move autonomously. In this setting, our decontamination strategy achieves an optimal time complexity ($\log n$ time steps), but the number of agents increases to $\frac{n}{2}$. Finally, we show that when the agents have the capability to clone combined with either visibility or synchronicity, we can reduce the move complexity—which becomes optimal—at the expense of an increase in the number of agents.

Keywords: Network Decontamination, Distributed Algorithms, Intruder Capture, Graph Search, Hypercube, Mobile Agents, Interconnection Networks.

Corresponding Author: Paola Flocchini, SITE, University of Ottawa, 800 King Edward, K1N 6N5, Ottawa, Canada. Phone: +1-613-562-5800 (6582). Fax: 1-613-562-5187.

*A preliminary version of this paper appeared in IPDPS 2005.

†School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, K1N 6N5, Canada. {flocchin,mhuang}@site.uottawa.ca. Partially supported by NSERC.

‡Dipartimento di Informatica, Università Ca' Foscari Venezia, Venezia, Italy. luccio@unive.it.

1 Introduction

1.1 The Problem

We consider the problem of deploying a team of mobile agents to decontaminate a network infected by a mobile virus. From a practical point of view, the disinfection of a network and the protection of its hosts from unwanted, possibly dangerous intrusions is a pressing security problem in networked environments. In fact, a great deal of research has been devoted to this issue, especially for detection (e.g., see [1, 17, 30]). The problem has been studied extensively from a graph-theoretical point of view under the terms *intruder capture*, *decontamination*, and *graph search* (e.g., see [2, 6, 10, 14, 19, 20, 22–24, 26]). In particular, research has looked at variations of the decontamination problem in which the agents are allowed to "jump" from one node to any other node. The focus has often been on determining the minimum number of agents able to perform decontamination because this number is closely related to standard graph parameters such as pathwidth and treewidth (e.g., see [10, 11, 26, 28, 29]).

In the *decontamination problem* without "jumping", defined in [2], a team of agents is initially located on a single node, the *homebase*, and the agents can move freely from the node they are on to a neighbouring node. At any point in time each node of the network can be in one of three possible states: *clean*, *contaminated*, or *guarded*. A node is guarded when it contains at least one agent, clean when an agent has been on the node and all the neighbouring nodes are clean or guarded, and contaminated otherwise. Initially all nodes are contaminated except for the homebase, which is obviously guarded. The solution to the problem is given by a cleaning strategy for the agents that guarantees that after a finite amount of time all the nodes are simultaneously clean. A strategy is called *monotone* if guarantees that after a node has been decontaminated it will not be re-contaminated. Decontamination has to be executed as efficiently as possible. Efficiency is measured in terms of the size of the team of agents, traffic (i.e., the number of moves the agents have to perform), and time (or steps) [*].

Starting with the model employed in [2] (here called the *local model*), we introduce additional assumptions in order to study the impact that more powerful agent capabilities or additional system requirements have on the solution process for our problem. In the *local model*, an agent located at a node can only "see"

[*] In the case of asynchronous systems, we measure ideal time, that is, we assume that it takes one unit of time for an agent to traverse a link.

local information, such as the state of the node, the labels of the incident links, and the other agents present on the node. *Visibility* is the capability of the agent to "see" the state of the neighbouring nodes, meaning that an agent can see whether a neighbouring node is guarded, clean, or contaminated. *Cloning* is the capability for an agent to create copies of itself. *Synchronicity* implies that local computations are instantaneous and it takes one unit of time (one step) for an agent to move from a node to a neighbouring one.

In this paper, our main focus is on understanding what impact the power of the agents and the underlying system has on the complexity of the problem. As a first step towards understanding these issues, we concentrate on a specific topology—the hypercube, which is a fairly common topology for interconnection networks and has been the subject of extensive investigation (e.g., see [21]).

We design strategies to solve the decontamination problem both in the local and the visibility models, and we compare their performances. In particular, we show that visibility is crucial to the reduction of time complexity, which, in fact, becomes optimal. This reduction however is obtained by increasing the number of agents (see Table 1 in the Conclusions for a summary of the results).

We then consider the cloning and synchronicity assumptions. We show that cloning is useful for reducing the move complexity — which becomes optimal — when it is combined with either visibility or synchronicity. Unfortunately, we could not achieve the same reduction while keeping an optimal number of agents.

An interesting consequence of the optimal $\Theta(\frac{n}{\sqrt{\log n}})$ bound on the number of agents that we obtain in the local model is the derivation of the search number for the hypercube in the classical graph search model, which was unknown. In fact, in the classical model, the lower bound would still hold, making our strategy optimal even when the agents are allowed to "jump."

1.2 Related Work

The decontamination problem has been introduced in [6, 25] and has been extensively studied in the literature under the term *graph search* (e.g., see [10, 19, 20, 24, 27]). The graph search problem considers a system of tunnels represented by the edges of a graph. Initially, all these tunnels are "contaminated" and have to be decontaminated or cleaned by a sequence of actions executed by a (minimal) set of *searchers*. The following operations are considered to be actions: (1) place a searcher on a node, (2) remove a searcher from a node, and (3) move a searcher along an edge. Many classes of graphs have been studied with the

main focus being on determining the optimal number of searchers or *search number*. The decision problem corresponding to the computation of the search number of a graph is NP-hard [24] with NP-completeness being shown in [4, 20]. In graph searching, there has been a particular interest in monotone strategies (e.g., see [4, 12, 20]). Monotonicity is a particularly interesting assumption because monotone strategies can always perform a polynomial number of moves (one move for each decontaminated node plus a few additional setup moves). An important result from Lapaugh has shown that monotonicity does not really change the difficulty of the graph search problem; in fact, it has been shown in [20] that for any graph there exists a monotone search strategy that uses the minimum number of agents.

The classical results for graph search 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 main difference in our setting is that agents *cannot be removed from the network*; they can only move from a node to a *neighbouring* one. This assumption is obviously motivated by the fact that we are considering software agents that are only able to move on the edges of the network. This additional constraint was introduced and first studied in [2] resulting in a *connected node search* where (i) the removal of agents is not allowed, and (ii) at any time of the search strategy, the set of clean nodes forms a connected subnetwork.

A connected graph search usually requires more agents to decontaminate a network G. It has been shown that for any graph G with n nodes the ratio between the connected search number $csn(G)$ and the regular search number $sn(G)$ is always bounded. More precisely, it is known that $csn(G)/sn(G) \leq \log n + 1$ (see [11]), and, for a tree T, $csn(T)/sn(T) \leq 2$ (see [3]). Monotonicity also plays an important role in connected graph searches. It has been shown that, as in the more general graph search problem, a solution allowing re-contamination of nodes cannot reduce the optimal number of agents required to decontaminate trees [2]. On the other hand, unlike the classical graph search, there exist graphs for which any given monotone connected graph search strategy requires more searchers than the optimal non-monotone connected search strategy [31]. The decision problem corresponding to the computation of the connected search number of a graph is NP-hard, but it is not known whether there exists a yes-certificate that is checkable in polynomial time.

With the connection assumption, the nature of the problem changes considerably and the classical results on graph search do not generally apply. The problem of finding the optimal number of agents has been

studied in some specific topologies. For example, it has been shown that the problem can be solved in linear time for trees [2]. Moreover, optimal strategies have been studied in chordal rings, tori, and meshes [14, 15], and in the Sierpiński graphs [22]. Arbitrary topology networks have also been considered: some heuristics have been proposed in [16] and an exponential move and time solution has been described in [5] to determine an optimal strategy.

Finally, both the connected and the non-connected versions of the problem have been studied (e.g., see [9, 11, 12]) under the assumption that the intruder is visible to the searchers (*visible search game*). The visible search number is also linked to standard graph parameters such as treewidth and pathwidth. Moreover, it has been shown that in the case of non-connected search any optimal strategy is monotone, while in the case of connected search there are graphs where the optimal strategy is not monotone.

In this paper we consider the same model as [2] and we use the term *decontamination* to refer to a connected monotone node search.

2 Definitions and Basic Properties

In a d-dimensional hypercube H_d with $n = 2^d$ nodes, each node corresponds to a d-bit binary string and two nodes are neighbours if their binary strings differ in precisely one bit. At each node x, there is a distinct label associated with each of its incident edges; the label between node x and node z is the position of the bit in which the corresponding binary strings differ, called *dimension*.

Let $G = H_d = (V, E)$ be a d-dimensional hypercube with $n = |V|$. Let $E(x)$ be the edges incident to $x \in V$. Let $\lambda_x : E(x) \rightarrow \{1, \ldots, d\}$ be an injective function defining the edge labelling for node x.

A team of autonomous mobile agents operates in G. Each agent is associated with a distinct identifier, can perform local computation, can move asynchronously from a node to a neighbouring one, and has some local memory ($O(\log n)$ bits suffice for all our algorithms). Moreover, each agent obeys the same set of behavioural rules, knows that it is operating in a hypercube, and can communicate with other agents only when they are simultaneously present at the same node (face-to-face communication). The environment is assumed to be asynchronous; that is, every action the agents perform (e.g., computing, moving) takes a finite but otherwise unpredictable amount of time.

Let us view the hypercube H_d as being organized in $d + 1$ levels and let level $i = 0, 1, \ldots, d$ consist

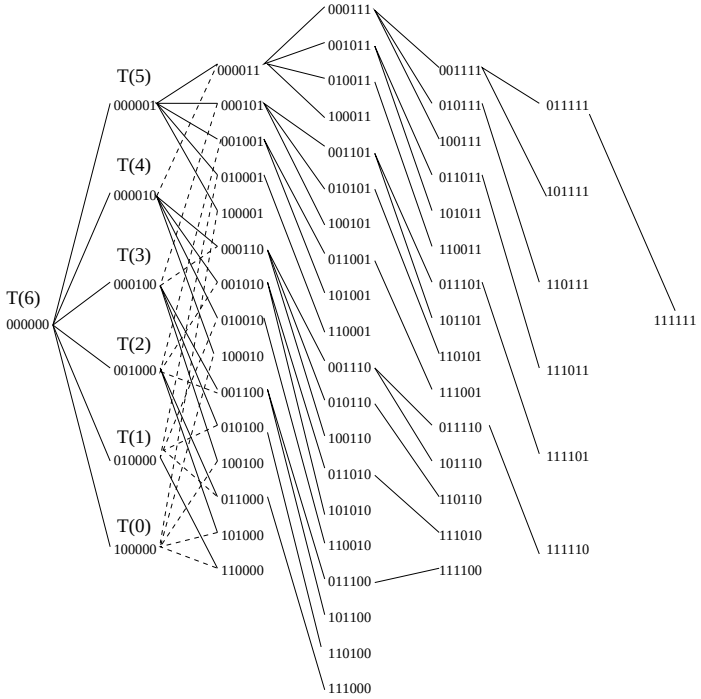

Figure 1: The broadcast tree T_6 of the hypercube H_6. Normal lines represent edges in T_6, dotted lines (only partially shown) the remaining edges of H_6.

of all the nodes whose binary representation contains i ones. Clearly, all the nodes at level i are connected only to nodes of level $i-1$ and to those of level $i+1$. Let $m(x)$ denote the position of the most significant bit of x.

Let T_d be a breadth-first spanning tree of H_d rooted at the source (node (00..00)) defined as follows: there is an edge in the spanning tree between x and all the nodes in the next level whose binary representation differs in a position higher than $m(x)$ (see Figure 1). This spanning tree is also called a *broadcast tree* because it is employed to perform optimal broadcast in the hypercube: a node x receiving a message from dimension i will forward it to all nodes connected by dimensions $j > i$.

Notice that the resulting spanning tree (also known as heap queue) has the following structure:

- a node of type $T(0)$ is a leaf
- a node of type $T(1)$ is a node with one child
- a node of type $T(k)$ is a node with k children of type $T(0), \ldots, T(k-1)$
- the source node $(00\ldots0)$ of T_d is of type $T(d)$

Let x, y be two neighbouring nodes ($(x,y) \in E(x)$). Node y is called a *small neighbour* of node x if

$\lambda_x(x,y) \leq m(x)$ and it is called a *big neighbour* of x if $\lambda_x(x,y) > m(x)$. Notice that the big neighbours of x are the children of x in the broadcast tree.

3 The Strategies

We first present a lower bound and then propose several strategies for the decontamination of the hypercube.

3.1 Lower Bound

In this section, we state a lower bound on the number of agents needed to clean the hypercube that holds for all our models.

The lower bound is linked to the notion of *pathwidth*, a classical graph parameter defined in [28]. A *path decomposition* of a graph $G = (V, E)$ is a collection $\{X_1, \ldots, X_r\}$ of subsets of V where:

- $\cup_{i=1}^{r} X_i = V$
- $\forall (x,y) \in E$, $\exists i \in \{1, \ldots, r\}$: $x, y \in X_i$
- $\forall i, j, k$ with $1 \leq i < j < k \leq r$, $X_i \cap X_k \subseteq X_j$

The *width* of a path decomposition $(X_1, \ldots, X_r)$ is defined as $\max_{1 \leq i \leq r} |X_i| - 1$, and the *pathwidth* of G is the minimum width over its path decompositions.

Theorem 1. *To solve the connected monotone search problem,* $\Omega(\frac{n}{\sqrt{\log n}})$ *agents are required.*

PROOF It is known that the node search number of a graph (in the classical model) is equal to its pathwidth plus one [19]. From [8, 7] we know that the pathwidth of a hypercube is $\Theta(\frac{n}{\sqrt{\log n}})$. Since the lower bound in the classical model is obviously the lower bound also in our model, the theorem follows. ∎

3.2 Local Model: Agent-Optimal Strategy

For ease of discussion in this section, we assume that the degree of the hypercube d is even and that $d \geq 4$. The same bounds are obtained for odd degrees with minor technical modifications.

The first strategy we present is optimal in terms of the number of agents. The main idea is that all the agents, starting from the same homebase, have their moves coordinated by an agent that acts as a leader. The agents visit all nodes while protecting the system from re-contamination.

3.2.1 Basic Properties

We start by reviewing a basic property that we will need for our strategy:

Property 1. *Let T_d be a broadcast tree for the hypercube.*

1. *The number of nodes at level l is $\binom{d}{l}$.*
2. *Level 0 has no leaves, level $l > 0$ has $\binom{d-1}{l-1}$ leaves, and the total number of leaves is 2^{d-1}.*
3. *At level 0 there is a unique node of type $T(d)$. At level $l > 0$ there are $\binom{d-k-1}{l-1}$ nodes of type $T(k)$ with $0 \leq k \leq d-l$.*

In the rest of the paper we will sometimes use an alternative labelling for the nodes of the hypercube. Given a node with binary representation x, we can also label it with a vector $\widehat{x}$ defined as follows: $\widehat{x} = \emptyset$ if x is the source node; otherwise, $\widehat{x} =< i_1, i_2, ..., i_l >$ where the i_k, for $1 \leq k \leq l \leq d$, are the increasing positions of the 1 bits in x, i.e., $i_k > i_{k-1}$ for $2 \leq k \leq l$. For example, node (001110) can also be labelled as $< 2, 3, 4 >$. In the following we will consider the nodes at each level of the broadcast tree to be lexicographically ordered according to this labelling (this is the order in which nodes appear in each level in Figure 1) and not according to their binary representation. Moreover, we will use the notations $<_R$ and $>_R$ when referring to lexicographic comparisons.

3.2.2 Cleaning strategy

One of the agents (e.g., the one with smallest id), will act as the *coordinator* for the entire cleaning process.

The cleaning strategy is carried out on the broadcast tree. The structure of the broadcast tree guarantees that when a level l is fully guarded all the agents on a node $\widehat{i}$ of level l can move to the next level without incurring a re-contamination of node $\widehat{i}$ if the agents on nodes $\widehat{j} <_R \widehat{i}$ (in level l) have already moved to the next level. In other words, the broadcast tree and the alternative labelling $\widehat{x}$ define a correct cleaning order for the nodes. The main idea is then to place enough agents on node $(00\ldots00)$ and to coordinate their movement on the edges of the broadcast tree, level by level. The group of agents available at the root is called the *set of available agents*.

Algorithm CLEAN

1. From the root to level 1

1.1 The coordinator, d times, guides a distinct agent from the root of the tree to each of its d children of types $T(d-1), \ldots, T(0)$ and each time returns to the root.

2. From level $l \geq 1$ **to level** $l+1 \leq d$ (level l has one agent per node)

2.1 Before starting to clean nodes at level $l+1$, the coordinator moves back 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 $T(k)$ with $0 \leq k \leq d-l$, except for the nodes of type $T(1)$ and $T(0)$ which do not require any extra agents). The coordinator sends $k-1$ additional agents, in no specific order, to each node of type $T(k)$, $k > 1$, at level l and then moves to the first node of level l.

2.2 When k agents are on a node of type $T(k)$ at level l, they are sent down the broadcast tree to the children at level $l+1$, guided by the coordinator. Let $\widehat{n_1}, \ldots, \widehat{n_m}$ ($m = \binom{d}{l}$) be the lexicographically-ordered nodes of level l. The coordinator sequentially chooses each node at level l following this lexicographical ordering and node by node guides an agent on each outgoing edge of the broadcast tree to level $l+1$.

2.3 When the coordinator reaches a leaf of level l, the agent it was guiding becomes available and returns to the root. Notice that when the coordinator reaches the last node of level l, the only active agents are the ones covering level $l+1$.

Figure 2 shows for H_4 the order in which the nodes are cleaned by the agents led by the coordinator.

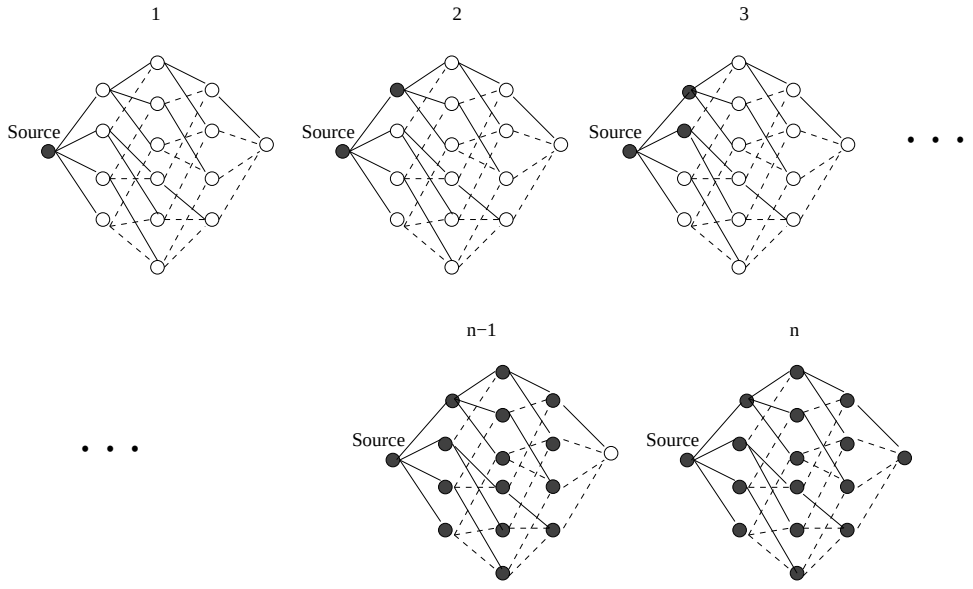

Figure 2: Algorithm CLEAN on a hypercube H_4. The nodes get cleaned sequentially.

3.2.3 Correctness and Analysis

Correctness. We now prove that Algorithm CLEAN is correct; that is, all the nodes in the hypercube will be cleaned and once a node has been cleaned it will never be re-contaminated.

Let x be a node of level l; let $N(x)$ denote the neighbours of x at level $l+1$; and let $NT(x)$ denote the children of x at level $l+1$ in the broadcast tree. Notice that $N(x)$ includes $NT(x)$ and possibly some other neighbours at level $l+1$. The following lemma contains an obvious observation:

Lemma 1. *If $z \in N(y) \setminus NT(y)$ then $z \in NT(x)$ for some x such that $\widehat{x} <_R \widehat{y}$.*

Lemma 2. *In Algorithm* CLEAN*, when agents leave unguarded a node x at level l, all the neighbours of x are either clean or guarded.*

PROOF This is clearly true for the node at level 0. Assume it is true for all nodes at 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 the nodes at level i are sending agents to level $i+1$, by the induction hypothesis and the cleaning strategy all nodes at level $i-1$ are clean. Let us now assume that cleaning occurs at node y of level i. For any $z \in N(y) \setminus NT(y)$, by Lemma 1, $\exists x$ such that $z \in NT(x)$ and $\widehat{x} <_R \widehat{y}$. By the cleaning strategy, the coordinator visits the nodes in lexicographical order at each level. So before the coordinator reaches node y, agents on node x are already sent to level $i+1$ and all $z \in N(y) \setminus NT(y)$ are already guarded by an agent each. If y is a node of type $T(j)$ where $j \geq 1$, by Step 2.1 of Algorithm CLEAN, enough agents are sent from the set of available agents on the homebase 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 its only child at level $i+1$. After all the agents on y are sent down to the children of y at level $i+1$, each node of $NT(y)$ is guarded by an agent and all the neighbours of y are either clean or guarded by an agent. If y is a leaf then $NT(y)$ is empty. When the coordinator reaches it, all the neighbours of y at level $i+1$ are guarded and all the neighbours at level $i-1$ are clean. ∎

Theorem 2. *The cleaning process of Algorithm* CLEAN *decontaminates all nodes. During the execution clean nodes cannot be re-contaminated.*

PROOF In Algorithm CLEAN, the decontamination is performed level by level, so when it reaches level d, all nodes have been cleaned. The fact that a clean node will not be re-contaminated directly follows from

Lemma 2. ∎

Complexity. To calculate the number of agents needed to perform the cleaning of the hypercube with Algorithm CLEAN, we first compute the number of agents that are taken from the set of available agents before performing the cleaning from a level to the next.

Lemma 3. *In Algorithm* CLEAN*, before cleaning from level $l \geq 1$ to level $l+1 \leq d$, $\binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$ extra agents are fetched from the root by the coordinator.*

PROOF In Step 2.1 of Algorithm CLEAN, $k-1$ extra agents for a node of type $T(k)$ are sent from the root. By Property 1 part 3, at level $l \geq 1$ there are $\binom{d-k-1}{l-1}$ nodes of type $T(k)$ with $0 \leq k \leq d-l$. So, in total, $\sum_{k=2}^{d-l}(k-1)\binom{d-k-1}{l-1}$ extra agents are sent from the root to level l while cleaning from level l to level $l+1$. Note that by first choosing $i = k-1$ and then $L = l-1$ we have

$\sum_{k=2}^{d-l}(k-1)\binom{d-k-1}{l-1} = \sum_{i=1}^{d-l-1} i\binom{d-(i+1)-1}{l-1} = \sum_{i=1}^{d-L-2}\binom{i}{1}\binom{d-i-2}{L}$.

Observe now that given $a, b \in \mathbf{N}$ we have $\binom{a}{b} = 0$ for $a < b$. Therefore:

$\sum_{i=1}^{d-L-2}\binom{i}{1}\binom{d-i-2}{L} = \sum_{i=1}^{d-2}\binom{i}{1}\binom{d-i-2}{L} = \sum_{i=0}^{d-2}\binom{i}{1}\binom{d-i-2}{L}$.

Referring to [18], we have $\sum_{i=0}^{d-2}\binom{i}{1}\binom{d-i-2}{L} = \binom{d-1}{L+2}$; thus,

$\sum_{i=0}^{d-2}\binom{i}{1}\binom{d-i-2}{L} = \binom{d-1}{l+1} = \binom{d}{l+1} - \binom{d-1}{l} = \binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$. ∎

We now compute the number of agents used by the algorithm to clean from a level to the next.

Lemma 4. *In Algorithm* CLEAN*, no more than $\binom{d}{\frac{d}{2}} + \binom{d-1}{\frac{d}{2}-2} + 1$ agents are used to clean from level $l \geq 1$ to level $l+1 \leq d$.*

PROOF By induction.

The lemma holds for level 1 because only d agents are needed and $d < \binom{d-1}{\frac{d}{2}-2} + \binom{d}{\frac{d}{2}} + 1$ for $d \geq 4$.

Let us assume that it holds for level $l \geq 1$ (i.e., after cleaning l levels). At this point, we have $\binom{d}{l} + 1$ active agents (including the coordinator) and every node of level l is guarded by one agent; all the other agents are available. We now show that our strategy does not use more than $\binom{d}{\frac{d}{2}} + \binom{d-1}{\frac{d}{2}-2} + 1$ agents to clean level $l+1$.

By Lemma 3, before cleaning level $l+1$, the coordinator collects $\binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$ extra agents, and thus exactly k agents for each node of type $T(k)$. In fact, by the induction hypothesis, each node at level l is already guarded by one agent before the extra agents arrive. 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 $\max_{1\leq l\leq d-1}\{\binom{d}{l+1} + \binom{d-1}{l-1}\} = \binom{d}{\frac{d}{2}+1} + \binom{d-1}{\frac{d}{2}-1} = \binom{d}{\frac{d}{2}} + \binom{d-1}{\frac{d}{2}-2}$ for $l = \frac{d}{2}$ or $l = \frac{d}{2} - 1$, respectively. Therefore, $\binom{d}{\frac{d}{2}} + \binom{d-1}{\frac{d}{2}-2} + 1$ is the maximum number of agents required by our algorithm and corresponds to the cleaning of the central level (this number corresponds to the nodes of level $\frac{d}{2}$, plus the leaves of the broadcast tree at level $\frac{d}{2} - 1$). ∎

Theorem 3. *Algorithm* CLEAN *uses* $\Theta(\frac{n}{\sqrt{\log n}})$ *agents to clean the hypercube, which is optimal.*

PROOF The lower bound follows from Theorem 1. The upper bound follows from Lemma 4 observing that, by the Stirling approximation, $\binom{d}{\frac{d}{2}} = O(\frac{4^d}{\sqrt{d}})$ which is $O(\frac{n}{\sqrt{\log n}})$. ∎

Notice that the optimal bound derived above on the number of agents needed to decontaminate the hypercube holds also in the classical graph search model when the agents can "jump".

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

Theorem 4. *The total number of moves performed by the agents in Algorithm* CLEAN *is* $O(n \log n)$. *The time complexity is* $O(n \log n)$ *time units.*

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

The number of moves performed by the agents:

It takes $2l$ moves for an agent to arrive from the root to a leaf of level l and go back to the root. By Property 1 part 2, there are $\binom{d-1}{l-1}$ leaves at level l. So in total there are $\sum_{l=1}^{d} 2l\binom{d-1}{l-1}$ moves made by the agents. To compute this quantity first note that $\binom{d}{l}$ is the number of nodes at level l. Summing over all the levels we obtain $\sum_{l=0}^{d}\binom{d}{l} = 2^d = n$. It follows that $\sum_{l=1}^{d}\binom{d-1}{l-1}\sum_{l=0}^{d-1}\binom{d-1}{l} = 2^{d-1} = \frac{n}{2}$.

We now have to compute

$\sum_{l=1}^{d} l\binom{d-1}{l-1} = 1\binom{d-1}{0} + \cdots + (\frac{d}{2}-1)\binom{d-1}{\frac{d}{2}-2} + \frac{d}{2}\binom{d-1}{\frac{d}{2}-1} + (\frac{d}{2}+1)\binom{d-1}{\frac{d}{2}} + (\frac{d}{2}+2)\binom{d-1}{\frac{d}{2}+1} + \cdots + d\binom{d-1}{d-1}$.
Using the property that $\binom{a}{b} = \binom{a}{a-b}$ we may group terms in pairs and obtain
$\sum_{l=1}^{d} l\binom{d-1}{l-1} = (d+1)\binom{d-1}{0} + \cdots + (d+1)\binom{d-1}{\frac{d}{2}-1} = (d+1)\sum_{l=0}^{\frac{d}{2}-1}\binom{d-1}{l}$.
We already know that $\sum_{l=0}^{d-1}\binom{d-1}{l} = 2^{d-1} = \frac{n}{2}$; we also know that $\sum_{l=0}^{d-1}\binom{d-1}{l} = 2\sum_{l=0}^{\frac{d}{2}-1}\binom{d-1}{l}$, so $(d+1)\sum_{l=0}^{\frac{d}{2}-1}\binom{d-1}{l} = (d+1)2^{d-2} = \frac{n}{4}(\log n + 1)$.
Finally, the total number of moves performed by the agents is

$$\sum_{l=1}^{d} 2l\binom{d-1}{l-1} = \frac{n}{2}(\log n + 1) = O(n \log n).$$

The number of moves performed by the coordinator:

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

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

3. Navigate within each level to get to the next node. Recalling that the procedure is run on a hypercube, we know that at level l the coordinator needs to navigate at most $2l$ edges if $l \leq \frac{d}{2}$; otherwise, $2d - 2l$ edges are needed to reach the next node at the same level. So, in total, the number of moves is at most $\sum_{l=1}^{\frac{d}{2}} 2l\binom{d}{l} + \sum_{l=\frac{d}{2}+1}^{d-1}(2d-2l)\binom{d}{l} = 4\sum_{l=1}^{\frac{d}{2}-1} l\binom{d}{l} + d\binom{d}{\frac{d}{2}} = O(n \log n)$.

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 traversed twice by the coordinator. So, in total, there are $2(2^d - 1) = 2(n-1)$ moves.

In total, the number of moves performed during the cleaning process is $O(n \log n)$.

Recall that we are working in an asynchronous environment, so we now consider the ideal time complexity for the cleaning strategy (i.e., we assume that it takes one unit of time for an agent to traverse an edge). Observing that the cleaning process is carried out sequentially by the coordinator, we see that the time required is equal to the number of moves of the coordinator. ∎

Note. Let us now briefly discuss what happens if the number of dimensions is odd. Since the algorithm does not depend on whether d is odd or even, the correctness still holds. The only thing affected is the

maximum number of agents employed. In fact, in the proof of Lemma 4, we have shown that the number of agents employed is $\max_{1\leq l\leq d-1}\{\binom{d}{l+1}+\binom{d-1}{l-1}\}+1$. When d is even, we have seen that this number corresponds to the number of nodes $\binom{d}{\frac{d}{2}}$ in the maximum level, plus the number of leaves $\binom{d-1}{\frac{d}{2}-2}$ in the previous level, plus 1. When d is odd, there are two central levels ($\frac{d+1}{2}$ and $\frac{d-1}{2}$) in the broadcast tree with the same number of nodes. In this case, $\max_{1\leq l\leq d-1}\{\binom{d}{l+1}+\binom{d-1}{l-1}\}+1$ corresponds to the number of nodes $\binom{d}{\frac{d+1}{2}}$ in level $\frac{d+1}{2}$ plus the number of leaves $\binom{d-1}{\frac{d+1}{2}-1}$ (or equivalently $\binom{d-1}{\frac{d-1}{2}}$) in the previous level, plus 1. This value is clearly still $O(\frac{n}{\sqrt{\log n}})$.

3.3 Visibility Model: Time-Optimal Strategy

In this section, we propose a solution to the decontamination problem in a model where the agents can "see" the state (clean, guarded, or contaminated) of all the neighbouring nodes. In fact, we make the following assumption about agent *visibility*:

An agent located at node x can see the state of its neighbours $N(x)$.

As we will see, the visibility assumption allows the agents to decide the next move solely on the basis of their local knowledge and without the need of being led by a coordinator. This feature allows us to reduce the time complexity, at the expense, however, of an increase in the number of agents.

3.3.1 Basic Properties

We now introduce some basic properties that we will need to address the problem (see [13] for the proofs).

Let C_i be the set of nodes whose most significant bit is in the i-th position (see Figure 3). Thus, C_d is exactly the set of all the leaves of the broadcast tree.

Property 2. *C_0 contains exactly one node; C_i, for $0 < i \leq d$, contains 2^{i-1} nodes.*

Property 3. *Let x be any node in C_i with $i > 0$. One small neighbour of x is in C_j (where $j < i$), and the remaining small neighbours, if any exist, are in C_i. The big neighbours of x, if any exist, are in some C_k (where $k > i$).*

Property 4. *Let x be any node in C_i with $i > 1$. There must exist at least one small neighbour y of x such that y is in C_i, and a small neighbour z of y such that z is in C_{i-1}.*

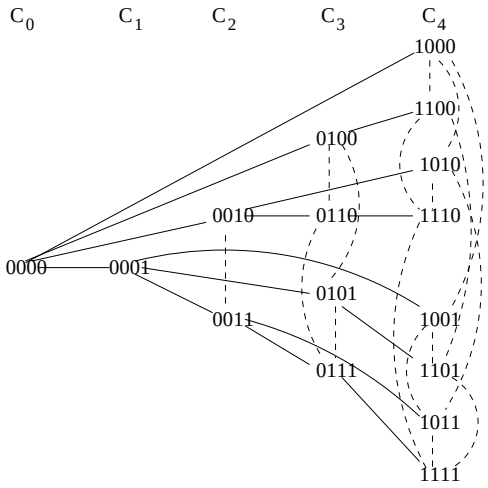

Figure 3: The hypercube H_4 in the new group structure.

3.3.2 Cleaning strategy

All the agents are initially located at the root (the homebase or source) of the broadcast tree; they are identical and autonomous; and they follow the same local rules. The agents move along the broadcast tree as in the previous model, but they do so without requiring coordination. In fact, they can independently proceed to clean the children (or big neighbours) in the broadcast tree when they "see" that the other neighbours (all the small neighbours) are either clean or guarded.

Algorithm CLEAN WITH VISIBILITY

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

1. Wait until 2^{k-1} agents are on x.
2. If $k < d$ then wait until all the small neighbours of x are clean or guarded.
3. One agent moves to the big neighbour of type $T(0)$; 2^{i-1} agents move to each of the big neighbours of type $T(i)$, for $0 < i < k$; and if there are no big neighbours, terminate.

Figure 4 shows the order in which the nodes of H_4 get cleaned with our strategy. As opposed to the strategy of the previous section, nodes are not cleaned sequentially; several nodes, in fact, could be cleaned independently.

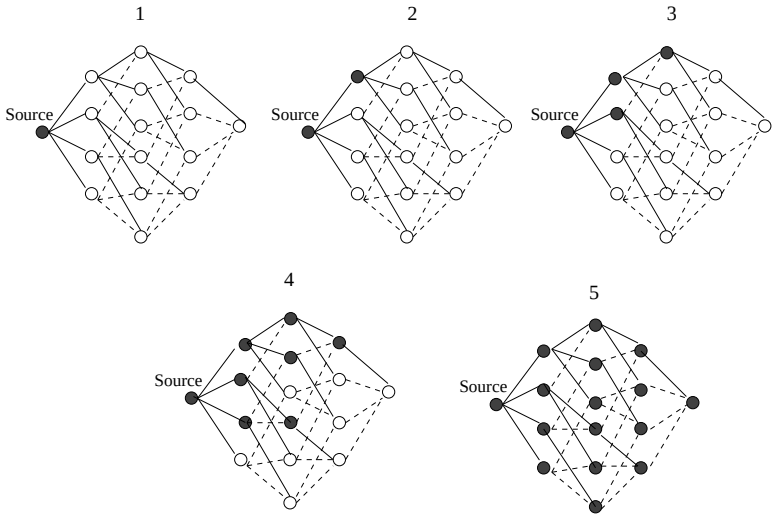

Figure 4: The execution of Algorithm CLEAN WITH VISIBILITY on a hypercube H_4.

3.3.3 Correctness and Complexity

Theorem 5. *The total number of agents needed to clean the d-dimensional hypercube using Algorithm* CLEAN WITH VISIBILITY *is $\frac{n}{2}$.*

PROOF By definition, Algorithm CLEAN WITH VISIBILITY sends one agent from level 0 to level 1 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 Algorithm CLEAN WITH VISIBILITY is correct; in other words, the network is clean and once a node has been cleaned it will never be re-contaminated.

Lemma 5. *In Algorithm* CLEAN WITH VISIBILITY*, when agents leave a node in C_i (leaving it unguarded) all its small 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 the argument presented in the proof of Theorem 5, every node will have enough agents to continue the cleaning process, and, by the second rule of the algorithm, the agents on x move to the big

neighbours only when all other neighbours are clean or guarded. ∎

Theorem 6. *During the cleaning process of Algorithm* CLEAN WITH VISIBILITY*, the agents clean all the nodes and a clean node will not be re-contaminated.*

PROOF 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 re-contaminated directly follows from Lemma 5. ∎

We now consider the time complexity of the cleaning strategy.

Theorem 7. *Cleaning the network with Algorithm* CLEAN WITH VISIBILITY *takes* $\Theta(\log n)$ *time units.*

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

Base case: At time $i = 0$, all the agents are placed on node $(00...00)$, the homebase. First notice that, since there are no agents on any other node at time 0, only the agents on C_0 can move at this time. By the cleaning strategy, the agents clean the homebase and then move to clean the d big neighbours, which are in C_j for $0 < j \leq d$. Node $(00..00)$ becomes clean at time 0; obviously, it cannot be re-contaminated. 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 the induction hypothesis and Theorem 6, all the nodes in C_k are clean for any $0 \leq k \leq i$. The agents that were once on them have left. Let node x be an arbitrary node in C_{i+1}. By Property 3, exactly one small neighbour of x is in C_k for $k \leq i$. By the induction hypothesis, at time k, the agents arrive at x and clean it. So at time $i + 1$, every node in C_{i+1} is guarded by at least one agent. Thus, all small neighbours of the nodes in C_{i+1} are clean or guarded. So, at time $i + 1$, every agent in C_{i+1} executes the algorithm; they clean the big neighbours, which, by Property 3, are in C_j with $j > i + 1$. Because the nodes in C_{i+1} have already been cleaned by their guarding agents upon arrival and because by Theorem 6 a clean node will not be re-contaminated, we know that at time $i + 1$ the nodes in C_{i+1} become clean after the agents on them move on.

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 small neighbours are not guarded. Let us consider any node y in C_j on which there are

agents. By Property 4, there exists one small neighbour z of y which is in C_j too and a small neighbour w of z which is in C_{j-1}, where $j-1 \geq i+1$. We know that the agents on z come from its small neighbour 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 the 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 small neighbours is not guarded.

The time complexity is clearly optimal, since $\log n$ is the diameter of the hypercube and the agents are initially all located in the same node. ∎

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

Theorem 8. *The number of moves performed by the agents in Algorithm* CLEAN WITH VISIBILITY *for the cleaning is* $O(n \log n)$.

PROOF All the agents start from the source (the homebase) 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). ∎

3.4 Visibility and Cloning: Time and Move-Optimal Strategy

Cloning is the capability for an agent to create copies of itself. In this section, we consider the visibility model where the agents have cloning capabilities. In this case we show that no coordinator is necessary to control the cleaning; in fact, all the agents can autonomously follow the same algorithm. Cloning allows the agents to make a smaller number of moves.

Initially, one agent is placed at node $(00...00)$, the homebase. It cleans node $(00...00)$ and clones $d-1$ new agents. In this way d agents are available on the node. Then one agent per edge is sent to clean each of the d neighbours. When an agent arrives at a node, it cleans it and then executes the following algorithm.

Algorithm CLEAN WITH VISIBILITY AND CLONING

Rule for the agents on node x:

1. Wait until all the small neighbours of x are clean or guarded.
2. Clone enough agents to clean the big neighbours of x and then send one agent per edge to clean each of the big neighbours. If all the small neighbours are guarded or clean and there are no big neighbours, terminate.

Similarly to Theorem 6 we have:

Theorem 9. *During the cleaning process of Algorithm* CLEAN WITH VISIBILITY AND CLONING *the agents clean all nodes and a clean node will not be re-contaminated.*

Regarding the complexity, first observe that agents can clone new agents whenever they are needed, so there are always enough agents to clean the network.

Theorem 10. *The cleaning of the network with Algorithm* CLEAN WITH VISIBILITY AND CLONING *requires* $\log n$ *time units,* $\frac{n}{2}$ *agents, and* $n-1$ *moves. The time and move complexities are optimal.*

PROOF The bound on the time units may be computed similar to the proof of Theorem 7.

Let us now compute the number of agents employed. Since all agents have to terminate, instead of counting how many agents are created overall, 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 exactly one agent, which then clones itself if there are big neighbours to clean. By Algorithm CLEAN WITH VISIBILITY AND CLONING, the agent on a node terminates only if the node does not have big neighbours. It is easy to see that there are 2^{d-1} such nodes. Each of these 2^{d-1} nodes is visited by exactly one agent. Hence the total number of agents used is equal to $2^{d-1} = \frac{n}{2}$.

Finally, the computation of the number of moves trivially follows from the fact 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.

The time complexity is optimal because $\log n$ is the diameter of the hypercube, and the agents start from the same location. The number of moves is also optimal because all nodes must be visited. ∎

3.5 Local, Cloning, and Synchronicity: Time and Move-Optimal Strategy

We now show that the same result obtained in the previous section for the visibility model with cloning can be achieved in the local model when cloning is available and the system is synchronous. In fact, we describe a strategy for the local model that exploits synchronicity to obtain an optimal time complexity, and cloning to obtain an optimal number of moves: this is done, however, at the expense of the number of agents used.

Our approach is based on the observation that, even if the agents do not have visibility, they can still move autonomously thanks to the synchronicity of the system. In this setting, in fact, synchronicity can be exploited by using a strategy very similar to the one of Algorithm CLEAN WITH VISIBILITY but without the need for the visibility assumption. Instead of waiting for all small neighbours to become clean or guarded, the agents on a node wait for an appropriate amount of time before moving to clean the big neighbours.

Recall that $m(x)$ denotes the position of the most significant bit of x. In the synchronous model, the agents on x can move to the big neighbours when time $t = m(x)$ because they *implicitly* know that at this time all the small neighbours of x are clean or guarded.

At time 0, one agent is placed at node $(00...00)$, the homebase. It cleans node $(00...00)$ and clones $d-1$ new agents. One agent per edge is then sent to clean each of the d neighbours. At time i, an agent arrives at a node, cleans the node and then executes the following algorithm.

Algorithm CLEAN WITH CLONING AND SYNCHRONICITY

Rule for the agents on node x:

1. Wait until $t = m(x)$.
2. Clone enough agents to clean the big neighbours of x and then send one agent per edge to clean each of the big neighbours. If there are no big neighbours, terminate.

The correctness follows from the next result:

Lemma 6. *At time i, $0 \leq i \leq d$, of Algorithm* CLEAN WITH CLONING AND SYNCHRONICITY*, there is one agent on every node of C_i. This node clones other agents and cleans the big neighbours. All nodes in C_j, for $0 \leq j \leq i$, are clean. At time d, all the agents on the nodes of C_d terminate.*

PROOF by induction.

Base case: At time $i = 0$ only node $(00..00)$ is in C_0, and the cleaning strategy places exactly one agent on it. This agent cleans the node, clones itself, and cleans its neighbours, so no re-contamination may occur and at time 1 all the nodes in C_0 and C_1 are clean.

Inductive step: First assume that at time i, for $0 \leq i \leq d-1$, there is one agent on every node of C_i, and that all nodes in C_j, for $0 \leq j \leq i$, are clean.

We show that at time $i+1$, there is one agent on every node in C_{i+1}. This agent either clones itself and moves to a big neighbour (if any) or terminates. Moreover, all nodes in C_{j+1}, for $0 \leq j \leq i$, are clean.

Let node x be any node in C_{i+1}. By Property 3, exactly one small neighbour of x, which we will call z, is in C_j, for some $j \leq i$, and all the other small neighbours of x, if any, are in C_{i+1}. By the induction hypothesis, at time j, there is one agent on z. Hence, by Algorithm CLEAN WITH CLONING AND SYNCHRONICITY, at time j, one agent is sent from z to x and has to stop at x up to time $t = i+1$. At this time, if $i+1 \leq d-1$ then the agent clones enough new agents and cleans its neighbours; otherwise, $i+1 = d$ and the agent is in C_d, meaning that it is on a leaf so it terminates. Moreover, at time $i+1$, every node in C_{i+1} is guarded by an agent. The only contaminated neighbours of x are the big neighbours. The agent on x clones enough new agents, cleans its neighbours, and x moves from a guarded to a clean state together with its small neighbours in C_{i+1}. Thus, the nodes in C_{j+1}, for $0 \leq j \leq i$, cannot be re-contaminated. ∎

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

It follows from Lemma 6 that:

Theorem 11. *During the cleaning process of Algorithm* CLEAN WITH CLONING AND SYNCHRONICITY *the agents clean all nodes and a clean node will not be re-contaminated.*

Regarding the complexity we have the following:

Theorem 12. *The cleaning of the network with Algorithm* CLEAN WITH CLONING AND SYNCHRONICITY *requires* $\log n$ *time units,* $\frac{n}{2}$ *agents, and* $n-1$ *moves. The time and move complexities are optimal.*

PROOF The bound on the time units comes from Lemma 6, and the bound on the number of agents and moves is similar to the proof of Theorem 10. ∎

4 Observations and Open Problems

In this paper we have considered the problem of decontaminating a hypercube network and, starting with the basic local model, we have considered additional assumptions on the agents' and system's capabilities (visibility, cloning, and synchronicity). Our goal was to start a study on the impact that these additional assumptions have on the efficiency of the solution process to the decontamination problem in general networks.

		Agents	**Time**	**Moves**
LOCAL	**Local**	$\Theta(\frac{n}{\sqrt{\log n}})$	$O(n \log n)$	$O(n \log n)$
	Local, Cloning, Synchronicity	$n/2$	$\Theta(\log n)$	$\Theta(n)$
VISIBILITY	**Visibility**	$n/2$	$\Theta(\log n)$	$O(n \log n)$
	Visibility and Cloning	$n/2$	$\Theta(\log n)$	$\Theta(n)$

Table 1: Comparisons of results for the various models.

From our observations (see Table 1 for a summary), visibility seems to be a crucial assumption for the reduction of time complexity, which, in fact, becomes optimal. However, we have been able to obtain this reduction only at the expense of increasing the number of agents. The same reduction can also be obtained in the local model, but only when both synchronicity and cloning are assumed. We have not been able to achieve the reduction with cloning or synchronicity alone. Furthermore, cloning is certainly useful for reducing the move complexity, which, in fact, becomes optimal when cloning is available in the visibility model. However, with cloning alone, we have not been able to obtain an optimal move complexity in the local model, where we can only achieve it by adding synchronicity. Another interesting observation concerns synchronicity. It appears to be useless when the system has visibility. As mentioned above, it remains an open problem whether synchronicity indeed does not add any power to a setting where the agents have visibility.

We observe that the use of an optimal number of agents in the weaker local model is obtained at the expense of the use of coordination. In fact, in our algorithms, whenever coordination is not employed, the number of agents grows. It is an open problem to design an agent-optimal strategy that does not assume the use of a coordinator. Vice versa, it would be interesting to prove that this strategy does not exist. Also,

regarding the optimal number of agents, we observe that none of our strategies achieves both optimal agent and time complexities. We conjecture that these two parameters are linked and that it is impossible to design an algorithm that minimizes both. It would be very interesting to prove this conjecture or find an example where such complexity can be achieved.

An interesting research direction is the investigation of various levels of visibility. In this paper we have considered the cases where there is no visibility, and where visibility is limited to neighbouring nodes. The extreme case of total visibility (i.e., where the agents can see the whole network) has been studied (see [9, 11, 12]) leaving the study of the intermediate visibility levels open.

Another interesting problem which we are now investigating is to determine, given a network topology, the biggest area that can be decontaminated with a fixed number of agents.

Acknowledgments. We would like to thank the anonymous referees for their helpful comments and Matthew Kellett for carefully proofreading the manuscript.

References

[1] M. Asaka, S. Okazawa, A. Taguchi, and S. Goto, A method of tracing intruders by use of mobile agents, Proc 9th Annual Conference of the Internet Society (INET), San Jose, CA,1999, webpage. http://www.isoc.org/inet99/proceedings/4k/4k_2.htm.

[2] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro, Capture of an intruder by mobile agents, Proc 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA), Winnipeg, Manitoba, Canada, 2002, pp. 200–209.

[3] L. Barrière, P. Fraigniaud, N. Santoro, and D.M. Thilikos, Searching is not jumping, Proc 29th International Workshop on Graph Theoretic Concepts in Computer Science (WG), Elspeet, the Netherlands, Lecture Notes in Computer Science, Vol. 2880, Springer, 2003, pp. 34–45.

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

[5] L. Blin, P. Fraigniaud, N. Nisse, and S. Vial, Distributed chasing of network intruders by mobile agents, Proc 13th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Chester, UK, Lecture Notes in Computer Science, Vol. 4056, Springer, 2006, pp. 70–84.

[6] D. Breisch, An intuitive approach to speleotopology, Southwestern Cavers VI (1967), 72–78.

[7] L.S. Chandran and T. Kavitha, The treewidth and pathwidth of hypercubes, Discrete Mathematics 306:3 (2006), 359–365.

[8] L.S. Chandran, T. Kavitha, and C. R. Subramanian, Isoperimetric inequalities and the width parameters of graphs, Proc 9th Annual International Computing and Combinatorics Conference (COCOON), Big Sky, MT, Lecture Notes in Computer Science, Vol. 2697, Springer, 2003, pp. 385–393.

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

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

[11] P. Fraigniaud and N. Nisse, Connected treewidth and connected graph searching, Proc 7th Latin American Theoretical Informatics Symposium (LATIN), Lecture Notes in Computer Science, Vol. 388, Springer, Valdivia, Chile, 2006, pp. 479–490.

[12] P. Fraigniaud and N. Nisse, Monotony properties of connected visible graph searching, Proc 32nd Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG), Bergen, Norway, 2006, pp. 229–240.

[13] P. Flocchini, M.J. Huang, and F.L. Luccio, Capturing an intruder in the hypercube by mobile agents, Technical Report TR-2005-2, School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada, 2005.

[14] P. Flocchini, M.J. Huang, and F.L. Luccio, Decontaminating chordal rings and tori using mobile agents, International Journal of Foundations of Computer Science 18:3 (2007), 547–564.

[15] P. Flocchini, F.L. Luccio, and L.X. Song, Size optimal strategies for capturing an intruder in mesh networks, Proc International Conference on Communications in Computing (CIC), Las Vegas, NV, 2005, pp. 200–206.

[16] P. Flocchini, A. Nayak, and A. Shulz, Cleaning an arbitrary regular network with mobile agents, Proc 2nd International Conference on Distributed Computing & Internet Technology (ICDCIT), Bhubaneswar, India, 2005, pp. 132–142.

[17] N. Foukia, J.G.Hulaas, and J. Harms, Intrusion detection with mobile agents, Proc 11th Annual Conference of the Internet Society (INET), Stockholm, Sweden, 2001, webpage. http://www.isoc.org/inet2001/CD_proceedings/Foukia/inet.pdf.

[18] R.L. Graham, D.E. Knuth, and O. Patashnik, Concrete mathematics, Second Edition, Addison-Wesley, Reading, MA, 1994.

[19] L. M. Kirousis and C. H. Papadimitriou, Searching and pebbling, Theoretical Computer Science 47 (1986), 205–218.

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

[21] F.T. Leighton, Introduction to parallel algorithms and architectures: Arrays, trees, hypercubes, Morgan Kaufmann Publishers Inc. San Francisco, CA, 1991.

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

[23] F. Luccio, L. Pagli, and N. Santoro, Network decontamination in presence of local immunity, International Journal of Foundations of Computer Science 18:3 (2007), 457–474.

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

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

[26] T. Parson, The search number of a connected graph, Proc 9th Southeastern Conference on Combinatorics, Graph Theory and Computing, Utilitas Mathematica, 1978, pp. 549–554.

[27] S. Peng, M. Ko, C. Ho, T. Hsu, and C. Tang, Graph searching on chordal graphs, Algorithmica 27 (2002), 395–426.

[28] N. Robertson and P.D. Seymour, Graph minors I, excluding a forest, Journal of Combinatorial Theory Series B 35 (1983), 39–61.

[29] N. Robertson and P.D. Seymour, Graph minors II, algorithmic aspects of tree-width, Journal of Algorithms 7 (1986), 309–322.

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

[31] B. Yang, D. Dyer, and B. Alspach, Sweeping graphs with large clique number, Proc 5th International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Computer Science, Vol. 3341, Springer, Hong Kong, China, 2004, pp. 908–920.