International Journal of Foundations of Computer Science
© World Scientific Publishing Company

Network Decontamination in Presence of Local Immunity*

FABRIZIO LUCCIO
Dipartimento di Informatica, Università di Pisa, Pisa, Italy
luccio@di.unipi.it

and

LINDA PAGLI
Dipartimento di Informatica, Università di Pisa, Pisa, Italy
pagli@di.unipi.it

and

NICOLA SANTORO
School of Computer science, Carleton University, Ottawa, Canada
santoro@scs.carleton.ca

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

We consider the problem of decontaminating a network infected by a mobile virus. The goal is to perform the task using as small a team of antiviral agents as possible, avoiding recontamination of disinfected areas. In all the existing literature, it is assumed that the immunity level of a decontaminated site is *nil*; that is, a decontaminated node, in absence of an antiviral agent on site, may be re-contaminated by any infected neighbour. The network decontamination problem is studied here under a new model of *immunity* to recontamination: we consider the case when a decontaminated vertex, after the cleaning agent has gone, will become recontaminated only if a majority of its neighbours are infected. We study the impact that the presence of local immunity has on the number of antiviral agents necessary to decontaminate the entire network. We establish both lower and upper bounds on the number cleaners in the case of (multidimensional) toroidal meshes, graphs of vertex degree at most three (e.g., cubic graphs, binary trees, etc.), and of tree networks. In all cases the established bounds are tight. All upper-bound proofs are constructive; i.e., we exhibit decontamination protocol achieving the claimed bound. We also analyze the total number of moves performed by the agents, and establish tight bounds in some cases.

Keywords: network decontamination, mobile virus, mobile agents, majority-based rule, toroidal meshes, trees.

*Research supported in part by the Natural Sciences and Engineering Research Council (Canada), Tecsis Co., and MIUR (Italy) under research project ALGO-NEXT.

1. Introduction

1.1. The Framework

Paralleling the diffusion of networked systems and the increase in both their size and complexity, the presence of dangerous and possibly malicious threats is experiencing a surge in variety and difficulty to be handled. Among the many pressing security threats in networked systems supporting mobile agents, two are predominant: the presence of static processes that can harm incoming agents (*harmful hosts*), and the presence of mobile agents that can harm the network (*harmful agents* or *intruders*) (e.g., see [12, 18, 21]). An example of the second type is a *virus*: an extraneous mobile agent infecting any visited site. The focus of this paper is on the latter.

The immediate impact of the presence of a virus in a network is that, in absence of anti-viral protection, the network sites become infected. In such cases, a crucial task is to decontaminate the infected network. The task is to be carried out by a team of anti-viral system agents (the *cleaners*). A cleaner is able to decontaminate an infected site once it arrives there; arriving at a clean site, clearly no decontamination operation needs to be performed by the cleaner. A decontaminated site can become re-contaminated (e.g. if the virus returns to that site in absence of a cleaner).

The goal is to perform the task minimizing the amount of decontamination and using as small a team of antiviral agents as possible. Additional desirable goals are to minimize the amount of agents' movements across the network, and the total time spent in the process.

A solution protocol will then specify the strategy to be used by the agents; that is, it specifies the sequence of moves across the network that will enable the agents, upon all being injected in the system at a chosen network site, to decontaminate the whole network avoiding any recontamination; any such protocol is said to be *monotone*. The reason to avoid recontamination derives from the requirement to minimize the amount of decontamination performed by the system agents: if recontamination is avoided, the number of decontamination operations are performed is the smallest possible, one per network site.

1.2. Previous Work

The *network decontamination* problem was first proposed by Breisch in 1967 in the context of a maze of caves contaminated by gas [6].

Since then, the problem has been extensively studied under the names of *graph search* and *intruder capture*. The reason for its success is that its several variants (edge-search, node-search, mixed-search, t-search, *etc.*) are closely related to standard graph parameters and concepts, including tree-width, cut-width, path-width, and, last but not least, graph minors [3]. Viewed sometimes as a pebbling problem in graphs (e.g., [14]) or as a pursuit-evasion game (e.g., [19]), the graph searching problem also arises in VLSI design through its equivalence with the gate matrix

layout problem [13]. The focus of these investigations is the analysis of the team size necessary to decontaminate classes of networks. Among the important results is that there are always size-optimal solutions that are *monotone*, i.e. that avoid recontamination [4, 15]. Contributions to related search problems can be found in [17, 22, 23] and the references therein.

In all these investigations on the graph search problem there is a common assumption made that the cleaning agents are able to *jump* across the network; that is, they assume that a cleaner is able to go "out of the system" and to reenter the system anywhere in one single step. This assumption clearly does not hold in the case of mobile agents in a network, since they can only move from a node to a *neighbouring* node. Actually, it does not hold even in the original setting of a maze of caves [6, 19]. The removal of this assumption makes the previous solutions and the existing bounds no longer valid [2].

The more general setting where agents can not jump, called *contiguous search*, has been first proposed and studied by Barriere *et al.* [1], where optimal strategies without recontamination were shown for trees. The investigations have thus focused on the analysis of classes of networks, and on the development of size-efficient monotone decontamination strategies for those classes. In particular, Flocchini *et al.* have studied *hypercubes* [7], *meshes* [9], and *chordal rings* [8], and *tori* [8]; Fomin *et al.* have investigated *outerplanar* graphs [10]; Blin *et al.* started the investigation of arbitrary graphs [5]. In the case of contiguous search, the analogous of the monotonicity result of [4, 15] for search with jumps holds in the case of trees [2] and other special graphs such as meshes, but not in general.

1.3. Recontamination and Immunization

What the new investigations on monotone strategies have in common with the old ones is the *recontamination model*, i.e., the rules that allow a decontaminated site to become recontaminated. In fact, in all investigations, it is assumed that a decontaminated site, in absence of a cleaner, becomes recontaminated if just one of its neighbours is contaminated. In other words, it is assumed that the immunity level of a decontaminated site is *nil*.

This assumption is quite strong and not necessarily realistic. In fact many systems employ *local majority* based rules at each site to enhance reliability and fault-tolerance. This is for example the situation in distributed systems where majority voting among various copies of crucial data are performed between neighbours at each step [20]. Indeed, local voting schemes are used as a decision tool in a number of consensus and agreement protocols, in consistency resolution protocols in distributed database management systems, 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.

Systems employing majority-based local voting schemes have clearly a higher level of resistance to viral recontamination. In fact, a decontaminated vertex, after the cleaning agent has gone, will avoid recontamination as long as a (strong)

majority of its neighbours are infected. In other words, the immunity level of a decontaminated vertex is half of its degree.

1.4. Main Results

In this paper we consider the network decontamination problem under our new model of *immunity to recontamination*. We study the effects of this level of immunity in toroidal meshes, in trees, and in graphs of vertex degree at most three. Separately we examine the case of k-ary trees with $k \geq 3$. We design and present monotone protocols for decontaminating such networks, and establish lower bounds on the number of cleaners and moves necessary for decontamination. We prove that the upper and lower bounds on the number of agents are tight for toroidal meshes and tree networks, and that all the other upper and lower bounds are close to each other. In particular we show that 2^k agents are necessary and sufficient for decontaminating a k-dimensional toroidal mesh regardless of its size; this must be contrasted with e.g. the $2\min\{n, m\}$ agents required to decontaminate a $n \times m$ toroidal mesh without local immunization [8]. Interestingly, among tree networks, binary trees were the worst to decontaminate without local immunization, requiring $\Omega(\log n)$ agents in the worst case [1]. Instead, with local immunization, they can be decontaminated by a *single* agent. In addition we prove that any graph with maximum vertex degree three can be dencontaminated with two agents (or one agent, if the graph contains a vertex of degree one), and these bounds are tight.

2. Basic Properties

Let the network be represented by a connected undirected graph. Vertices are indicated with a white circle, and said to be *white*, if they have not been visited by an agent yet; are indicated with a black circle, and said to be *black*, if they contain an agent; are indicated with a star, and said to be *star*, if they have been previously visited by an agent that eventually left, and have not been recontaminated yet. Black and star vertices are said to be *clean*. For a vertex v, let $d(v)$ denote its degree, $m(v) = \lfloor d(v)/2 \rfloor + 1$ denote the majority of its neighbours, and $s(v)$ denote the number of its clean neighbours at any given moment.

The presence of the majority rule imposes immediate limits on computability; for example, to avoid recontamination of a vertex v, an agent may leave it only if a *strong majority* (i.e., $m(v)$) of clean neighbours remains. Notice that the status of a vertex (clean or contaminated) is not detectable from a distance. We have:

Lemma 1 *At any step of a monotone algorithm, let A be an agent located at vertex v:*

1. *if $s(v) < m(v) - 1$, A can not move from v;*
2. *if $s(v) = m(v) - 1$, A can only move to a white neighbour of v;*
3. *if $s(v) \geq m(v)$, A can move to any neighbour of v.*

Lemma 2 *At any step of a monotone algorithm all the clean vertices form a connected subgraph.*

The proof of Lemma 2 is immediate because all agents start moving from the same vertex. We shall now study monotone algorithms for decontaminating meshes and trees, and prove lower and upper bounds to the number of agents and moves.

3. Decontaminating Toroidal Meshes

3.1. Upper Bounds

We start our analysis with *k-dimensional toroidal meshes*. One such a network of $n_1 \times n_2 \times \ldots \times n_k$ vertices is such that each vertex $v_{i_1,i_2,\ldots,i_k}$, with $0 \leq i_j \leq n_j - 1$, is connected to the $2k$ vertices $v_{i_1-1,i_2,\ldots,i_k}$, $v_{i_1+1,i_2,\ldots,i_k}$, $v_{i_1,i_2-1,\ldots,i_k}$, $v_{i_1,i_2+1,\ldots,i_k}$, ..., $v_{i_1,i_2,\ldots,i_k-1}$, $v_{i_1,i_2,\ldots,i_k+1}$, where the operations on each index i_j, are done modn_j. For any vertex v we have $d(v) = 2k$, $m(v) = k + 1$. We shall see that 2^k agents are sufficient.

For $k = 1$ the mesh is a ring, and $2^1 = 2$ agents are trivially necessary and sufficient. A straightforward monotone algorithm for a ring of vertices v_i, $0 \leq i \leq n_1 - 1$, is as follows:

Algorithm 1 *Decontaminating a ring R of n_1 vertices with a set $S = \{A_0, A_1\}$ of two agents starting in an arbitrary vertex v_i.*

move A_1 to vertex v_{i+1};
RING(R, S) */procedure call*
procedure RING(R, S)
  move synchronously A_0, A_1 in opposite directions
  until they reach two adjacente vertices v_{j+1}, v_j $(j = i + \lfloor n_1/2 \rfloor)$.

Note that Algorithm 1 makes $n_1 - 1$ moves, and such a number of moves is clearly necessary for decontaminating all the vertices.

We now present the general monotone algorithm for $k > 1$. Subsequently, to make it clear, we will show how it works for $k = 2$. For simplicity we assume initially that all the n_i are even, postponing a discussion for n_i odd.

Algorithm 2 *Decontaminating an $n_1 \times n_2 \times \ldots \times n_k$ k-dimensional toroidal mesh M, with $k > 1$ and all n_i even, with a set $S = \{A_0, A_1, \ldots, A_{2^k-1}\}$ of 2^k agents starting in an arbitrary vertex $\bar{v}$.*
notation: v^-, v^+ are the vertices adjacent to a vertex $v = v_{i_1,i_2,\ldots,i_k}$, with index i_k decreased or increased by 1, respectively.

PLACE($S, \bar{v}, k$) */procedure call*
procedure PLACE(S, v, k) *Initial positioning of the agents.*
divide S in two disjoint subsets S_0, S_1 of 2^{k-1} agents each;
move synchronously the agents of S_1 one step, from v to v^+;
if $k > 1$ **then**
do in parallel (PLACE($S_0, v, k-1$), PLACE($S_1, v^+, k-1$)).
The agents are now in the vertices of a k-cube C.

notation: C_0, C_1 are the subcubes of C with $i_k = h$, $i_k = h+1$, respectively (the value of h depends on $\bar{v}$); S_0, S_1 are the subsets of agents in C_0, C_1; M_j is the submesh of M with $i_k = j$, $0 \leq j \leq n_k - 1$.

MESH(M, C, S, k) */procedure call*
procedure MESH(M, C, S, k) *decontaminating M starting with the agents in C.*
if $k = 1$ **then** RING(M, S) **else**
do in parallel
(MESH($M_h, C_0, S_0, k-1$), MESH($M_{h+1}, C_1, S_1, k-1$));
The agents are now in the vertices of a new k-cube C.
for $j \leftarrow 1$ **to** $n_k/2 - 1$ **do**
redefine C_0, C_1 by replacing each $v \in C_0$ with v^-, each $v \in C_1$ with v^+;
move synchronously the agents of S_0, S_1 one step to C_0, C_1;
do in parallel
(MESH($M_{h-j}, C_0, S_0, k-1$), MESH($M_{h+1+j}, C_1, S_1, k-1$)).
The agents are now in the vertices of a new k-cube C.

Before proving the correctness of the algorithm, let us simulate its execution on a 2-dimensional toroidal mesh of 6×6 vertices, with four agents starting from $v_{0,0}$. We have $d(v) = 4$, $m(v) = 3$. Indices i_1, i_2 denote columns and rows, respectively. Figure 1 shows the agent positions at the following steps. (a) After procedure PLACE: the agents are in a 2-cube C, with C_0, C_1 respectively in rows 0, 1. (b) After the parallel calls MESH($M_h, C_0, S_0, k-1$), MESH($M_{h+1}, C_1, S_1, k-1$) with $h = 0$, $k = 2$, i.e., calls on M_0, M_1, respectively rows 0, 1 of the mesh: these calls decontaminate the two rows in two parallel steps (8 total moves of the 4 agents), then the agents are in a new 2-cube. Note that each vertex left by an agent is adjacent to three clean vertices and stays clean forever. (c) After the first move of the agents of S_0, S_1 in the **for** cycle, respectively to rows 5 (through toroidal connections) and 2. Now the two pairs of agents can decontaminate the rows independently, since each row is adjacent to another clean row, hence three clean neighbours are assured for each vertex visited. (d) After completion of the **for** cycle: the agents are in a new 2-cube.

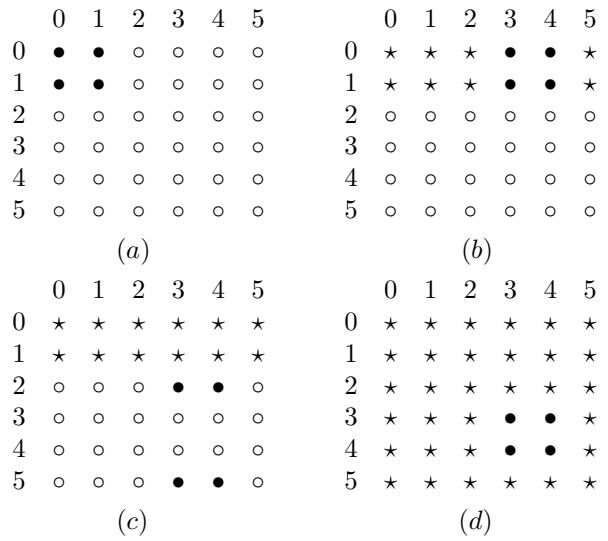

Figure 1: Four agents decontaminating a 2-dimentional mesh, starting from vertex $v_{0,0}$. Agent positions after the execution: of **procedure** PLACE (a); of the first **do in parallel** of **procedure** MESH (b); of the first **move** in the **for** cycle (c); of the whole algorithm (d).

The number of moves is 4 in procedure PLACE, plus $n_1 \times n_2 - 4$ in procedure MESH, for a total of $n_1 \times n_2 = 36$. Note that if the mesh had $i_2 = 5$ rows 0 to 4, three of them should be decontaminated in the **for** cycle. The two pairs of agents should work in parallel on rows 4 and 2, then one pair should work on row 3 while the other stands still. In so doing, the two pairs would finish in non adjacent subcubes, and four (i.e., $i_1 - 2$) additional moves are needed for bringing all the agents in a 2-cube. We have in general:

Theorem 1 *2^k synchronous agents can decontaminate a k-dimensional toroidal mesh of $N = n_1 \times n_2 \times \ldots \times n_k$ vertices, making $U_k = N + (k-2) \cdot 2^{k-1}$ moves if all the n_i are even, $U_k = N + (k-2) \cdot 2^{k-1} + O(N)$ moves otherwise.*

Proof. Use Algorithm 2, assuming that all n_i are even. Monotonicity and correctness of the procedure PLACE are immediate. No vertex is left without at least one agent after being visited, and the final placement obtained follows by the recursive structure of a k-cube. The procedure MESH is composed of two phases: 1. a **do in parallel** statement, and 2. a **for** cycle. In phase 1, the agents of the two subsets S_0, S_1 work in parallel on two adjacent submeshes of dimension $k-1$. Assuming inductively that each submesh is correctly decontaminated in dimension $k-1$ (the induction basis for $k = 1$ has already been proved), the pair of adjacent submeshes are also correctly decontaminated in dimension k because the degree of each vertex increases by two, hence the required majority of clean vertices increases by one, where the extra clean nighbours are mutually provided by the two

submeshes. In phase 2, then, each of the subsets S_0, S_1 can work independently on a submesh of dimension $k-1$, because the extra clean neighbour is provided by the submeshes already decontaminated.

The total number of agent moves can be expressed as $U_k = P_k + Q_k$, where the two addends are respectively due to the procedures PLACE and MESH. We have $P_k = 2^{k-1} + 2P_{k-1}$, because the 2^{k-1} agents of S_1 are first moved by one step, then the agents of S_0 and S_1 are recursively moved to two $(k-1)$-cubes. Since we have $P_1 = 1$ for the ring, the above recurrence immediately yields $P_k = k \cdot 2^{k-1}$. If all the n_i are even, the procedure MESH makes moves only to white vertices, then we have $Q_k = N - 2^k$ and the stated value of U_k follows.

If one (or more) of the n_i is odd the number of agents is still 2^k, but the **for** cycle of MESH has to be modified by executing the last iteration for $\lceil n_i/2 \rceil - 1$ only for subset S_0, while the agents in S_1 stand still. After this, however, S_0 and S_1 are in two non adjacent subcubes, and $O(n_i)$ extra moves are needed to bring them into a unique k-cube. Summing up for all the n_i, the stated value of U_k again follows. □

Applying Theorem 1 for $k = 1$, $k = 2$ and even n_i, we have the values $U_1 = n_1 - 1$ and $U_2 = n_1 \times n_2$ already found. For $k > 2$, however, the algorithm requires more moves than white cells, because of the initial construction of the k-cube.

3.2. Lower Bounds

We now prove that the upper bound of Theorem 1 on the number of agents is strict for any $k \geq 1$. For the number of moves, instead, we are able to derive a matching lower bound for $k = 1, 2$ only. We have:

Theorem 2 *For decontaminating a k-dimensional toroidal mesh M of $N = n_1 \times n_2 \times \ldots \times n_k$ vertices, with k constant with respect to $n_1, \ldots, n_k$, at least 2^k agents are necessary.*

Proof. For $k = 1$ the bound is obvious. For $k > 1$, recall from lemma 2 that, at any step of a decontaminating algorithm, all the clean vertices form a connected subgraph G of M. When n moves have been made, $n < min(n_1, \ldots, n_k) - 1$, no agent has completed a turn of M in any dimension. Therefore, due to the mesh structure of M, G is a h-dimensional polyhedron, $h \leq k$. The external surface of G contains "poly-vertices" and "poly-edges" (in fact, polyhedral convex vertices, and edges, with a prefix "poly" to be distinguished from mesh vertices and edges), and poly-faces of any dimension up to $h-1$. Poly-vertices have h clean neighbours, mesh vertices on the poly-edges have $h+1$ clean neighbours, and all the other vertices of G have $> h+1$ clean neighbours. For avoiding recontamination, the vertices of G must have at least $k+1$ clean neighbours, or contain an agent. Then, all poly-vertices must contain an agent, and, for $h < k$, all mesh vertices on the poly-edges must also contain an agent. Note that G has at least 2^h poly-vertices, then at least 2^h agents are needed. For $h = k$ the bound on the number of agents immediately

follows. For $h < k$ we must prove that other $2^k - 2^h$ agents are needed. In fact, since k is independent of $n_1, \ldots, n_k$, the total length of the poly-edges increases arbitrarily over $2^k - 2^h$ for increasing number n of moves, $n < min(n_1, \ldots, n_k) - 1$, and the bound follows for any $h < k$. □

We can now establish a lower bound on the number of moves. We have:

Theorem 3 *Decontaminating a k-dimensional toroidal mesh of N vertices with 2^k agents requires at least L_k moves, with $L_1 = N-1$, $L_2 = N$, and $L_k = N+2^k-2k-2$ for $k > 2$.*

Proof. Since all the agents start from the same vertex v, at least $N - 1$ moves are necessary to decontaminate the remaining white vertices. Thus the bound L_1 is immediate. However, a similar bound could be reached for $k > 1$ only if the agents moved to white vertices only, that is impossible as we shall prove now. For $k = 2$, $d(v) = 4$, it is immediate to see by lemma 1 that at least one agent must move onto a vertex containing another agent, thus proving the bound L_2. In fact two agents, say A_0, A_1, can take different directions in the first two moves from v. At this point A_0, A_1 can not move further. Then either another agent A_2 joins A_0 or A_1 (and we have proved the bound), or A_2 moves to a third white vertex adjacent to v. In this case, however A_0, A_1, A_2 can not move further, then either A_3 joins one of them, or A_3 moves to the remaining white vertex adjacent to v, but then no agent can move further. The bound L_2 is then proved. For $k > 2$ we have $d(v) = 2k$, then at most $2k$ agents can move from v to white neighbours, while other $2^k - 2k - 1$ agents must move to already visited neighbours, and the last agent may remain in v. The bound L_k immediately follows. □

Then, the upper bound U_k of Theorem 1 is tight for $k = 1, 2$, but we still have a gap between L_k and U_k for $k > 2$. We conjecture that L_k should be raised to meet U_k. Instead of resorting to a complicated extension of the reasoning made for $k = 2$, however, we prefer to leave the problem open, in search of a brilliant argument valid for all k.

4. Decontaminating Networks with Degree ≤ 3

In this section we consider the class of graphs with maximum vertex degree three. This large class of graphs comprises, in addition to rings and lines, a large variety of graphs, including cubic graphs and binary trees.

The existence of a local-majority immunity yields a perhaps surprising result. In fact, regardless of their size, the graphs in this class can all be decontaminated by at most two cleaners. Furthermore, the protocol yielding this bound is remarkably simple. For a graph G, let *d-max(G)* $= max_{v \in G}(d(v))$. We have:

Theorem 4 *Let $G = (V, E)$ be a graph with* d-max$(G) \leq 3$. *If $\forall v \in V$, $d(v) > 1$ then two agents are necessary and sufficient to decontaminate G; otherwise one agent suffices.*

The proof of this theorem is constructive. Let r be the homebase. The protocol is a DF-traversal of the network by an agent while the other (if any) does not move from the homebase.

Algorithm 3 TRAVERSE(r, G).
if $d(r) = 1$ **then** *use one agent A;*
else *use two agents A, B, of which only A moves;*
Starting from r, agent A performs a depth-first traversal of G.

We will now show that the protocol is correct.

Lemma 3 *Let $G = (V, E)$ be a graph with* d-max$(G) \leq 3$. *Protocol TRAVERSE(r, G) correctly decontaminates G with $2|E|$ moves.*

Proof. The depth-first traversal of G by agent A constructs a depth-first spanning-tree T_r of G, rooted in r. Let $p_r(x)$ denote the parent of x in T_r, where by convention $p_r(r) = r$. In the depth-first traversal, every link (x, y) will be traversed twice. Consider when the agent A traverses (x, y) for the first time; without loss of generality let A move from x to y. This implies that y is still infected. By induction, assume that all the decontaminated area, including $p_r(x)$, can not be recontaminated (this is trivially true at the beginning). When A moves to y, if $x \neq r$ then x has two distinct decontaminated neighbours, y and $p_r(x)$, and therefore will not be recontaminated; if $x = r$, then either agent B is there (if $d(x) > 1$) or x is a leaf; in both cases, the move of A to y will decontaminate y without recontaminating x. Thus, when y is decontaminated, $x = p_r(y)$ is also decontaminated and the rest of the decontaminated nodes can not be recontaminated because of this move. The number of moves follows from the fact that in the depth-first traversal, every link is traversed twice. □

This completes the proof of Theorem 4, provided that, if G contains a vertex with degree one, the homebase r is such a vertex.

Note that the depth-first traversal used by protocol TRAVERSE can be substituted by any traversal algorithm for agents, including those *without return*, i.e., that stop as soon as all the nodes have been traversed, without the agent returning to the homebase. The use of such a traversal could reduce the number of moves; however, a termination criterion (e.g., knowledge of $n = |V|$) must be accessible to the moving agent.

In the case of *binary trees*, this reduction can be significant. In fact, let $diam(G)$ denote the diameter of G; then

Theorem 5 *A single agent can decontaminate any known binary tree T of n vertices with $2(n-1) - diam(T)$ moves.*

Proof. By choosing the homebase and the last node to be visited to be leaves on a diametral path. □

This decreases the bound of Lemma 3 by $diam(T)$. It is immediate to verify that this bound is indeed tight:

Theorem 6 *For decontaminating a known binary tree of n vertices and diameter d with one agent, at least $2(n-1)-d$ moves are necessary.*

5. Decontaminating General Trees

Tree structured networks have been carefully studied without local immunity [1, 2, 16]. As one may expect, the results in our model are different.

We have already examined the case of trees where the maximum degree is at most three (e.g., binary trees and lines) in the previous section. We will now examine the general case of arbitrary trees.

The analysis of the number of cleaners needed to decontaminate an arbitrary tree T is much more difficult than in the binary case. Let $\mu(T,r)$ denote the minimum number of agents needed to decontaminate T starting from r; then

$$\mu(T) = \min_r\{\mu(T,r)\} \tag{1}$$

denotes the minimum number of agents needed to decontaminate T.

In this section, we will first provide a complete characterization of $\mu(T)$; this characterization allows the determination of its exact value for every tree T. We will then present a monotone protocol that allows any known tree T to be decontaminated by a team of precisely $\mu(T)$ cleaners.

5.1. Lower Bound

We will first of all establish a lower bound on $\mu(T,r)$. To do so we first need to introduce some terminology. Given a homebase r in T, let T_r denote T when rooted in r; given a node x, let $p_r(x)$ denote its parent in T_r.

A decontaminated node x is said to be *guarded* when an agent resides in it. A decontaminated node x is said to be *stable* if the following two conditions hold:

1. x is not guarded, or is guarded but the removal of all its guards do not recontaminate it; and
2. a majority of its children in T_r are stable: the majority is *strong* if $x = r$, *simple* otherwise.

The following property trivially holds for any execution of any solution protocol starting from the homebase r:

Lemma 4
(a) Every node $x \neq r$ must be decontaminated by cleaners arriving from $p_r(x)$.
(b) Every node must become stable.
(c) Once a node is stable, it must remain clean.

Given a node x in T_r, let us denote by $\nu(x,r)$ the minimum number of agents needed to make x stable. The relationship between μ and ν is rather straightforward. In fact, since (by Lemma 4) every node must become stable, then

Theorem 7 $\mu(T,r) \geq \max_x\{\nu(x,r)\}$

In other words, any lower bound on ν provides immediately a lower bound on μ. This is what we will do in the rest of this section.

Let $y_1, y_2, \ldots, y_k$ be the children of x in T_r and, without loss of generality, let $\nu(y_i,r) \leq \nu(y_{i+1},r)$. An important property is the following:

Lemma 5 *Let $x \neq r$. Then*

$$\nu(x,r) \geq \nu(y_{\lceil \frac{k}{2} \rceil},r) + \alpha(x,r),$$

where $\alpha(x,r) = 0$ if $\nu(y_{\lceil \frac{k}{2} \rceil},r) > \nu(y_{\lceil \frac{k}{2} \rceil - 1},r)$, and $\alpha(x,r) = 1$ otherwise.

Proof. Consider the time when x becomes stable. At this time, by definition of stable node, a (simple) majority of its children must be stable. This means that at least one child y_j with $j \geq \lceil \frac{k}{2} \rceil$ must be stable before x becomes stable; this in turn implies neighbours must be that at least $\nu(y_{\lceil \frac{k}{2} \rceil},r)$ agents are needed to make x stable. In the case $\nu(y_{\lceil \frac{k}{2} \rceil},r) = \nu(y_{\lceil \frac{k}{2} \rceil - 1},r)$, actually at least one more agent is needed to make x stable. In fact, at least two children y_a and y_b with $b > a \geq \lceil \frac{k}{2} \rceil - 1$ must be stable before x becomes stable. If y_a and y_b are made stable sequentially, since $\nu(y_b,r) \geq \nu(y_a,r) \geq \nu(y_{\lceil \frac{k}{2} \rceil - 1},r) = \nu(y_{\lceil \frac{k}{2} \rceil},r)$, to avoid recontamination, x must be left guarded while the first of y_a and y_b is rendered stable; hence at least $\nu(y_a) + 1 \geq \nu(y_{\lceil \frac{k}{2} \rceil - 1},r) + 1 = \nu(y_{\lceil \frac{k}{2} \rceil},r) + 1$ cleaners are needed, proving the lemma. If y_a and y_b are made stable concurrently the number of agents required would be $\nu(y_a,r) + \nu(y_b,r) \geq 2\ \nu(y_{\lceil \frac{k}{2} \rceil - 1},r) = 2\ \nu(y_{\lceil \frac{k}{2} \rceil},r) \geq \nu(y_{\lceil \frac{k}{2} \rceil},r) + 1$ proving the lemma. □

The case when $x = r$ is similar except that, by definition, to be stable the root requires that a strong (instead of simple) majority of its children must be stable. The differences in the bounds are only in the rounding in the indices:

Lemma 6 *Let $x = r$. Then*

$$\nu(r,r) \geq \nu(y_{\lfloor \frac{k}{2} \rfloor + 1,},r) + \beta(x,r)$$

where $\beta(x,r) = 0$ if $\nu(y_{\lfloor \frac{k}{2} \rfloor + 1},r) > \nu(y_{\lfloor \frac{k}{2} \rfloor},r)$, and $\beta(x,r) = 1$ otherwise.

Lemmas 5 and 6 provide a recursive characterization of the function ν. To complete the characterization we must to provide the base case, which trivially holds:

Lemma 7 *Let x be a leaf. Then $\nu(x, r) = 1$*

5.2. Tightness of Bound

In this section we show that the lower bound determined in the previous section (Theorem 7) is tight. We will first of all prove that the expressions of Lemmas 5 and 6 are actually equalities. We will do so constructively, providing a protocol, STABILIZE, that makes a node stable with precisely the number of cleaners expressed by the bound.

Let $y_1, y_2, \ldots, y_k$ be the children of x in T_r where, as before and without loss of generality, $\nu(y_i, r) \leq \nu(y_{i+1}, r)$. The protocol STABILIZE is rather simple and is described below, where *stable*(.) is a predicate initially set to *false* for all nodes:

Algorithm 4 STABILIZE($x, \nu(x, r)$)
if x is a leaf **then**
move one agent from $p_r(x)$ to x to decontaminate it;
return the agent to $p_r(x)$;
stable(x) $\leftarrow$ **true**;
else
move $\nu(x, r)$ agents from $p_r(x)$ to x;
if $x \neq r$ **then** $l \leftarrow \lceil \frac{k}{2} \rceil$ **else** $l \leftarrow \lfloor \frac{k}{2} \rfloor + 1$;
for $i = 1, ..., l$ **do** STABILIZE($y_i, \nu(y_i, r)$);
return the agents from x to $p_r(x)$;
stable(x) $\leftarrow$ **true**.

Theorem 8

1. *(a)* STABILIZE($x, \nu(x, r)$)) *correctly makes x stable using $\nu(x, r)$ agents starting from $p_r(x)$.*
2. *(b) The expressions of Lemmas 5 and 6 are equalities.*

Proof. By induction on $level(x, r)$, where $level(x, r)$ is recursively defined as follows: $level(x, r) = 0$ if x is a leaf, and $level(x, r) = 1 + Max_{y_i}\{level(y_i, r)\}$ otherwise. Notice that $\forall x$, $level(x, r) < level(r, r) = l_r$. The lemma trivially holds if x is a leaf, i.e. when $level(x) = 0$.

Let it hold for all nodes with level at most $l \geq 0$, and let $level(x) = l+1$. Consider first the case $l + 1 < l_r$; i.e., $x \neq r$. We must prove that STABILIZE($x, \nu(y_{\lceil \frac{k}{2} \rceil}, r) + \alpha(x, r)$) correctly makes x stable. Observe that STABILIZE is invoked sequentially

to $y_1, ..., y_{\lceil \frac{k}{2} \rceil}$ in that order; since $\nu(y_i, r) < \nu(y_{\lceil \frac{k}{2} \rceil}, r) + \alpha(x, r)$, $1 \leq i \leq \lceil \frac{k}{2} \rceil$, x is guarded by an agent during all but the last of these invocation. During this last invocation, $p_r(x)$ as well as a simple majority of the children of x are decontaminated; hence x is not recontaminated during this time. When the last invocation terminates and the predicate $stable(x)$ is set to true, by inductive hypothesis, a simple majority of the children of x are stable; since also $p_r(x)$ is decontaminated, a strong majority of the neighbours of x are decontaminated. Therefore, even though x is unguarded (all the agents have returned to $p_r(x)$ after the execution of STABILIZE) x is not recontaminated. In other words, at this time x satisfies the definition of a stable node.

In the case $l + 1 = l_r$ (i.e., $x = r$), the proof that STABILIZE$(x, \nu(y_{\lfloor \frac{k}{2} \rfloor + 1}, r) + \beta(x, r))$ correctly makes x stable follows using a reasoning similar to the one above. □

In other words, the bounds on ν are tight. While this fact does not imply that the lower bound of Theorem 7 on μ is tight, this is indeed the case. In fact it is possible to effectively use protocol STABILIZE to decontaminate the entire network with the number of agents expressed in Theorem 7. The protocol achieving this task is as follows:

Algorithm 5 DECONTAMINATE(r, T)
STABILIZE$(r, \nu(r, r))$
$S \leftarrow \{x | stable(x) = \textbf{true}\}$;
while $S \neq V$ **do**
select $y \in V \setminus S$ such that $p_r(y) \in S$;
move $\nu(y, r)$ agents from r to $p_r(y)$;
STABILIZE$(y, \nu(y, r))$;
move all agents to r.

The process performed by DECONTAMINATE is rather simple: first it makes the root r stable; then, repeatedly, it selects a node y that is not stable but has a stable neighbour z and makes y stable, until all nodes are stable. It is easy to verify that among the nodes not yet stable, such a node y always exists; furthermore $z = p_r(y)$. Notice that in order to use the protocol, the whole structure of the tree must be known. The correctness of the protocol, as well as the number of agents sufficient for the operation, then follow directly from the correctness and the number of agents used by protocol STABILIZE:

Lemma 8 *Protocol DECONTAMINATE(r, T) correctly decontaminates T starting from r using $\max_x\{\nu(x, r)\}$ agents.*

The tightness of the bound on μ and the optimality of protocol DECONTAMINATE now follow:

Theorem 9 *Protocol DECONTAMINATE(r, T) decontaminates T starting from r using $\mu(r, T)$ agents.*

Proof. From Lemma 7 and Theorem 8. □

As a consequence we have also that $\mu(T) = \min_r \max_x \{\nu(x, r)\}$; thus, the strategy of choosing as a homebase the node r with minimum $\mu(x, T)$ yields the following:

Theorem 10 *It is possible to decontaminate an arbitrary tree T using $\mu(T)$ agents.*

5.3. Specific Tree Classes

The characterization we just provided is very general and the values of ν and thus μ can be easily computed for any specific tree. However, no closed formula can be stated for a generic tree.

Closed formulas can be given for specific classes of trees T with *d-max(T)* > 3 (trees with *d-max(T)* ≤ 3 have been studied in Section 4). Let us examine the class of *complete k-ary trees* with $k \geq 3$ and height h, that is rooted trees where all the internal vertices have k children, and all the leaves are at level 0 (the root is at level h).

Let $T_r[k, h]$ be a tree in this class, and let r be its root. The calculation of $\mu(T_r[k, h])$ is straightforward. We have:

Lemma 9 *Let $k \geq 4$. Then $\mu(x, T_r[k, h]) = h + 1$.*

Proof. For simplicity, denote by T_z the tree $T_r[k, h]$ when the homebase is z. By Lemmas 5-7 and the fact that (by Theorem 8(b)) the expressions there are equalities, it follows that $\nu(x, z) = level(x, T_r[k, h]) + 1$. Hence the claim. □

Corollary 1 *$h+1$ agents can decontaminate a complete k-ary tree of height h, with $k \geq 4$.*

In the case $k = 3$, the same calculations show that one less agent suffice. We have:

Lemma 10 $\mu(T_r[3, h]) = h$

Corollary 2 *h agents can decontaminate a complete 3-ary tree of height h.*

6. Conclusions

6.1. Summary of Results

We have introduced and studied the problem of decontaminating a network in presence of *local immunity*. In contrast with the traditional decontamination models that assume no immunity, the proposed model describes the decontamination problem more accurately in the case of networks that employ local-majority rules.

We have studied the effects of local immunity on the costs of *monotone decontamination*, in particular on the minimum number of system agents needed. We have established both lower and upper bounds on the number cleaners in the case of (multidimensional) toroidal meshes, graphs of vertex degree at most three (e.g., cubic graphs, binary trees, etc.), and of tree networks. In all cases the established bounds are tight. All upper-bound proofs are constructive; i.e., we exhibit monotone decontamination protocols achieving the bound. We have also analyzed our protocols with respect to the total number of moves performed by the agents.

With respect to previous works, the availability of local immunity induces great changes in the results. For example, we have shown that both in binary trees and in toroidal meshes, a constant number of cleaners suffices regardless of the size of the network; in contrast, without local immunity, the number of cleaners must be a function of the size of the network.

6.2. Synchronous vs Asynchronous

The protocols we have described for networks of degree at most three, and those for arbitrary tree networks make no assumptions on the amount of time required for an operation by an agent (decontamination, move, computation). Thus, they work even in truly *asynchronous* systems.

The protocols we have introduced for (multidimensional) mesh networks are described for a *synchronous* system: all agents are synchronized, and each operation of a specific type (decontamination, move, computation) requires always the same amount of time regardless of the agent performing it and of its location. This means that the upper-bounds for multidimensional meshes, as described, hold for synchronous systems. This is however not a severe limitation. In fact, as already observed by Flocchini *et al.* in systems without local immunity [7, 8, 9], and using exactly the same technique proposed there, it is possible to transform the proposed protocols in ones that make no assumptions on the amount of time required for an operation by an agent and, thus, work even in *asynchronous* systems. The price to pay is the use of an additional system agent, called *synchronizer*, that coordinates and guides the movements of the other agents [7, 8, 9]. In other words, the gap between upper and lower bound in the case of asynchronous multidimensional meshes

is *at most one*.

Using this approach, the number of moves performed by the cleaners is unchanged; to this, we must clearly add the number of moves performed by the synchronizer.

6.3. Open Problems

In this paper we have introduce the notion of local immunity and started the analysis of its impact to the process of monotone decontamination. Many problems and questions are opened by this paper.

First and foremost, what happens in other classes of networks ? Is the presence of majority-based immunity going to cause dramatic improvements in other networks, comparable to the ones we observed in toroidal meshes and trees ? And if not, why ?

For the networks studied in this paper, it is possible to improve some of the results, as for example finding a tight lower bound on the number of moves. In this regards, what would happen if the number of moves is the main cost measure, instead of the number of agents ? We already know that, in this case, the solution for tree networks can be improved (increasing the number of agents).

Another, more fundamental question relates to the monotone nature of the solution protocols. If we remove monotonicity, what would be the minimum number of agents needed for decontamination of the network considered here ? It is known that, without immunity, the same number of agents are needed in trees regardless of whether or not monotonicity is required [2].

Acknowledgements

We would like to thank Paola Flocchini whose suggestions have lead to the results of Section 4. We would also like to thank the anonymous referees for their useful comments and suggestions.

References

1. L. Barrière, P. Flocchini, P. Fraignaud, and N. Santoro. Capture of an intruder by mobile agents. *Proc. 14th Symp. Parallel Algorithms and Architectures* (SPAA'02), 200-209, 2002.
2. L. Barrière, P. Fraignaud, N. Santoro, and D. Thilikos. Searching is not jumping. *Proc. 29th Int. Workshop on Graph-Theoretic Concepts in Computer Science* (WG'03), 34-45, 2003.
3. D. Bienstock. Graph searching, path-width, tree-width and related problems. *DIMACS Series in Disc. Maths. and Theo. Comp. Sc.*, 5, 33–49, 1991.
4. D. Bienstock and P. Seymour. Monotonicity in graph searching. *J. Algorithms* 12: 239–245, 1991.
5. L. Blin, P. Fraignaud, N. Nisse, and S. Vial. Distributed chasing of network intruders. *Proc. 13th Colloquium on Structural Information and Communication Complexity* (SIROCCO'06), 2006.
6. R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers* VI(5):

72–78, 1967.

7. P. Flocchini, M.J. Huang, and F.L. Luccio. Contiguous search in the hypercube for capturing an intruder. *Proc. 19th IEEE Int. Parallel Distributed Processing Symp.* (IPDPS), 62-71, 2005.
8. P. Flocchini, M.J. Huang, and F.L. Luccio. Decontamination of chordal rings and tori. *Proc. 8th Workshop on Advances in Parallel and Distributed Computational Models*, 2006.
9. P. Flocchini, F.L. Luccio, and L.X. Song. Size optimal strategies for capturing an intruder in mesh networks. *Proc. Int. Conf. on Communications in Computing* (CIC 05), 200-206, 2005.
10. F. Fomin, D. Thilikos, and I. Todineau. Connected graph searching in outerplanar graphs. *Proc. 7th Int. Conf. on Graph Theory* (ICGT 05), 2005.
11. P. Fraigniaud, and N. Nisse Connected treewidth and connected graph searching. *Proc. 7th Latin American Symposium on Theoretical Informatics* (LATIN'06), 479-490, 2006.
12. M.S. Greenberg, J.C. Byington, and D. G. Harper. Mobile agents and security, *IEEE Communication Magazine*, 36(7): 76-85, 1998.
13. N. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6): 345–350, 1992.
14. L. Kirousis and C. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2): 205–218, 1986.
15. A. Lapaugh. Recontamination does not help to search a graph. *Journal of the ACM* 40(2): 224–245, 1993.
16. 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.
17. S. Neufeld. A pursuit-evasion problem on a grid. *Information Processing Letters*, 58(1): 5–9, 1996.
18. R. Oppliger. Security issues related to mobile code and agent-based systems. *Computer Communications*, 22(12): 1165-1170, 1999.
19. T. Parson. Pursuit-evasion in a graph. *Theory and Applications of Graphs*, Lecture Notes in Mathematics, Springer-Verlag, 426–441, 1976.
20. D. Peleg. Local majorities, coalitions and monopolies in graphs: A review. *Theoretical Computer science*, 231-257, 2002.
21. K. Schelderup and J. Ines. Mobile agent security - Issues and directions. *Proc. 6th Int. Conf. on Intell. and Services in Networks*, LNCS 1597, pp. 155-167, 1999.
22. I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, 1992.
23. M. Yamamoto, K. Takahashi, M. Hagiya, and S.-Y. Nishizaki. Formalization of graph search algorithms and its applications. *LNCS* 1479, Springer, 1998.