Network Decontamination under m-Immunity[1]

Paola Flocchini[a], Fabrizio Luccio[b], Linda Pagli[b], Nicola Santoro[c]

[a] *University of Ottawa, Canada. Email: flocchin@site.uottawa.ca*
[b] *Università di Pisa, Italy. Email: {luccio,pagli}@di.unipi.it*
[c] **corresponding author**. *School of Computer Science, Carleton University, K1S 5B6 Ottawa, Canada. Phone: 1-613-520-4333. Fax:1-613-520-4336. Email: santoro@scs.carleton.ca*

Abstract

We consider the problem of decontaminating an infected network using as few mobile cleaning agents as possible and avoiding recontamination. After a cleaning agent has left a vertex v, this vertex will become recontaminated if m or more of its neighbours are infected, where $m \geq 1$ is a threshold parameter of the system indicating the local immunity level of the network. This *network decontamination* problem, also called *monotone connected graph search* and *intruder capture*, has been extensively studied in the literature when $m = 1$ (no immunity).

In this paper, we extend these investigations and consider for the first time the network decontamination problem when the parameter m is an arbitrary integer value $m \geq 1$. We direct our study to widely used interconnection networks, namely meshes, tori, and trees. For each of these classes of networks, we present decontamination algorithms with threshold m; these algorithms work even in asynchronous setting, either directly or with a simple modification requiring one additional agent. We also establish general lower bounds on the number of agents necessary for decontamination with immunity m; these bounds are tight in the case of trees, while large gaps still exists in the case of meshes and tori.

Keywords:. network decontamination, connected graph search, monotone algorithms, immunity, meshes, tori, trees.

1. Introduction

1.1. The Framework

Parallel and distributed computing systems are designed around interconnection networks. As the size, the complexity, and the importance of a system increases, the presence of malicious threats cannot be avoided. Such threats

[1] Some of these results have been presented at the 23rd IEEE International Parallel and Distributed Processing Symposium and at the 8th International Conference on Algorithms and Complexity.

may be brought by *intruders* that travel through the network and infect any visited site, as for example a virus. The focus of this paper is on counteracting such a threat by a team of mobile *cleaning agents* (or simply *agents*) that traverse the network decontaminating the visited sites.

The team of mobile agents enter the network, viewed as a simple undirected graph, at a single vertex, called *homebase*. An agent located at a vertex v can move to any of the neighbouring nodes of v, decontaminating it with its presence; upon departure of the (last) agent, a vertex can become re-contaminated if a sufficient number of its neighbours are contaminated. The goal is to decontaminate the whole network using as small a team of cleaning agents as possible avoiding any recontamination. This *network decontamination* problem, known also as *monotone connected graph search* and as *intruder capture*, has been extensively studied in the literature (e.g., see [1, 2, 6, 11, 12, 13, 17, 19, 24, 25, 28, 33]).

The re-contamination process is regulated by a parameter m: if no agent is there, a decontaminated vertex v will become re-contaminated if m or more of its neighbours are infected. The parameter m indicates the local *immunity* level of the network. The number of agents necessary to decontaminate the entire network is clearly a function of m and of the basic parameters of the given network.

In the literature the problem has generally been studied only when $m = 1$, that is the presence of a single infected neighbour recontaminates a disinfected node with no agents. This assumption corresponds to a system without any immunity, and the problem has been investigated under such assumption for a variety of network classes, including trees, hypercubes, meshes, tori, outerplanar graphs, chordal graphs, etc. (e.g., see [3, 9, 11, 12, 13, 17, 28])

The assumption $m = 1$ is quite restrictive. In fact, to enhance reliability, many systems employ *threshold* rules at each site, for example performing voting among various copies of crucial data between neighbours at each step [31]. Indeed, threshold schemes are used for consistency resolution protocols in distributed database management; data consistency protocols in quorum systems; mutual exclusion algorithms; key distribution in security; reconfiguration under catastrophic faults in system level analysis; and computational models in discrete–time dynamical systems. This leads to consider systems with a higher level of resistance to viral threats, where a vertex can be recontaminated after an agent has left only if a threshold $m > 1$ of its neighbours are infected. In turn, this opens the research investigation of the general decontamination problem when the threshold indicating the level of *local immunity* is a global parameter $m \geq 1$. This is precisely the question we address in this paper.

1.2. Main Contributions

We investigate the decontamination problem for arbitrary $m \geq 1$. We focus on three common classes of interconnection networks: *Meshes*, *Tori*, and *Trees*; for each network G in those classes, we establish bounds on the number of agents necessary to decontaminate G with threshold m. The upper bound proofs are constructive, and some of our decontamination protocols are shown to be optimal. More precisely:

- We first consider d-dimensional *meshes* with $N = n_1 \times n_2 \times \cdots \times n_d$ vertices, where $d \geq 2$; w.l.g., let $2 \leq n_1 \leq n_2 \leq \cdots \leq n_d$. We prove that for each such mesh one agent suffices for $m \geq d$, and $n_1 \times n_2 \times ... \times n_{d-m}$ agent suffice for $1 \leq m < d$ by exhibiting a solution algorithm that uses these many agents. We also establish a general lower bound, and show it is tight for $m = d - 1$.

- We then extend the analysis to d-dimensional *toroidal meshes* M, with $N = n_1 \times n_2 \times \cdots \times n_d$ vertices, where $d \geq 2$; w.l.g., let $2 \leq n_1 \leq n_2 \leq \cdots \leq n_d$. We prove that $2^m \times n_1 \times n_2 \times ... \times n_{d-m}$ agents suffice for $1 \leq m \leq d - 1$, and 2^{2d-m} agents suffice for $d \leq m \leq 2d$.

  Note that with growing m more infected neighbors must be present to recontaminate a vertex, so the number of agents must decrease if m grows. This is immediately clear in our protocol for $d \leq m \leq 2d$. For $1 \leq m \leq d - 1$ the number of agents grows with the term 2^m but decreases more rapidly with the number of n_i present in the bound, under the obvious hypothesis that all n_i are greater than 2.

  We also establish a general lower bound, and show it is tight for some values of d and m.

- Finally we consider the family of *trees*. Unlike the case of meshes and tori, the number of needed cleaning agents may be different for different trees of the same size. For every tree and any value of $m \geq 1$ we determine a lower bound to the number of agents, and we then prove that this number is also sufficient by presenting a simple decontamination protocol using precisely those many agents.

The algorithms for meshes and tori are established in a quasi-synchronous setting: agents operate in synchronized steps, but operations within a step (e.g., movements) are not necessarily synchronized. The algorithms can be extended to completely asynchronous settings with simple modifications to their structure and the use of one single extra agent, as in [11, 12, 13]. The proposed algorithms for trees work directly in asynchronous settings.

Although not the main concern of this paper, we also consider the number of moves performed by an optimal-size team of agents as a function of the parameter m. We prove that, for all three families of graphs, all the solution protocols we have presented are optimal, in order of magnitude, with respect to the number of moves.

The paper is organized as follows. In Section 2, we introduce the model and basic properties. In Sections 3, 4, and 5 we present decontamination algorithms for meshes, toroidal meshes, and trees, with the upper and lower bounds on the numbers of agents and moves. In the concluding Section 6 we discuss extension of our studies.

1.3. Previous Work

The network decontamination problem was surprisingly introduced in speleology [7], and then has been extensively studied in the field of *graph searching*

(e.g., see [4, 5, 26]). The problem also arises in graph pebbling [22], as a pursuit-evasion game [27, 30], and in VLSI design [21]. An important aspect of these studies is that there are always *monotone* solutions that use a minimum number of agents, that is protocols that avoid recontamination of vertices after they have been decontaminated [5, 23]. Although formally very similar to ours, all these investigations assume that the decontaminating agents are able to *jump* from one vertex to any other vertex in one step. This marks a first difference with our mode of operation, called *contiguous search*, where the agents can only move from a node to a neighbours in the graph. In fact, the removal of the jumping assumption makes the previous solutions no longer valid [3] .

Contiguous search was first proposed in [2], where optimal monotone strategies were shown for trees. The investigation has then been directed to the monotone decontamination of specific classes of networks, in particular *Trees* [3], *weighted Trees* [9, 10], *Hypercubes* [11], *Meshes* [13], *Pyramids* [33], *Chordal Rings* [12], *Tori* [12], *outerplanar graphs* [17], *chordal graphs* [28], *Sierpinski graphs* [25], *Star graphs* [20], *product graphs* [19], graphs with large clique number [34], while the study of arbitrary graphs has been started in [6] and [18]. While knowledge of the network is generally assumed, some results are also known when only limited amount of information is available or can be obtained [3, 18, 29]. In all these studies it is assumed that, in the absence of a cleaning agent, a decontaminated vertex without agents becomes re-contaminated if $m = 1$ of its neighbours is contaminated.

The case when re-contamination, to occur, requires the presence of more than one contaminated neighbour is referred to as *local immunity*. The only study of network decontamination with local immunity was given in [24] for meshes and trees. In that paper a vertex can be recontaminated after an agent is gone if the majority of its neighbors are infected, while here recontamination is ruled by a global parameter m that applies to all vertices.

A different type of immunity has been defined in the case of *synchronous* networks; when an agent leaves a vertex, that vertex is immune from recontamination for $t \geq 0$ additional time units, regardless of the number of contaminated neighbours [14]. This type of *temporal immunity* has been studied for tree networks [14] and meshes and tori [8].

2. Model and Basic Properties

The network is represented as a simple connected undirected graph G with N vertices. For a vertex v in G, let $\delta(v)$ denote its degree, and let $\delta(G)$ denote the maximum value of $\delta(v)$ among all the vertices of G.

A team of *agents* with distinct Ids operates in G. The agents have their own constant memory, communicate with each other when at the same node, move from node to neighboring node, and execute the same protocol. Distinct Ids and face-to-face communication allow the agents to coordinate their activities and to assign different roles and tasks as and when required by the protocol.

Initially, all vertices are *infected* and the agents enter the network at a single vertex, called *homebase*. The presence of one or more agents at a vertex decon-

taminates that vertex making it *clean*. We say that at a given time a vertex is *grey* if infected, *black* if it is clean and it contains one or more agents, and *white* if it is clean but no agent is there.

A white vertex is re-contaminated (i.e., it becomes grey) if m or more of its neighbors are grey. Notice that, by definition, a white vertex v with $\delta(v) < m$ can never be re-contaminated. When an agent moves from u to v on edge (u, v), it protects u from possible contamination by v. By definition, a black vertex does not become grey regardless of the color of its neighbors. A system so defined is said to have an immunity threshold m to recontamination, or simply to have *m-immunity*.

The task of the team of agents is to decontaminate G avoiding any recontamination. The goals are (1) to determine the *smallest team size* for which such a task can be achieved; and (2) to devise an *optimal monotone strategy*, that is a solution protocol that allows such a minimal team to decontaminate G without recontamination. Note that, by definition, for any solution algorithm, at any time during its execution, all the clean vertices form a connected subgraph. This problem is that of monotone connected graph search, where (immunity to) re-contamination is controlled by the parameter m. A secondary purpose is for the minimal team to be able to perform decontamination using as few moves as possible.

Our main results, to be discussed in the following sections, are about decontamination of the most commonly used interconnection networks, namely *meshes*, *toroidal meshes* (or *tori*), and arbitrary *trees*. Each family will be characterized by proper parameters concerning the shape and size of its members, and the respective algorithms will be designed in function of these parameters and of m. Note that the homebase depends on the family under investigation. As usual in these studies the homebase for meshes is a corner: a different vertex would require different algorithms and yield different results. For toroidal meshes, taking any vertex as homebase would produce the same results due to the complete symmetry of the network. In the case of trees, instead, the homebase is taken as a parameter of the problem.

Let $\mathcal{A}(G, m)$ (resp. $\mathcal{M}(G, m)$) denote the minimum number of agents (resp. the number of moves) needed to decontaminate network G with m-immunity; when no ambiguity arises, we will omit the indication of G.

For decontamination with m-immunity the following bounds hold for any network:

Proposition 1. *Let G be a simple connected graph. We have:*
(i) $\mathcal{A}(m) = 1$ *if* $m \geq \delta(G)$*;*
(ii) $\mathcal{A}(m) \leq 2$ *if* $m = \delta(G) - 1$.

Proof. (i) By definition, a white vertex v with $\delta(v) \leq m$ can never be re-contaminated; hence a single agent traversing the network will perform the decontamination. (ii) This is easily achieved, e.g., having an agent stay at the homebase while the other performs a traversal of the network. □

Of course Proposition 1 does not rule out the possibility that a single agent can decontaminate the graph also for $m = \delta(G) - 1$ or even smaller values of m (this will happen in meshes). The proof indicates that the decontaminating algorithm is monotone no matter which path the agent follows in its traversal; as for the number of moves, if the graph G contains a Hamiltonian path π, as it happens in meshes and tori, we have $\mathcal{M}(G, m) = N - 1$ for $m \geq \delta(G) - 1$, where N is the number of nodes. If a Hamiltonian path does not exist, as in arbitrary trees, these values are higher. Due to Proposition 1 only the case $m < \delta(G) - 1$ is really of interest and will be thoroughly studied.

In *synchronous* systems the agents operate in synchronized *steps*: in each step all the agents communicate (with those at the same vertex), compute, and move (when required). We actually consider the less powerful *quasi-synchronous* systems, where all operations started in step i are completed by step $i + 1$, but the operations within a step might not be instantaneous. In particular, if some (or all) agents move at step i no assumption is made on when they reach the target vertex provided they are all in their next destination before step $i + 1$ begins. For example let $m = 1$ (that is, one infected neighbour suffices for contaminating any vertex) and consider the two elementary examples of Figure 1. Case (a) is immediate. In case (b), when agent a_1 leaves from v_3 this vertex may be re-contaminated by v_4 during the same step if agent a_2 has not yet reached v_4. Symmetrically when agent a_2 leaves from v_2 this vertex may be re-contaminated.

In a *asynchronous* system there is no common notion of time or step, no assumptions exist on synchronization of the agent's actions and movements, and every operation by each agent takes a finite but otherwise unpredictable amount of time.

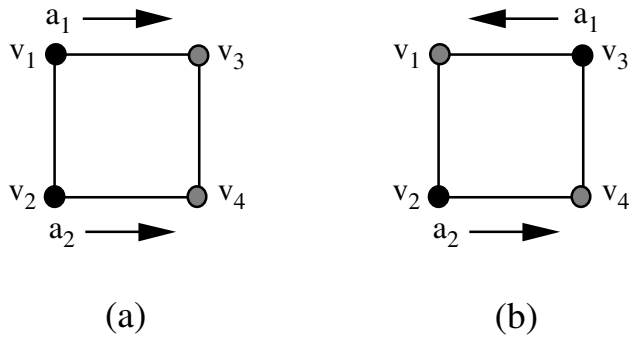

Figure 1: Let $m = 1$. At a given step agents a_1,a_2 (black dots) move as indicated. After the step all vertices become decontaminated in graph (a), while vertices v_2,v_3 may be re-contaminated in graph (b).

Our protocols for trees work in any environment, including asynchronous ones. Our algorithms for meshes and tori operate in quasi-synchronous systems; in some cases they operate in asynchronous systems as well. If not, the following

technique allows extending the algorithms to handle the asynchronous case with the addition of one agent, as observed in [11, 12, 13]:

- If a set of agents decontaminates a network working synchronously, then the same moves can be done asynchronously provided that, between any wave of parallel moves, one additional agent travels through the clean vertices of the network to inform all the other agents that the current wave of moves is completed (recall that the clean vertices form a connected subgraph).

In other words, using this mechanism, the synchronous solutions can operate also in asynchronous environment with the use of a single additional agent.

The protocols we present are described from a global point of view; their operations are however fully distributed. Since the agents have distinct IDs, are aware of the topology, and initially they are all located at the same node, each agent will initially be assigned a specific role; this role allows the agent to locally determine the sequence of movements it has to perform according to the protocol. In the case of meshes and tori, the protocols are described assuming a quasi-synchronous system; in such a system, each agent can locally compute also the timing (in terms of number of steps) of each of its moves. If the mesh/torus is asynchronous, the timing would be given by the additional distinct agent whose task is to synchronize the operations.

3. Decontaminating Meshes

A *d-dimensional mesh* Z (*d-mesh* for short) is a network of $N = n_1 \times n_2 \times \cdots \times n_d$ vertices, with $d \geq 2$ and $2 \leq n_1 \leq n_2 \leq \cdots \leq n_d$. Each vertex $v_{i_1,i_2,\ldots,i_d}$, $0 \leq i_j \leq n_j - 1$, is connected to the $2d$ vertices $v_{i_1-1,i_2,\ldots,i_d}$, $v_{i_1+1,i_2,\ldots,i_d}$, $v_{i_1,i_2-1,\ldots,i_d}$, $v_{i_1,i_2+1,\ldots,i_d}$, ..., $v_{i_1,i_2,\ldots,i_d-1}$, $v_{i_1,i_2,\ldots,i_d+1}$, whenever these indices stay inside the closed intervals $[0, n_j - 1]$ (i.e., any border vertex $v_{i_1,\ldots,i_j=0,\ldots,i_d}$, or $v_{i_1,\ldots,i_j=n_j-1,\ldots,i_d}$, cannot be respectively connected to $v_{i_1,\ldots,i_j-1,\ldots,i_d}$, or to $v_{i_1,\ldots,i_j+1,\ldots,i_d}$).

3.1. Upper Bound for Meshes

We present a protocol ClearMesh, for decontaminating a synchronous d-mesh with m-immunity and analyze its complexity. The decontamination process consists of two phases. Phase 1 amounts to defining an initial set C of vertices and dispatching $|C|$ agents into these vertices. In Phase 2, these agents move from C to clean the whole mesh. The size of C depends on the values of m and d. Taking $v_{0,0,\ldots,0}$ as the homebase, we pose:

Definition 1. *The* initial set C *of a d-mesh contains:* (i) *the only vertex* $v_{0,0,\ldots,0}$*, for* $m \geq d$*;* (ii) *all the vertices with indices* $i_d, i_{d-1}, ..., i_{d-m+1}$ *equal to zero, for* $1 \leq m < d$*. In case* (ii) *we have* $|C| = n_1 \times n_2 \times ... \times n_{d-m}$.

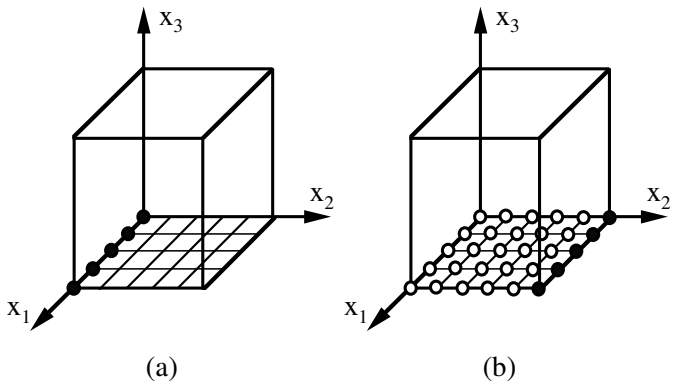

Figure 2: (a) A 3-mesh with n_1 agents placed on C, for $m = 2$. (b) Decontamination of the plane $i_3 = 0$. All the vertices not explicitly shown are gray.

In the 3-mesh of Figure 2(a), letting $m = d - 1 = 2$, C consists of the n_1 vertices with indices $i_3 = i_2 = 0$. At the end of phase 1 C contains one agent per vertex and is built in such a way that no white vertices are created during the process, to avoid possible re-contamination. This is achieved by moving the agents inside C only. Several agents are placed in the same vertex v at certain steps, one of which remains in v as its final destination while the others move away.

Algorithm INITIAL-SET of Figure 3 gives a general scheme of phase 1 for $m \leq d$. In fact, for $m = d$ the initial set consists of vertex $v_{0,0,,0}$ only. This algorithm will also apply to toroidal meshes of the next section 4. A key point is that each agent reaches its final destination through a shortest path from $v_{0,0,,0}$. The **move** operation of the last statement is done synchronously for all the agents involved. We have:

Lemma 1. *For a d-mesh with* $1 \leq m < d$, INITIAL-SET *is a monotone algorithm that places* $n_1 \times n_2 \times ... \times n_{d-m}$ *synchronous agents in the vertices of* C *in less than* $\frac{1}{2}(d-m)\, n_{d-m}^{d-m+1}$ *moves.*

Proof. The number of agents equals the cardinality of C. For proving monotonicity note that only the vertices of C are eventually reached by an agent, and that one agent remains in any such a vertex up the the algorithm termination. For computing the number of moves, note that each agent starts from the homebase and reaches its final destination through a shortest path, then the total number of moves equals the sum of the lengths of all such paths. Since in the homebase all the indices are equal to zero and the length of the paths is computed in Manhattan metric, the number of moves in turn equals the sum of the values of the indices of all the vertexes of C. Letting $j = d - m$ and $\Delta_j = \frac{1}{2} n_j (n_j - 1)$, the number of moves is: $\Delta_1 n_2 n_3 ... n_j + \Delta_2 n_1 n_3 ... n_j + ... + \Delta_j n_1 n_2 ... n_{j-1} = \frac{1}{2}(n_1^2 n_2 ... n_j + n_1 n_2^2 ... n_j + ... + n_1 n_2 ... n_j^2 - j(n_1 n_2 ... n_j)) \leq \frac{1}{2} j (n_j^{j+1} - n_1^j)$, where the equality holds for $n_1 = n_2 = ... = n_j$, and the bound

algorithm INITIAL-SET(d,m)

if (Z is a d-mesh with $m \leq d$)
  let C be the submesh of Z composed of all the vertices with indices $i_d, i_{d-1}, ..., i_{d-m+1}$ equal to zero;
if (Z is a d-tomesh)
  if ($m \geq d$) **let** C be the ($2d$-m)-cube with base $v_{0,0,...,0}$
  else let C be the (d-m)-d-brick with base $v_{0,0,...,0}$;
forany $v \in C$ choose a shortest path from $v_{0,0,...,0}$ to v;
let P be the set of such paths for all the vertices of C;
start with a set of $|C|$ agents in $v_{0,0,...,0}$;
while $\exists\, v \in C$ containing more than one agent:
  let (v, w_1), ..., (v, w_r) be the edges from v included in a path in P;
  let p_i be the number of paths in P containing (v, w_i);
  keep one agent in v (its final destination);
  for $i = 1$ **to** r **move** p_i agents to w_i through edge (v, w_i).

Figure 3: Decontamination phase 1 for a d-mesh or a d-toroidal-mesh Z.

follows. □

By Lemma 1 the number of moves to put the agents in C is $O(dN)$, where the upper bound is attained for $m = 1$. If d is taken as a constant, this value does not exceed, in order of magnitude, the lower bound of $N - 1$ necessary to visit the whole mesh starting from the homebase.

In phase 2 of the decontamination process, the agents move from the vertices of C until the whole d-mesh has been cleaned. This is attained by algorithm MESH given in Figure 4. Let us first explain informally how it works for $d = 3$ (Figure 2), with $m = 3$ or $m = 2$. For $m = 3$, C contains only the homebase $v_{0,0,0}$ and one agent suffices. The agent travels along the x_1 axis to reach vertex $v_{n_1-1,0,0}$. Note that none of the vertices traversed by the agents can be recontaminated, because vertex $v_{0,0,0}$ has degree 3 and 1 clean neighbor, and vertices $v_{1,0,0,0}, v_{2,0,0,0}, ..., v_{n_1-2,0,0}$ have degree 4 and 2 clean neighbors. Now the agent makes one step in the x_2 direction to $v_{n_1-1,1,0}$ and then travels for decreasing values of the index x_1 through $v_{n_1-2,1,0}$, $v_{n_1-3,1,0}$,..., to $v_{0,1,0}$. The vertices thus traversed have degree 5 and 3 clean neighbors, so they also stay clean. The agent continues its journey through $v_{0,2,0}$ to $v_{n_1-1,2,0}$, then $v_{n_1-1,3,0}$ to $v_{0,3,0}$, etc., until the whole bottom plane $i_3 = 0$ is decontaminated. Then the agent makes one step in the x_3 direction to decontaminate the plane $i_3 = 1$, and proceeds plane by plane for increasing value of i_3 until the whole mesh is cleaned. During this process the internal vertices of the mesh have degree 6 and 4 clean neighbors (including the one below) so they also stay clean. This

algorithm MESH(d,m)

let each vertex of C contain an agent;
if $(m = d)$ **move** the agent in $v_{0,0,...,0}$ along the canonical path in Z;
if $(m < d)$
{ **foreach** vertex $v = v_{\bar{i}_1,...,\bar{i}_{d-m},0,...,0}$ of C
{ **let** Z_v be the m-submesh composed of all the vertices with
$i_1 = \bar{i}_1, ..., i_{d-m} = \bar{i}_{d-m}$;
let π_v be the canonical path in Z_v
(*the indices* $i_d, i_{d-1}, ..., i_{d-m+1}$ *are updated in nested cycles, from* i_{d-m+1} *in the inner cycle to* i_d *in the outer cycle*) };
move all the agents v in synchronous steps along their π_v. }

Figure 4: Decontamination phase 2 for a d-mesh Z, with $m \leq d$.

Hamiltonian path followed by the agent will be called *canonical*. Note that it can be obviously extended to any number of dimensions.

For $m = 2$, after the agents have been placed in the vertices of C by INITIAL-SET (Figure 2(a)), they move in the bottom plane $i_3 = 0$ in synchronous waves, for increasing values of the vertex index i_2, until they reach the right-hand side of the plane (i_2=$n_2 - 1$). During this transfer each vertex of the bottom plane is in contact with only one gray neighbor, in fact the one with $i_3 = 1$, and is not re-contaminated; so the bottom plane stays clean (Figure 2(b)). Now all the agents go up synchronously of one step, to the vertices with i_2=$n_2 - 1$ and $i_3 = 1$, and then move as before along the plane i_3=1, from right to left, until they reach the left-hand side of the plane. Again all the vertices of the plane have been exposed to the only gray neighbors above them. The process goes on similarly, plane by plane for increasing value of i_3, until the whole mesh is cleaned.

In general algorithm MESH is based on the following principles. For $m = d$ one agent can decontaminate the whole network traveling along a canonical path, as already discussed for the example of Figure 2. As the same strategy obviously applies for any $m > d$, the algorithm is defined for $1 \leq m \leq d$. For $m < d$ consider a generic vertex v of the initial set C, and the agent a placed in v by the algorithm INITIAL-SET. This agent decontaminates a specific m-submesh Z_v consisting of all the vertices with the $d - m$ indices $i_{d-m}, ..., i_1$ with the same values as the ones of v, and the other indices $i_d, i_{d-1}, ..., i_{d-m+1}$ assuming all combinations of values. In the example of Figure 2(a), the agent in $v_{2,0,...,0}$ will decontaminate the 2-submesh containing all vertices with $i_1 = 2$, that can be represented as a vertical face parallel to the plane x_2, x_3. For this purpose, agent a travels in Z_v along the Hamiltonian path π_v obtained spanning on all the values of the indices $i_d, i_{d-1}, ..., i_{d-m+1}$ in nested cycles, changing i_{d-m+1} first, then changing i_{d-m+2}, etc. up to i_d. Note that π_v is the canonical path for Z_v. It is crucial that all the agents of C move in synchronous

waves along identical paths in their m-submeshes Z_v for protecting the vertices just left. It is worth noting that, at each wave, the agents occupy a new (d-m)-submesh whose vertices have the same values for m indices. In fact they are the indices with value 0 in C, one of which changes at each wave. We have:

Lemma 2. MESH *is a monotone decontamination algorithm with* $|C|$ *synchronous agents starting in* C. *For* $m = d$ *the single agent makes* $N - 1$ *moves. For* $1 \leq m < d$ *the* $n_1 \times n_2 \times ... \times n_{d-m}$ *agents make* $N - n_1 \times n_2 \times ... \times n_{d-m}$ *moves.*

Proof. For proving complete decontamination and monotonicity note that all the vertices of Z are visited by an agent and, in all the **move** operations, each vertex v left by an agent has at least $\delta(v) - m + 1$ clean neighbors. For computing the number of moves note that all the vertices of the mesh outside C are traversed exactly once by one agent, so they are all cleaned in $N - |C|$ moves. □

Combining Lemmas 1 and 2 we immediately have:

Theorem 1. *Protocol* CLEARMESH *monotonically decontaminates a* d*-mesh in* $\mathtt{M}(m) = \Theta(N)$ *moves by a team of* $\mathtt{A}(m)$ *synchronous agents, where*

$\mathtt{A}(m) = 1$, *for* $m \geq d$;

$\mathtt{A}(m) = n_1 \times n_2 \times ... \times n_{d-m}$, *for* $1 \leq m < d$.

3.2. Lower Bound for Meshes

By definition, at any step of a monotone decontamination protocol of a graph G, all the clean vertices form a connected subgraph B that includes the homebase; B, also said to be *clean*, is called a *guarded block* and constitutes the key concept for a lower bound argument. Let $\alpha(B)$ denote the number of vertices of B with at least m gray neighbors. Since each of these vertices must contain an agent, $\alpha(B)$ gives the minimum number of agents needed by that protocol to protect B from re-contamination.

Let $\mathcal{P}$ be the set of all monotone decontamination algorithms for a d-mesh Z with home base $v_{0,0,...,0}$. Given algorithm $P \in \mathcal{P}$, each execution in Z generates a sequence $S(P) = B_0, B_1, ..., B_{k(P)}$ of guarded blocks, where $B_i \subset B_{i+1}$, B_0 contains the homebase only, and $B_{k(P)} = Z$. Let $\alpha(S(P)) = Max_i\{\alpha(B_i)\}$; then the number of agents used by P is obviously greater than or equal to $\alpha(S(P))$.

Given that one agent is sufficient to decontaminate Z with $m \geq d$, lower bounds on $\mathcal{A}(m)$ must be studied for $1 \leq m < d$.

Theorem 2. *For a* d*-mesh with* $1 \leq m \leq d - 1$ *we have* $\mathcal{A}(m) \geq n_1 + n_2 + \cdots + n_{d-m} - (d - m - 1)$.

Proof. Given $P \in \mathcal{P}$, let $S(P) = B_0, B_1, ..., B_{k(P)}$, and let t be the smallest index such that $B_t \in S(P)$ contains a vertex at the extreme of the mesh in at least $d - m$ directions; that is, there is a permutation $x_{i_1}, ..., x_{i_d}$ of $x_1, ..., x_d$, such that for each coordinate i_j, $1 \leq j \leq d - m$, at least one vertex of B_t

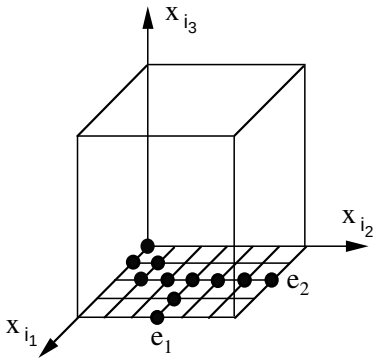

Figure 5: Paths π_1, π_2 in a 3-mesh with $m = 1$. We have $n_{i_1} = 5, n_{i_2} = 7$ and any value of n_{i_3}, hence $\nu = 5 + 7 - 1 = 11$. In this case the paths lie in the subspace x_{i_1}, x_{i_2}.

has coordinate $i_j = n_{i_j} - 1$. Let $e_1, \ldots, e_{d-m}$ be such extreme vertices (or one of them for each coordinate, if there are more than one). Since B_t is connected and contains the home base, B_t must contain a path π_i from $v_{0,0,\ldots,0}$ to each of the e_i. The total number of vertices in these paths must be $\nu \geq 1 + (n_{i_1} - 1) + (n_{i_2} - 1) + \cdots + (n_{i_{d-m}} - 1) = n_{i_1} + n_{i_2} + \cdots + n_{i_{d-m}} - (d - m - 1)$ (see Figure 5).

Note that for each vertex v in each π_i there may be other vertices in B_t with the same values of i_j, $1 \leq j \leq d - m$, and different values of the other coordinates. Among these vertices consider the ones with maximum value of i_{d-m+1}; then, among them, the ones with maximum value of i_{d-m+2}, etc., up to the single vertex w with maximum value of i_d.

Either vertex w was contained in B_{t-1}, then it has at least one gray neighbor in each of the m dimensions $i_j > d - m$. Or w has been inserted in B_t for the first time. In both cases, w must contain an agent, hence B_t must contain at least ν agents.

The minimum value of ν holds for a block B_t consisting only of the vertices lying on the paths π_i, along the coordinate axes $x_{i_1}, \ldots, x_{i_{d-m}}$. This value is minimized for the permutation x_{i_1},..., $x_{i_d} = x_1$,..., x_d, proving that $n_1 + n_2 + \cdots + n_{d-m} - (d - m - 1) \leq \alpha(B_t) \leq \alpha(S(P))$. Since this holds for every $P \in \mathcal{P}$, the theorem follows. □

As a consequence, by Theorem 1, our algorithm ClearMesh is optimal for $m = d - 1$. Notice that the proof of Theorem 2 holds even if the executions sequences $S(P)$ are restricted to be the ones occurring in fully synchronous settings; hence the lower bound holds under all possible timing and synchronization constraints. As for the number of moves, we simply note that the obvious lower bound of $N - 1$ on $\mathcal{M}(m)$ matches the upper bound of Theorem 1 in order of magnitude.

For $m < d - 1$ there is a gap between the upper and the lower bound

of Theorems 1 and 2, that increases for decreasing values of m and becomes very large if m is small. Possibly the lower bound could be greatly improved, to approach the upper bound of Theorem 1. Consider the guarded blocks $B_0, B_1, ...$ in any decontamination process. For $1 \leq m \leq d-2$ let B_i be the first guarded block contained in a subspace of $r = d - m$ dimensions, that is $m = d - r$. Since each vertex of B_i has at least one gray neighbor in each of the other $d - r$ dimensions, all the vertices of B_i must contain an agent. If the guarded blocks following B_i in the decontaminating sequence keep growing in the same subspace, their size tends to the product of the corresponding $r = d - m$ values n_i, and the lower bound tends to $n_1 \times n_2 \times ... \times n_{d-m}$. However the guarded blocks may invade subspaces with $r > d - m$ dimensions before the above situation is met, making the lower bound evaluation much more difficult. If B_j is one such block, only the vertices on the frontier F of B_j may need a protecting agent, where F is a "surface" of dimension $r - 1$ whose number of vertices also tends to the product of $r - 1 \geq d - m$ values n_i.

4. Decontaminating Toroidal Meshes

As in the definition of simple meshes (see previous section), a *d-dimensional toroidal mesh* Z (*d-tomesh* for short) is a network with $N = n_1 \times n_2 \times \cdots \times n_d$ vertices, $2 \leq n_1 \leq n_2 \leq \cdots \leq n_d$. Each vertex $v_{i_1,i_2,...,i_d}$, $0 \leq i_j \leq n_j - 1$, is connected to the $2d$ vertices $v_{i_1-1,i_2,...,i_d}$, $v_{i_1+1,i_2,...,i_d}$, $v_{i_1,i_2-1,...,i_d}$, $v_{i_1,i_2+1,...,i_d}$, ..., $v_{i_1,i_2,...,i_d-1}$, $v_{i_1,i_2,...,i_d+1}$, where, in the present case, all the operations on the indices i_j are done mod n_j. That is border vertices do not exist and a vertex $v_{i_1,...,i_j=0,...,i_d}$, or $v_{i_1,...,i_j=n_j-1,...,i_d}$, is also connected to $v_{i_1,...,i_j=n_j-1,...,i_d}$, or to $v_{i_1,...,i_j=0,...,i_d}$, respectively. As we shall see this complicates the situation substantially.

As before Z can be built by connecting a number n_d of (d-1)-tomeshes $Z_0, ..., Z_{n_d-1}$, called (d-1)-*submeshes* of Z, with each submesh Z_i adjacent to Z_{i-1}, $Z_{(i+1) mod\, i_d}$. Recursively a d-tomesh contains $n_d \times n_{d-1} \times \cdots \times n_{d-i}$ disjoint (d-i-1)-submeshes, for $0 \leq i \leq d-1$ (for $i = d-1$ each submesh reduces to a single vertex).

4.1. Upper Bound for Toroidal Meshes

As for simple meshes, the decontamination protocol ClearToMesh consists of a Phase 1 where an initial set C of vertices is defined and $|C|$ agents are dispatched into these vertices starting from the homebase $v_{0,0,...,0}$; and a Phase 2 where these agents move from C to clean the whole d-tomesh.

Due to Proposition 1 the basic cases $m = 2d$ and $m = 2d-1$ are elementary. We have $\mathtt{A}(2d) = 1$, $\mathtt{A}(2d-1) = 2$, and $\mathtt{M}(2d) = \mathtt{M}(2d-1) = N - 1$, where these values obviously match the lower bounds. For general values of m and d the situation is more complicated. C may be a *c-cube*, $1 \leq c \leq d$, or a *b-h-brick*, i.e. a set of 2^{h-b} adjacent b-submeshes of Z, $1 \leq b < h \leq d$. See the definitions below where a *base vertex* $\bar{v}$ of the set is also specified. Either in cube or in brick form, the set C contains one agent per vertex at the end of phase 1, and is

built in such a way that no white vertices are created during the process. As for simple meshes each agent reaches its destination vertex in C along a shortest path from the homebase. For a d-tomesh Z and a vertex $\bar{v} = v_{\bar{i_1},\bar{i_2},...,\bar{i_d}}$ we pose:

Definition 2. *A c-*cube with base $\bar{v}$, $1 \leq c \leq d$, *is a set of 2^c vertices contained in Z. In each vertex of the c-cube the indices i_j, $1 \leq j \leq c$, assume the two values $\bar{i_j}$ (as in $\bar{v}$), $\bar{i_j}+1$ in all possible combinations, while the remaining indices (if any) have the same values $\bar{i}_{c+1}, ..., \bar{i_d}$ of $\bar{v}$.*

Note that a c-cube has the same structure of a c-dimensional hypercube.

Definition 3. *A b-h-*brick with base $\bar{v}$, $1 \leq b \leq h \leq d$, *is a set of 2^{h-b} adjacent b-submeshes contained in Z. In each vertex of the b-h-brick the indices i_j, $1 \leq j \leq b$, assume all the values $0, 1, ..., n_j - 1$ in all possible combinations; the indices i_j, $b+1 \leq j \leq h$ (if any), assume the two values $\bar{i_j}$ (as in $\bar{v}$), $\bar{i_j}+1$ in all possible combinations; and the remaining indices (if any) have the same values $\bar{i}_{h+1}, ..., \bar{i_d}$ of $\bar{v}$.*

We immediately have:

Proposition 2. *A c-cube contains 2^c vertices, each adjacent to c other vertices of the cube. A b-h-brick contains $2^{h-b} n_1 n_2 ... n_b$ vertices, each adjacent to $b+h$ other vertices of the brick.*

See Figure 6 for examples. A 3-cube has $2^3 = 8$ vertices each adjacent to other three, and a 2-3-brick has $2n_1n_2$ vertices each adjacent to other five. Note that a b-h-brick with $b = h$ is an h-submesh of Z, with $0 \leq i_j \leq n_j - 1$ for $1 \leq j \leq b$, and the other indices with the same values $\bar{i_j}$ as in $\bar{v}$.

4.1.1. Tomesh decontamination phase 1

The algorithm for phase 1 is INITIAL-SET of Figure 3, already used for meshes, with proper initialization of the set C. Note that C is a cube for $d \leq m \leq 2d$, or a brick for $1 \leq m \leq d-1$. In fact the algorithm does not need to be applied for $m > 2d-2$ since decontamination is trivial in this case. The reasons for the particular choice of C will be clear in the following. Note that no vertex v of the initial set C is left unattended after it has been reached for the first time, i.e. no such a vertex becomes white, because when v is reached for the first time one of the visiting agents is kept in v as its final destination. We have:

Lemma 3. *Algorithm INITIAL-SET uses 2^{2d-m} synchronous agents and requires $M_c = 2^{2d-m-1}(2d-m)$ moves, for $d \leq m \leq 2d$; or uses $2^m n_1 n_2 ... n_{d-m}$ agents and requires $M_b = 2^{m-1} n_1 n_2 ... n_{d-m}(n_1 + n_2 + \cdots + n_{d-m} + 2m - k)$ moves for $1 \leq m \leq d-1$.*

Proof. The number of agents derives immediately from Proposition 2. The number of moves equals the sum of the values of the indices of all the vertices of C. In fact each agent starts in the homebase where all the indices are equal to zero, and reaches its final destination through a shortest path whose length

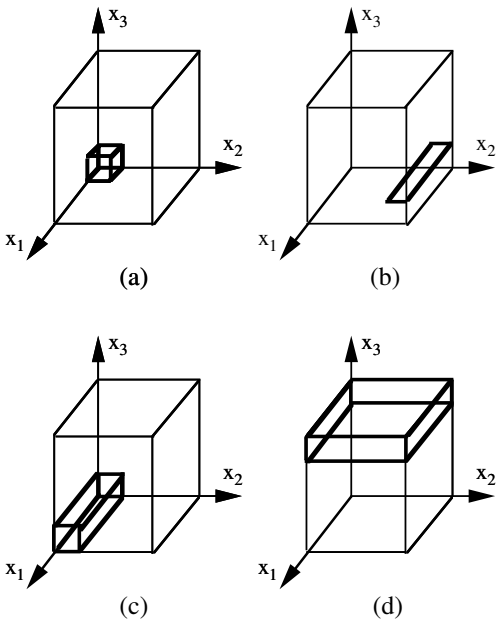

Figure 6: c-cubes and b-h-bricks in a 3-dimensional mesh. (a) 3-cube with base $v_{0,0,0}$. (b) 1-2-brick with base $v_{0,n_2-2,1}$. (c) 1-3-brick with base $v_{0,0,0}$. (d) 2-3-brick with base $v_{0,0,n_3-2}$.

is computed in Manhattan metric. With this in mind, the value of M_c for a cube is computed immediately. For a b-d-brick with $b = d - m$, instead, we have $M_b = M_1 + M_2$, where M_1, M_2 are computed separately as the sum of all the values of the indices i_1 to i_b, and i_{b+1} to i_d, respectively. With some easy computations we obtain: $M_1 = 2^{m-1}n_1n_2...n_b(n_1 + n_2 + \cdots + n_b - b)$, $M_2 = 2^{m-1}n_1n_2...n_b(d - b)$, and the statement follows. □

If C is a cube, the numbers of agents and moves required by algorithm INITIAL-SET are a function of m and d only. If C is a brick, instead, these numbers are also a function of the dimensions of Z. For example, for a 2-3-brick (i.e., $d = 3$ and $m = 1$, then $d - m = 2$) we need $2n_1n_2$ agents and $n_1n_2(n_1 + n_2 - 1)$ moves (see Lemma 3). Note that the number of moves may be quite higher than the number of vertices of the brick (e.g., this number is $2n_1n_2$ for a 2-3-brick), due to the fact that many agents traverse the same vertices.

4.1.2. Tomesh decontamination phase 2

Depending on the value of m, phase 2 of the decontamination process starts with one of the two different initial sets built with algorithm INITIAL-SET. Although the following actions will differ, a common result applies to both cases. We have:

Definition 4. *A b-h-brick B is* clean *if* $b + h \geq 2d - m + 1$ *and all the vertices of B are clean.*

Proposition 3. *A clean brick B cannot be re-contaminated even if the agents present in B (if any) leave from the brick.*

Proposition 3 is immediately proved by recalling that each vertex v of B is adjacent to $b + h$ other vertices in B by Proposition 2, then the neighbors of v lying outside of B are $2d-(b+h) \leq m-1$. If the agents in B move out the brick is left with white vertices only, but still cannot be re-contaminated. Note that a clean brick is a special case of a guarded block as introduced in subsection 3.2.

The general strategy for phase 2 is using a minimum number of agents for building a clean brick. As this brick may now be left unguarded, the agents move to an adjacent brick of the mesh Z and the process is iterated. The following algorithms CUBE and BRICK initially build a clean brick for $d \leq m \leq 2d$ and $1 \leq m \leq d-1$, respectively. The process is then repeated on adjacent bricks of the same size until a ($2d$-m)-submesh S of Z (for CUBE), or the whole mesh Z (for BRICK), is decontaminated. In the former case, for $m = d$ the submesh S coincides with Z while for $m > d$ the tomesh is subdivided into ($2d$-m)-submeshes S_1=S, S_2, ..., S_r, with $r = n_{2d-m+1} \cdot n_{2d-m+2} \cdot ... \cdot n_d$, and S_2, ..., S_r are then decontaminated. The whole algorithm is ITERCUBE reported below. To ease the formulation of all these algorithms we assume that the values of $n_1, ..., n_d$ are even. Odd dimensions only require some obvious corrections to handle border conditions, without affecting the number of agents and affecting only marginally the total number of moves.

Consider the set A of agents brought into the initial vertex set C by algorithm INITIAL-SET. For a vertex v and $x \in \{1, 2, \ldots, d\}$, let $v(x, +1)$ (respectively, $v(x, -1)$) denote the vertex with the same index values of v except for index i_x that is increased (respectively, decreased) by 1 modulo n_x. The following macro-code MOVE(W, x, y) is called for $W \subset A$ and $y \in \{+1, -1\}$. Recall that each **move** operation is performed synchronously for all the agents involved.

macro MOVE(W, x, y)

move each agent of W from its vertex v to $v(x, y)$

Consider now algorithm CUBE of Figure 7, to be applied for $d \leq m \leq 2d-2$ (recall that decontamination is trivial for $m > 2d-2$). Assume that $d \geq 2$. C is a ($2d$-m)-cube with base $v_{0,0,\ldots,0}$, with the initial set A of 2^{2d-m} agents placed in C as done with algorithm INITIAL-SET. For an arbitrary non-empty set $\alpha \subset \{1, 2, ..., k\}$ and for $j \in \alpha$, let $A_j \subset A$ and $\bar{A}_j \subset A$ be the subsets of agents in the vertices with index values $i_j = 1$, and $i_j = 0$, respectively. CUBE is a recursive algorithm working on Z, A, called with input parameter $t = 2d - m$. y is a global variable. We have:

Lemma 4. *Algorithm CUBE called with* $t = 2d-m$ *and* $y = +1$ *decontaminates a* ($2d$-m)*-submesh S of Z leaving the agents in a* ($2d$-m)*-cube.*

Proof. Consider the basic case $t = 2$ (i.e. $m = 2d - 2$). In the cycles for i the agents of A_1 are moved along dimension 1, forward or backward depending on

```
algorithm CUBE(t,y)
if t = 2
    for j = 1 to n_2/2
        { for i = 1 to (n_1 − 2) MOVE(A_1,1,y);
          y = −y;
          if j < n_2/2 { MOVE(Ā_2,2,−1); MOVE(A_2,2,+1); } }
else for h = 1 to n_t/2
        { CUBE(t-1);
          if h < n_t/2 { MOVE(Ā_t,t,−1); MOVE(A_t,t,+1); } }
```

Figure 7: Decontamination of a (2d-m)-submesh of Z, for $d \leq m \leq 2d-2$ and $d \geq 2$.

the value of j. At the end of the cycles for j the agents occupy an (2d-m)-cube C', and have created a 2-(2d-m)-brick B with base $v_{0,0,...,0}$ and all white vertices in $B \backslash C'$. Note that $v_{0,0,...,0} \notin C'$. When the agents of A move along dimensions 3, 4, ..., $2d-m$ at each iteration of the nested cycles for h, all the vertices of B become white and cannot be re-contaminated. □

The submesh S decontaminated by CUBE has vertices with indices $i_j = 0, ..., n_j - 1$ for $1 \leq j \leq 2d-m$, while all the other indices (if any) have value 0. If $m = d$ we have $S = Z$. The functioning of CUBE is illustrated in Figure 8 for a 3-tomesh with $m = 4$. In this simple case we have $t = 2$, so the **else** portion of the algorithm is not executed and the decontaminated submesh is the "plane" (2-2-brick) x_1, x_2. In any case we then invoke algorithm ITERCUBE shown in Figure 9, that works on Z, A, makes use of CUBE, and is called for $d \leq m \leq 2d-2$ with input parameter $s = d$ (if $m = d$ ITERCUBE calls CUBE only once). The tomesh of Figure 8 would be decontaminated by ITERCUBE plane by plane for increasing index i_3, by moving the four agents one position ahead along x_3 at each iteration of the i-cycle.

Based on algorithms INITIAL-SET and ITERCUBE, and including the trivial cases $m = 2d$, $m = 2d-1$, we can state an upper bound on the number of agents and moves.

Lemma 5. *For $d \leq m \leq 2d$, a d-tomesh can be decontaminated by 2^{2d-m} synchronous agents with $N + 2^{2d-m-1}(2d-m-2)$ moves.*

Proof. Consider algorithm ITERCUBE with input $s = d$. Each call CUBE(2d-m) decontaminates a (2d-m)-submesh S of Z as shown in Lemma 4. The agents are then moved away and all vertices of S become white. This submesh is a b-h-brick with $b = h = 2d-m$, that is $b + h > 2d-m$ since $m < 2d$. Hence S is a clean brick (see Definition 4) and cannot be re-contaminated. The process is iterated recursively in the i-cycles, with obvious meaning. The total number of moves is $M_c + M_1$ where the two terms account for the moves required by INITIAL-SET and ITERCUBE, respectively. M_c is given in Lemma 3. M_1

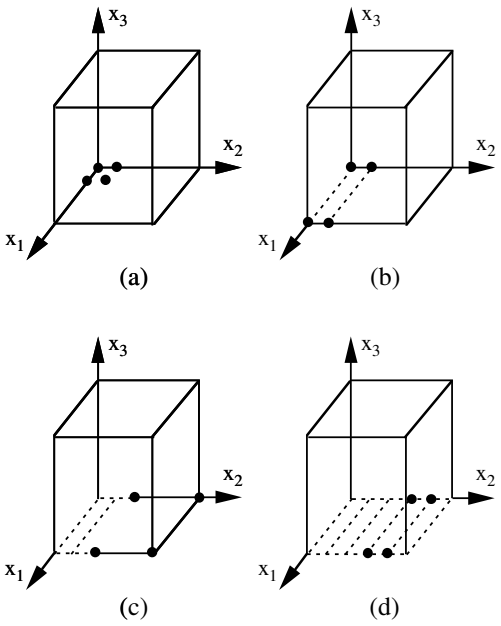

Figure 8: Algorithm CUBE applied to a 3-tomesh, for $m = 4$: agents are black dots, and the white vertices are displayed in dashed lines. (a) The initial 2-cube. (b) White vertices after the first completion of the i-cycle. (c) The configuration after the first moves of $\bar{A}_2$, A_2. (d) The final configuration.

is computed immediately noting that, in each execution of CUBE and in the following executions of MOVE in the cycles for i, agents move only to clean vertices. Hence M_1 equals the number of clean vertices left after the application of INITIAL-SET. Then the moves are $2^{2d-m-1}(2d-m) + N - 2^{2d-m}$. □

Let us now consider the case $1 \leq m \leq d-1$. The initial set C built by algorithm INITIAL-SET is a b-d-brick with $b = d - m$ and base $v_{0,0,\ldots,0}$, occupied by a set A of $2^m n_1 n_2 ... n_b$ agents. Phase 2 is now solved by algorithm BRICK of Figure 10 that works recursively on Z, A and is called with input parameter $t = d$. Its structure, and the following analysis, is similar to the one of CUBE; its functioning is illustrated in Figure 11 for a 3-tomesh with $m = 2$. We have:

Lemma 6. *For $1 \leq m \leq d-1$, a d-tomesh can be decontaminated by $2^m n_1 n_2 ... n_b$ synchronous agents, where $b = d - m$, with $N + 2^{m-1} n_1 n_2 ... n_b (n_1 + n_2 + ... + n_b + 2m - d - 2)$ moves.*

Proof. Consider algorithm BRICK with input $t = d$. At the end of the basic i-cycle the agents occupy a b-h-brick C' and have created a $(b+1)$-d-brick B with base $v_{0,0,\ldots,0}$ and all the vertices of $B \backslash C'$ are white. Each vertex of B has $(b+1) + d = 2d - m + 1$ clean neighbors, hence $m - 1$ gray neighbors, and

```
algorithm ITERCUBE(s)
if (s = 2d − m) CUBE(s)
else for i = 1 to n_s
    { ITERCUBE(s − 1); MOVE(A,s,+1); }
```

Figure 9: Phase 2 of a d-tomesh decontamination for $d \leq m \leq 2d-2$.

```
algorithm BRICK(t)
if t = b + 1
    { for i = 1 to (n_{b+1} − 2) MOVE(A_{b+1},b + 1,y);
      y = −y; }
else for j = 1 to n_t/2
        { BRICK(t-1);
          if j < n_t/2
              { MOVE(Ā_t,t,−1); MOVE(A_t,t,+1); } }
```

Figure 10: Phase 2 of a d-tomesh decontamination for $1 \leq m \leq d-1$. The initial set is a b-d-brick with $b = d - m$.

cannot be re-contaminated. The process is iterated recursively in the j-cycles, with obvious meaning.

$\mathcal{M}(m)$ is given by $M_b + M_2$ where the two terms account for the moves required by INITIAL-SET and BRICK, respectively. M_b is given in Lemma 3. M_2 is computed immediately noting that, in each execution of the basic i-cycle, and in the following executions of MOVE in the j-cycles, agents move only to clean vertices. Then the moves are $2^{m-1}n_1n_2...n_b(n_1+n_2+...+n_b+2m-d)+N-2^m n_1n_2...n_b$, and the thesis follows. □

Combining Lemmas 5 and 6, with easy calculations on the number of moves under the reasonable assumption that $d < n_d$, we have:

Theorem 3. *Protocol* ClearToMesh *monotonically decontaminates a d-tomesh with a team of $\mathtt{A}(m)$ synchronous agents in $\mathtt{M}(m)$ moves, where*
$\mathtt{A}(m) = 2^m n_1n_2...n_{d-m}$ *and* $\mathtt{M}(m) < dN$, *for* $1 \leq m \leq d-1$*;*
$\mathtt{A}(m) = 2^{2d-m}$ *and* $\mathtt{M}(m) < 2N$, *for* $d \leq m \leq 2d$.

4.2. Lower Bound for Toroidal Meshes

The study of the lower bound for toroidal meshes is conducted on the same grounds as the one for simple meshes, although all the vertices have the same degree $2d$ and the guarded blocks have a different structure due to the absence of border conditions in the network.

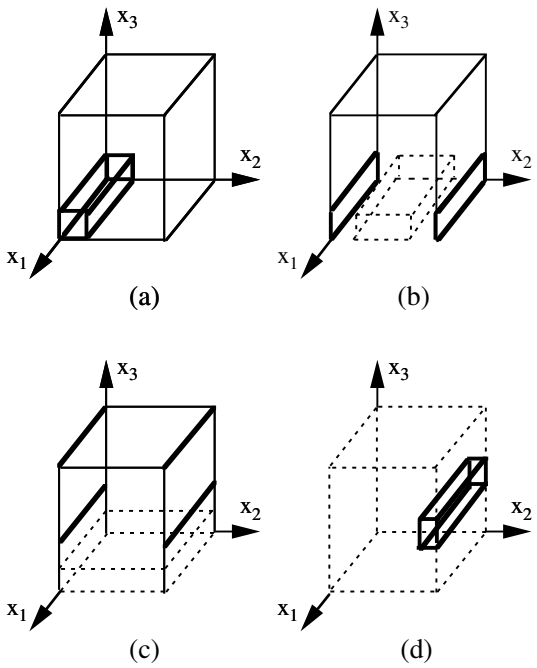

Figure 11: Algorithm BRICK for $d = 3$ and $m = 2$, hence $t = 3$ and $b = 1$: agents and white vertices are displayed in the engrossed and dashed areas, respectively. (a) The initial 1-3-brick. (b) White vertices after the completion of the first i-cycle. (c) The configuration after the first moves of $\bar{A}_3$, A_3. (d) The final configuration.

Let $\mathcal{P}$ be the set of all monotone decontamination algorithms for a d-tomesh Z with home base $v_{0,0,...,0}$. Given algorithm $P \in \mathcal{P}$, each execution in Z generates a sequence $S(P) = B_0, B_1, ..., B_{k(P)}$ of guarded blocks, where $B_i \subset B_{i+1}$, B_0 contains the homebase only, and $B_{k(P)} = Z$. Let $\alpha(S(P)) = Max_i\{\alpha(B_i)\}$; then the number of agents used by P is obviously greater than or equal to $\alpha(S(P))$.

As done for the upper bound, the two cases $1 \leq m \leq d-1$, and $d \leq m \leq 2d$ will be considered separately. For $1 \leq m \leq d-1$ Theorem 2 stated for meshes is still valid because the protection against recontamination offered by the coordinate hyper-planes of the space $x_1, ..., x_n$ is not effective in tori. In fact we must expect stronger bounds in the present case.

Theorem 4. *For a d-tomesh with* $1 \leq m \leq d-1$ *we have* $\mathcal{A}(m) \geq 2(n_1 + n_2 + \cdots + n_{d-m} - (d-m-1))$.

Proof. The proof is an extension of the one of Theorem 2 for meshes, based on an arbitrary permutation $x_{i_1}, ..., x_{i_d}$ of the indices of $x_1, ..., x_d$, and a guarded block B_t where, for each coordinate i_j, $1 \leq j \leq d-m$, there is a path π_{i_j} spanning over all the values $0 \leq i_j \leq n_{i-j} - 1$. As in the proof for meshes, at least $n_{i_1} + n_{i_2} + \cdots + n_{i_{d-m}} - (d-m-1)$ agents are needed to guard B_t, located

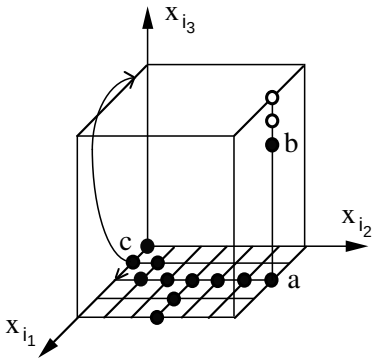

Figure 12: The example of Figure 5 in a toroidal mesh. Two agents a, b are needed at $x_{i_1} = 2$, $x_{i_2} = 6$. If agent c makes a step along any of the coordinate axes a new agent is needed in its previous position.

in vertices with values $\bar{i}_j$, $1 \leq j \leq d - m$ relative to all the paths π_{i_j}.

As more than one vertex with coordinates $\bar{i}_j$ may exist in B_t, in the case of meshes an agent was placed in the vertex with maximum value of i_{d-m+1}, then with maximum value of i_{d-m+2}, etc.. Now, due to the absence of border conditions in tori, at least two agents are needed as shown in Figure 12 (agents a and b) where the paths π_{i_j} are the same as in Figure 5. If only one vertex with coordinates $\bar{i}_j$ exists, only one agent is needed (agent c). However, when in the evolution of the algorithm another vertex with coordinates $\bar{i}_j$ is decontaminated, a second agent will be needed. In particular agent c may move along any of the coordinate axes, and another agent must be placed in the vertex just left by c.

As a conclusion, at least two agents are needed for any of the coordinates of the vertices in all paths π_{i_j}. The total number of vertices in these paths is minimized for the permutation of the indices x_{i_1},..., $x_{i_d} = x_1$,..., x_d, completing the proof. □

For $d \leq m \leq 2d$ we can state:

Theorem 5. *For a d-tomesh with we have* $\mathcal{A}(m) \geq 2^{2d-m}$ *for* $2d-3 \leq m \leq 2d$, *and* $\mathcal{A}(m) \geq 2^3$ *for* $d \leq m \leq 2d-4$.

Proof. The lower bounds for $m = 2d$ and $m = 2d - 1$ (1 and 2 agents) can be immediately proved by inspection, matching the upper bounds of Theorem 3. Furthermore, it can be proved by straightforward exhaustion that 3 (respectively 7) agents are insufficient to decontaminate any d-tomesh for $m = 2d - 2$ (respectively $m = 2d - 3$) because, after a very few steps, the agents could not move any further without causing re-contamination. The lower bound of 2^3 agents trivially holds for the case $d \leq m \leq 2d - 4$. □

Note that, in the case $d \leq m \leq 2d - 4$, an exhaustive proof could provide a better bound for some values of m; clearly this approach becomes exceedingly complicated for growing m.

As a consequence, by Theorem 3, our algorithm `CleanToMesh` is optimal for $2d-3 \leq m \leq 2d$. Notice that the proof of Theorems 4 and 5 hold even if the executions sequences $S(P)$ are restricted to be the ones occurring in fully synchronous settings; hence the lower bound holds under all possible timing and synchronization constraints.

As for the number of moves, we simply note that the obvious lower bound of $N-1$ on $\mathcal{M}(m)$ matches the upper bound of Theorem 3 in order of magnitude for $d \leq m \leq 2d$, and it is inferior by a factor d for $1 \leq m \leq d-1$.

5. Decontaminating Tree Networks

We now study decontamination of an arbitrary unrooted tree T. Observe that, in a tree T, the removal of an edge (u,v) in T creates two rooted subtrees: the subtree rooted in u (and not containing v) which we denote $T(u \setminus v)$, and the one rooted in v (and not containing u) denoted by $T(v \setminus u)$.

In a tree T, a single agent performing a simple traversal starting from a leaf is sufficient to decontaminate T if $m \geq \delta(T)-1$. Hence in the following we will assume $m < \delta(T)-1$

5.1. Lower Bounds for Trees

The minimum number of agents needed to decontaminate a tree T depends on the choice of the homebase, i.e., the node v of T from which the agents start. Let $\mathcal{A}(v,T,m)$ denote the minimum number of agents needed to decontaminate T when v is the homebase; then $\mathcal{A}(T,m) = \min_v\{\mathcal{A}(v,T,m)\}$ denotes the smallest number of agents needed to decontaminate T when the agents can choose the homebase.

We now are going to determine a lower bound on $\mathcal{A}(v,T,m)$. Let $v_1,...,v_d$ be the neighbors of v in T, where $d=\delta(v)$. Without loss of generality, let $\mathcal{A}(v_i,T(v_i \setminus v),m) \geq \mathcal{A}(v_{i+1},T(v_{i+1} \setminus v),m)$ for $1 \leq i < d$.

Theorem 6.
(i) If $d=0$, *then* $\mathcal{A}(v,T,m)=1$.
(ii) If $d>0$, *then* $\mathcal{A}(v,T,m) \geq \mathcal{A}(v_1,T(v_1 \setminus v),m)$.
(iii) If $d > m$ *and* $\mathcal{A}(v_1,T(v_1 \setminus v),m) = \mathcal{A}(v_{m+1},T(v_{m+1} \setminus v),m)$, *then* $\mathcal{A}(v,T,m) \geq \mathcal{A}(v_1,T(v_1 \setminus v),m)+1$.

Proof. Cases (i) and (ii) trivially hold: when $d=0$, T is composed of a single node and $\mathcal{A}(v,T,m)=1$; when $d>0$, by definition, $\mathcal{A}(v_1,T(v_1 \setminus v),m)$ agents are needed to decontaminate the subtree $T(v_1 \setminus v)$ starting from v_1, and, since the homebase is v, because of monotonicity the decontamination of $T(v_1 \setminus v)$ must start from v_1; hence $\mathcal{A}(v,T,m) \geq \mathcal{A}(v_1,T(v_1 \setminus v),m)$. In case (iii), we have $d>m$ and $\mathcal{A}(v_1,T(v_1 \setminus v),m) = \mathcal{A}(v_{m+1},T(v_{m+1} \setminus v),m) = A$. Consider the subtrees $T(v_j \setminus v)$, $1 \leq j \leq m+1$ and, by contradiction let $\mathcal{A}(v,T,m) = \mathcal{A}(v_1,T(v_1 \setminus v),m) = A$; since the decontamination of any of the subtrees $T(v_j \setminus v)$, $1 \leq j \leq m+1$, requires the transfer of A agents to $T(v_j \setminus v)$, and since we have assumed $\mathcal{A}(v,T,m)=A$, then when decontaminating the

first of those subtrees, v is left unprotected while the other m such subtrees are still contaminated; hence, v will become recontaminated. In other words, in case (iii), it must be $\mathcal{A}(v,T,m) > \mathcal{A}(v_1, T(v_1 \setminus v), m)$. □

We now are going to determine a lower bound on $\mathcal{A}(v,T,m)$. Consider the function $\alpha(v,T,m)$ defined recursively as follows

$$\alpha(v,T,m) = \begin{cases} 1 & \text{if } d = 0 \\ \alpha(v_1, T[v_1 \setminus v], m) & \text{if } 0 < k \leq m \\ \alpha(v_1, T[v_1 \setminus v], m) & \text{if } (k > m) \text{ and } (a_1 > a_{m+1}) \\ 1 + \alpha(v_1, T[v_1 \setminus v], m) & \text{if } (k > m) \text{ and } (a_1 = a_{m+1}) \end{cases}$$

where $a_i = \alpha(v_i, T(v_i \setminus v), m)$.

For all pairs of neighboring nodes u, v the values $\alpha(v,T,m), \alpha(u,T,m)$, $\alpha(v, T(v \setminus u), m)$ and $\alpha(u, T(u \setminus v), m)$ can be computed by solving the recurrent relation; hence the value $\alpha(v,T,m)$ is uniquely determined for every v in T, and so is the value $\alpha(T,m) = \min_v\{\alpha(v,T,m)\}$. Note that this computation can be performed efficiently using the saturation technique [32], as in [2, 24]. The importance of the function α is that it is a lower bound on A.

Theorem 7. $\mathcal{A}(v,T,m) \geq \alpha(v,T,m)$

Proof. By induction on the height $h(T,v)$ of T when rooted in v. Trivially, when $h(T,v) = 0$ (i.e., T is composed of a single node), $\mathcal{A}(v,T,m) = 1 = \alpha(v,T,m)$. Let the theorem hold for any tree T' and node v' where $0 \leq h(T',v') \leq h$. Consider now a tree T and node v when $h(T,v) = h+1$; by inductive hypothesis, for each subtree $T(v_j \setminus v)$, $\mathcal{A}(v_i, T(v_i \setminus v), m) \geq \alpha(v_i, T(v_i \setminus v), m)$. We now consider two cases,

Case 1. Let $\alpha(v_1, T(v_1 \setminus v), m) = \alpha(v_{m+1}, T(v_{m+1} \setminus v), m)$; in this case, by definition of α and inductive hypothesis,

$$\mathcal{A}(v_1, T(v_1 \setminus v), m) + 1 \geq \alpha(v_1, T(v_1 \setminus v), m) + 1 = \alpha(v,T,m).$$

If $\mathcal{A}(v_1, T(v_1 \setminus v), m) = \mathcal{A}(v_{m+1}, T(v_{m+1} \setminus v), m)$, then, by theorem 6 and inductive hypothesis, we have

$$\mathcal{A}(v,T,m) \geq \mathcal{A}(v_1, T(v_1 \setminus v), m) + 1 \geq \ \alpha(v,T,m).$$

If $\mathcal{A}(v_1, T(v_1 \setminus v), m) > \mathcal{A}(v_{m+1}, T(v_{m+1} \setminus v), m)$, then, by Theorem 6 and inductive hypothesis,

$$\mathcal{A}(v,T,m) \geq \mathcal{A}(v_1, T(v_1 \setminus v), m) > \mathcal{A}(v_{m+1}, T(v_{m+1} \setminus v), m) \geq$$

$$\geq \alpha(v_{m+1}, T(v_{m+1} \setminus v), m) = \alpha(v_1, T(v_1 \setminus v), m);$$

that is, $\mathcal{A}(v,T,m) \geq \alpha(v_1, T(v_1 \setminus v), m) + 1 = \alpha(v,T,m)$.

Case 2. Let $\alpha(v_1, T(v_1 \setminus v), m) > \alpha(v_{m+1}, T(v_{m+1} \setminus v), m)$; in this case, by definition of α, Theorem 6, and inductive hypothesis, we have

$$\mathcal{A}(v,T,m) \geq \mathcal{A}(v_1, T(v_1 \setminus v), m) \geq \alpha(v_1, T(v_1 \setminus v), m) = \alpha(v,T,m).$$

which completes the proof. □

Algorithm DECONTAMINATE(T, v, m)
move $\alpha(T, v)$ agents **to** v;
let $v_1, ..., v_d$ be the neighbours of v in T, and wlg let
$\alpha(v_i, T(v_i \setminus v), m) \geq \alpha(v_{i+1}, T(v_{i+1} \setminus v), m)$, $1 \leq i < d$.
for $j = d$ **downto** $j = 1$
DECONTAMINATE$(T(v_j \setminus v), v_j, m)$.
move all the agents at v_j **to** v.

Figure 13: Decontamination of a tree T starting from v, with $1 \leq m \leq d-1$

We next consider a lower bound on the number of moves performed by a minimal team of agents. Let $\mu(v, T, m)$ denote the minimum number of moves necessary to decontaminate T starting from v with $\mathcal{A}(v, T, m)$ agents, with all agents returning to v. Again, let $v_1, ..., v_d$ be the neighbors of v in T, where $k = \delta(v)$.

Theorem 8.

$$\mu(v, T, m) = 0 \quad \text{if } d = 0$$

$$\mu(v, T, m) = \sum_{1 \leq j \leq d} [\mu(v_j, T(v_j \setminus v), m) + 2\ \mathcal{A}(v_j, T(v_j \setminus v), m)] \quad \text{if } d > 0$$

Proof. Consider the subtrees $T(v_j \setminus v)$, $1 \leq j \leq d$, of T. By definition, $\mathcal{A}(v_j, T(v_j \setminus v), m)$ agents must be moved from v to v_j to decontaminate $T(v_j \setminus v)$; since all agents must return to v, this accounts for $2\ \mathcal{A}(v_j, T(v_j \setminus v), m)$ moves; by definition $\mu(v_j, T(v_j \setminus v), m)$ moves are needed to decontaminate $T(v_j \setminus v)$ with $\mathcal{A}(v_j, T(v_j \setminus v), m)$ agents starting from v_j. □

Let $\mathcal{M}(m)$ denote the minimum number of *moves* necessary to decontaminate T starting from v with $\mathcal{A}(m)$ agents and all agents returning to the homebase. By definition, $\mathcal{M}(m) = \min\{\mu(v, T, m) : \mathcal{A}(v, T, m) = \mathcal{A}(m)\}$.

5.2. Upper Bound and Optimality for Trees

We first prove that the lower bound of Theorem 7 is tight; that is

Theorem 9. $\mathcal{A}(v, T, m) = \alpha(v, T, m)$

The proof is constructive. Consider the protocol DECONTAMINATE(T, v, m) to decontaminate T starting from v shown in Figure 13.

Theorem 10. *Procedure* DECONTAMINATE(T, v, m) *decontaminates monotonically T starting from v using $\alpha(v, T, m)$ agents.*

Proof. The proof is by induction on the height $h(T, v)$ of T. The theorem trivially holds when $h(T, v) = 0$ (i.e., T is composed of a single node). Let the theorem hold for $0 \leq h(T, v) \leq h$. Consider now T when $h(T, v) = h + 1$. Let $v_1, ..., v_d$ be the neighbours of v in T, and wlg let $\alpha(v_i, T(v_i \setminus v), m) \geq \alpha(v_{i+1}, T(v_{i+1} \setminus v), m)$, with $1 \leq i < d$. By inductive hypothesis, for each subtree

Algorithm OPTIMALTREEDECONTAMINATION(T, m)
choose as starting point a vertex v such that
$\alpha(v, T, m) = \mathcal{A}(T, m)$ and $\mu(v, T, m) = \mathcal{M}(T, m)$,
DECONTAMINATE(T, v, m).

Figure 14: Optimal decontamination of a tree T, with $1 \leq m \leq d-1$

$T(v_j \backslash v)$ can be decontaminated by Procedure DECONTAMINATE$(T(v_j \backslash v), v_j, m)$ starting from v_j using $\alpha(v_i, T(v_i \setminus v), m)$ agents. Hence in the execution of Procedure DECONTAMINATE(T, v, m) with $\alpha(v, T, m)$, the subtrees $T(v_i \setminus v)$ $(1 < i \leq d)$, $T(v_i \setminus v)$ are sequentially decontaminated starting from those requesting the smallest number of agents. During this entire process v is always protected from recontamination: when $j \geq m$, by definition of α, $\alpha(v_j, T(v_j \setminus v), m) < \alpha(v_i, T(v_i \setminus v), m)$ and thus v_i is protected by the remaining agents; when $j < m$, v_i does not require protection anymore because at most $m-1$ neighbours are still contaminated. □

From Theorems 7 and 10, Theorem 9 follows.

To obtain a decontamination protocol that uses precisely $\mathcal{A}(T, m)$ agents performing $\mathcal{M}(T, m)$ moves is now straightforward (see Figure 14). Note that the algorithm performs an ordered sequential traversal of the tree, each step performed in one time unit; since once all nodes have been visited no return to the home base is needed, the total amount of time is less than $2(N-1)$

Theorem 11. *Protocol* OPTIMALTREEDECONTAMINATION *decontaminates monotonically any arbitrary tree T using $\mathcal{A}(T, m)$ agents performing $\mathcal{M}(T, m)$ moves in at most $2N-3$ steps.*

6. Conclusions and Open Problems

We have investigated the network decontamination problem under an arbitrary immunity level $m \geq 1$, which includes as a particular case the existing studies that assume no immunity ($m = 1$). We have established upper bounds on the number of agents needed to decontaminate meshes, tori, and tree networks. The algorithms yielding these bounds can operate even if the networks are completely asynchronous, either directly or with a simple modification requiring one additional agent. We have also established general lower bounds on the number of agents necessary for decontamination with immunity m. These bounds are tight in the case of trees; in the case of meshes and tori, they are tight only for some values of the parameters. Large gaps still exists.

These results are the first in the investigation of the decontamination problem with $m > 1$. Many research directions are now open.

First and foremost, there is a large gap between upper and lower bounds for meshes; the same holds for tori. Narrowing these gaps, in particular proving stronger lower bounds, is the outstanding open problem. This however appears to be a rather challenging research objective.

Another important task is to extend the investigation to other families of interconnection networks and graph classes of practical interest. In particular *meshes of trees*, that is bi-dimensional meshes with binary tree access to the rows and the columns; *butterflies*, for which the synchronization problem discussed in Figure 1 crucially arises; and *cube-connected cycles* (CCC's), an extension of hypercubes. Regarding to CCC's, note that local immunity for all the graphs with vertices of degree at most 3 has been studied in [24] where the majority of infected neighbors of each vertex was considered instead of a general value of m. Since all the vertices of a CCC have degree 3, immunity reduces here to the trivial case $m = 2$ for which two agents suffice. The case $m = 1$, instead, appears particularly hard to handle.

Recently, the model for synchronous environments has been extended allowing for a *temporal immunity* [14]: without an agent, a decontaminated node is immune to re-contamination for $\tau \geq 0$ units of time; after that, it can again be recontaminated. Note that $\tau = 0$ corresponds to the classical model. Temporal immunity has been investigated, for tree networks [14] and meshes and tori [8], assuming that (after τ time units) the re-contamination threshold is $m = 1$. An important research direction is thus to consider the monotone connected graph search problem in the most general model with both temporal immunity $\tau \geq 0$ and local immunity threshold $m \geq 1$.

In our investigation, as well as in most of the existing literature, the immunity threshold is a global parameter m. The only exception is [24] where the decontamination of meshes and tori was studied when the threshold immunity of a node is not a global parameter but a majority function of the degree of that node. Thus an important research direction is to define the immunity threshold as a function of the degree of each vertex. This generalizes the decontamination problem, including all the existing results as special cases.

Acknowledgments.
The authors would like to thank the anonymous referees for their insightful comments and questions. This work has been partially supported by the Italian Ministry MIUR under PRIN 2012C4E3KT - national research project AMANDA; by NSERC under Discovery Grants; and by Prof. Flocchini's University Research Chair.

[1] L. Barrière, P. Flocchini, F.V. Fomin, P. Fraignaud, N. Nisse, N. Santoro, and D.M. Thilikos. Connected graph searching. *Information and Computation* 219: 1-16, 2012.

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

[3] L. Barrière, P. Fraignaud, N. Santoro, and D. Thilikos. Searching is not jumping. *Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science* (WG), 34-45, 2003.

[4] D. Bienstock. Graph searching, path-width, tree-width and related problems. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 5: 33–49, 1991.

[5] D. Bienstock and P. Seymour. Monotonicity in graph searching. *Journal of Algorithms* 12: 239–245, 1991.

[6] L. Blin, P. Fraignaud, N. Nisse, and S. Vial. Distributed chasing of network intruders. *Theoretical Computer Science* 399 (1-2): 12-37. 2008.

[7] R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers* VI(5): 72–78, 1967.

[8] Y. Daadaa, P. Flocchini, N. Zaguia. Network decontamination with temporal immunity by cellular automata. *Proceedings of the 9th International Conference on Cellular Automata for Research and Industry* (ACRI), 287-299, 2010.

[9] D. Dereniowski. Connected searching of weighted trees. *Theoretical Computer Science* 412 (41): 5700-5713, 2011.

[10] D. Dereniowski. Approximate search strategies for weighted trees. *Theoretical Computer Science* 463 : 96-113, 2012.

[11] P. Flocchini, M.J. Huang, and F.L. Luccio. Decontamination of hypercubes by mobile agents. *Networks* 52 (3): 167-178, 2008.

[12] 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): 547-563, 2006.

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

[14] P. Flocchini, B. Mans, and N. Santoro. Tree decontamination with temporary immunity. *Proceedings of the 19th International Symposium on Algorithms and Computation* (ISAAC), 330–341, 2008.

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

[16] F.V. Fomin, D.M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science* 399(3): 236-245, 2008

[17] F.V. Fomin, D.M. Thilikos, and I. Todineau. Connected graph searching in outerplanar graphs. *Proceedings of the 7th International Conference on Graph Theory* (ICGT), 213-216, 2005.

[18] D. Ilcinkas, N. Nisse and D. Soguet. The cost of monotonicity in distributed graph searching. *Distributed Computing* 22 (2), 117-127, 2009.

[19] N. Imani, H. Sarbazi-Azadb, and A.Y. Zomaya. Capturing an intruder in product networks. *Journal of Parallel and Distributed Computing* 67 (9): 1018–1028, 2007.

[20] N. Imani, H. Sarbazi-Azad, A.Y. Zomaya, and P. Moinzadeh. Detecting threats in star graphs. *IEEE Transactions on Parallel and Distributed Systems* 20 (4): 474-483 , 2009.

[21] N. Kinnersley. The vertex separation number of a graph equals its pathwidth. *Information Processing Letters* 42(6): 345–350, 1992.

[22] L. Kirousis and C. Papadimitriou. Searching and pebbling. *Theoretical Computer Science* 47(2): 205–218, 1986.

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

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

[25] F.L. Luccio Contiguous search problem in Sierpinski graphs. *Theory of Computing Systems* 44(2): 186-204, 2009.

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

[27] S. Neufeld. A pursuit-evasion problem on a grid. *Information Processing Letters* 58(1): 5–9, 1996.

[28] N. Nisse. Connected graph searching in chordal graphs. *Discrete Applied Mathematics* 157 (12): 2603-2610, 2009

[29] N. Nisse, D. Soguet. Graph searching with advice. *Theoretical Computer Science* 410 (14): 1307-1318, 2009

[30] T. Parson. Pursuit-evasion in a graph. *Theory and Applications of Graphs*, Lecture Notes in Mathematics, Springer-Verlag, 426–441, 1976.

[31] D. Peleg. Local majorities, coalitions and monopolies in graphs: A review. *Theoretical Computer science*, 231-257, 2002.

[32] N. Santoro. *Design and Analysis of Distributed Algorithms*. Wiley-Interscience. Hoboken, NJ, 2007.

[33] P. Shareghi, H. Sarbazi-Azad, and N. Imani. Capturing an intruder in the pyramid. *Proceedings of the International Computer Science Symposium in Russia* (CSSR), 580-590, 2006.

[34] B. Yang, D. Dyer, and B. Alspach. Sweeping graphs with large clique number. *Discrete Mathematics* 309 (18): 5770-5780, 2009.