Network Decontamination with Local Immunization

Fabrizio Luccio[1], Linda Pagli[1], Nicola Santoro[2]

[1]Università di Pisa,
Dipartimento d'Informatica
{luccio,pagli}@di.unipi.it

[2] Carleton University,
School of Computer Science
santoro@scs.carleton.ca

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, avoiding any recontamination of disinfected areas, and minimizing the amount of agents' movements across the network. In all the existing literature, it is assumed that the immunity level of a disinfected site is nil.
In this paper we consider the network decontamination problem under a new model of immunity *to recontamination: we consider the case when a disinfected vertex, after the cleaning agent has gone, will become recontaminated only if a weak majority of its neighbours are infected. We study the effects of this level of immunity on the number of antiviral agents necessary to decontaminate the entire network. We focus on tori and on trees, and establish lower-bounds on the team size; we also establish lower bounds on the number of moves performed by an optimal-size time of cleaners. We design and present strategies for disinfecting tori and trees; we prove that these strategies are optimal in terms of both team size and number of moves. In particular, the upper and lower bounds are are tight for tree networks and for synchronous tori; the bounds are within a constant factor of each other in the case of asynchronous tori.*

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 (*harful hosts*), and the presence of mobile agents that can harm the network (*harmful agents* or *intruders*) (e.g., see [10, 16, 19]). 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 nework 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*), able to disinfect visited sites, avoiding any recontamination of disinfected areas. The goal is to perform the task using as small a team of antiviral agents as possible and minimizing the amount of agents' movements across the network.

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 disinfect the whole network. The protocol must guarantee that, after a site has been disinfected, it will not be recontaminated; typically, contamination will reoccur if the virus returns to that site in absence of a system agent. Such a protocol will be called *monotone*.

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 [5].

Since then, the problem has been extensively studied under the name of *graph searching*. 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

1-4244-0054-6/06/$20.00

but not least, graph minors [3]. Viewed sometimes as a pebbling problem in graphs (e.g., [12]) or as a pursuit-evasion game (e.g., [17]), the graph searching problem also arises in VLSI design through its equivalence with the gate matrix layout problem [11]. Among the important results is that there are always optimal solutions that are *monotone*, i.e. that avoid recontamination [4, 13]. Contributions to related search problems can be found in [15, 20, 21] 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 [5, 17]. The removal of this assumption makes the previous solutions no longer valid [2].

The more general setting where agent 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 the team size necessary to decontaminate classes of networks, and on the development of monotone decontamination strategies for those classes. In particular, Flocchini *et al.* have studied *hypercubes* [6], *meshes* [8], and *chordal rings* and *tori* [7]; Fomin *et al.* have investigated *outerplanar* graphs [9].

1.3 Recontamination and Immunization

What the new investigations have in common with the old ones is not only the goal of avoiding recontamination but also the *recontamination model*, i.e., the rules that allow a disinfected site to become recontaminated. In fact, in all investigations, it is assumed that a disinfected 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 disinfected 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 [18]. 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 schems have clearly a higher level of resistance to viral recontamination. In fact, a disinfected vertex, after the cleaning agent has gone, will become recontaminated only if a (weak) majority of its neighbours are infected. In other words, the immunity level of a disinfected vertex is half of its degree.

1.4 Main Results

In this paper we consider the network decontamination problem under this new model of *immunity to recontamination*. We study the effects of this level of immunity in tori and in trees; and establish lower-bounds on the number of cleaners necessary for decontamination, and on the number of moves performed by an optimal-size team of cleaners. We design and present strategies for disinfecting tori and trees and prove that these strategies are optimal in terms of both team size and number of moves. In particular, the upper and lower bounds are are tight for tree networks and for synchronous tori; the bounds are within a constant factor of each other in the case of asynchronous tori. Among other results, we show that with local immunization only $O(1)$ agents are needed to decontaminate meshes and tori, regardless of their size; this must be contrasted with e.g. the $2\min\{n, m\}$ agents required to decontaminate a $n \times m$ torus without local immunization [7]. 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.

Due to space limitations, some of the proofs of lemmas and theorems are omitted.

2 Basic Properties

Let the network be represented by an 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 cannot 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 disinfecting meshes and trees, and prove lower and upper bounds to the number of agents and moves.

3 Disinfecting Toroidal Meshes.

We first consider networks whose structure is a *k-dimensional toroidal mesh*, with $k = 1$ (i.e., the mesh is a ring), $k = 2$, and $k = 3$.

For $k = 1$, two (in fact, 2^1) agents starting from any position and moving in opposite directions are trivially necessary and sufficient for disinfecting a network of any number of vertices. This case does not differ from the standard operation of cleaning agents on a ring. The real novelty is for 2 and 3-dimensional meshes, where respectively 2^2 and 2^3 agents are necessary and sufficient, as we show below.

We recall that a 2-dimensional toroidal mesh of $m \times n$ vertices is a mesh where each vertex $v_{i,j}$, with $0 \leq i \leq m-1$, $0 \leq j \leq n-1$, is connected to the four vertices $v_{i-1,j}$, $v_{i+1,j}$, $v_{i,j-1}$, $v_{i,j+1}$, where the operations on indices i and j are respectively done mod$m$ and mod n. For any vertex v we have $d(v) = 4$, $m(v) = 3$. A lower bound on the number of agents needed for a 2-dimensional toroidal mesh is given in the following:

Theorem 1. *For disinfecting a 2-dimensional toroidal mesh of $m \times n$ vertices:*
1. at least four agents are necessary;
2. four agents must perform at least $m \times n$ moves.

The lower bounds of Theorem 1 hold even if the system is fully *synchronous*, i.e. even if the agents are perfectly synchronized in ther actions and movements. Indeed, for synchronous systems the bounds of Theorem 1 are strict, as shown by the following algorithnm for monotone decontamination.

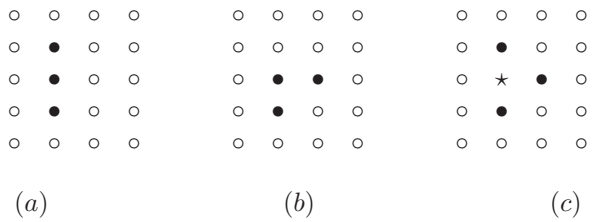

Figure 1. Three agents starting from the same vertex can initially move as shown in (a), or (b), or to symmetric positions. The next move can only produce configuration (c), or symmetric, after which no agent can move.

Algorithm 1. *Disinfecting an $m \times n$ 2-dimensional toroidal mesh with four agents A_0, A_1, A_2, A_3.*

1. start with the agents in vertex $v_{0,0}$;
2. with four total steps move A_1 to $v_{0,1}$, A_2 to $v_{1,0}$, A_3 to $v_{1,1}$, leaving A_0 in $v_{0,0}$. Note that all the agents can now move to white neighbours by Lemma 1 (Figure 2 (a));
3. move A_1 to $v_{0,2}$, then A_3 to $v_{1,2}$ (Figure 2 (b)); continue moving A_1 and A_3 alternatively on rows 0 and 1 until they reach $v_{0,n-1}$ and $v_{1,n-1}$, respectively. Now all the vertices in rows 0 and 1 are clean;
4. move A_2 to $v_{2,0}$ and A_3 to $v_{2,n-1}$ (Figure 2 (c)). Note that A_3 is adjacent to A_2 by a toroidal connection, hence it can be moved to a white neighbour by Lemma 1;
5. move A_3 to $v_{2,n-2}$, then to $v_{2,n-3}$, and so on until $v_{2,1}$, is reached (Figure 2 (d));
6. move A_2 to $v_{3,0}$ and A_3 to $v_{3,1}$; proceed moving A_3 along row 3, and so on, until the whole network has been visited.

Using Algorithm 1 we have:

Theorem 2. *Four synchronous agents can monotonically disinfect a 2-dimensional toroidal mesh of $m \times n$ vertices making $m \times n$ moves.*

If the system is not synchronous, the same strategy will work employing just one additional agent, whose

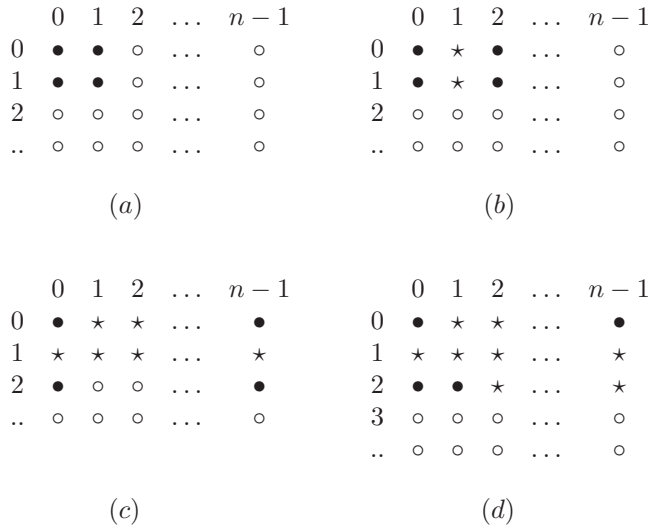

Figure 2. Illustration of Algorithm 1.

only function is to synchronize the movements of the other agents. In particular, this agent will act as a moving *token*: each of the other agents will perform its move according to Algorithm 1 only upon being visited by the token. It is not difficult to verify that this can be accomplished by at most doubling the number of moves:

Theorem 3. *Five asynchronous agents can monotonically disinfect a* 2*-dimensional toroidal mesh of* $m \times n$ *vertices making at most* $2m \times n$ *moves.*

Let us now turn our attention to the 3-dimensional case. We recall that a 3-dimensional toroidal mesh of $m \times n \times p$ vertices is a mesh where each vertex $v_{i,j,k}$, with $0 \leq i \leq m-1, 0 \leq j \leq n-1, 0 \leq k \leq p-1$, is connected to the six vertices $v_{i-1,j,k}$, $v_{i+1,j,k}$, $v_{i,j-1,k}$, $v_{i,j+1,k}$, $v_{i,j,k-1}$, $v_{i,j,k+1}$, where the operations on indices i, j, and k are respectively done mod$m$, mod$n$, and mod p. For any vertex v we have $d(v) = 6$, $m(v) = 4$. A lower bound on the number of agents needed for disinfecting such a network is given by:

Theorem 4. *For disinfecting a* 3*-dimensional toroidal mesh of* $m \times n \times p$ *vertices:*
1. at least eight agents are necessary;
2. eight agents must perform at least $m \times n \times p + 3$ *moves.*

The lower bounds of Theorem 4 hold even if the system is fully *synchronous*, i.e. even if the agents are perfectly synchronized in ther actions and movements. Indeed, for synchronous systems the first bound of Theorem 4 is strict and the second bound is almost strict.

For this purpose we give the following algorithnm for monotone disinfecting, as an extension of Algorithm 1 in three dimensions.

Algorithm 2. *Disinfecting an* $m \times n \times p$ 3*-dimensional toroidal mesh with eight agents* A_0, A_1, ..., A_7.

1. start with the agents in vertex $v_{0,0,0}$;
2. with twelve total steps move A_1, A_2, ..., A_7 to $v_{0,0,1}$, $v_{0,1,0}$, ..., $v_{1,1,1}$ respectively, leaving A_0 in $v_{0,0,0}$;
3. move A_2, A_3, A_6, A_7 alternatively one position each, to vertices with increasing value of the index j, until they reach $v_{0,n-1,0}$, $v_{0,n-1,1}$, $v_{1,n-1,0}$, $v_{1,n-1,1}$ respectively. Now all the vertices in rows (0,j,0), (0,j,1), (1,j,0), (1,j,1), for $0 \leq j \leq n-1$, are clean;
4. move A_4, A_5, A_6, A_7 one position each, to vertices with $i = 2$. A_6, A_7 are again adjacent to A_4, A_5 by a toroidal connections, hence they can be moved to white neighbours by Lemma 1. Therefore, move A_6, A_7 alternatively one position each, to vertices with decreasing value of the index j, until they meet A_4, A_5;
5. move A_4, A_5, A_6, A_7 one position each, to vertices with $i = 3$. Then move A_6, A_7 to vertices with increasing value of the index j, and repeat steps 4. and 5. until all the vertices with $k = 0$ and $k = 1$ have been visited. If m is odd move back A_7 of $n-2$ steps in direction j, to the vertex $v_{m-1,n-1,1}$;
6. now the four agents A_1, A_3, A_5, A_7 are in the vertices $v_{0,0,1}$, $v_{0,n-1,1}$, $v_{m-1,0,1}$, $v_{m-1,n-1,1}$ respectively, forming a square of unitary side. Move these agents one position each onto the plane $k = 2$;
7. note that each vertex v in the plane $k = 2$ is adjacent to a clean vertex in $k = 1$, then three clean neighbours in $k = 2$ are sufficient to protect v. Let the agents A_1, A_3, A_5, A_7 traverse the whole plane $k = 2$ with the previous Algorithm 1. Then, if n is odd, move back A_7 of $n-2$ steps;
8. again the four agents A_1, A_3, A_5, A_7 form a square of unitary side. Move these agents to the next plane $k \leftarrow k + 1$ and repeat steps 7. and 8. until the whole network has been visited.

We have:

Theorem 5. *Eight synchronous agents can monotonically disinfect a* 3*-dimensional toroidal mesh of* $m \times n \times p$ *vertices. For* m *and* n *even the agents make* $m \times n \times p + 4$ *moves.*

Note that if m and/or n is odd some extra moves are done over the bound $m \times n \times p + 4$, as indicated in steps 5. and 7. of the algorithm.

As for the 2-dimensional case, if the system is not synchronous, the same strategy will work employing just one additional agent, whose only function is to synchronize the movements of the other agents. In particular, this agent will act as a moving *token*: each of the other agents will perform its move according to Algorithm 2 only upon being visited by the token. Again, it is not difficult to verify that this can be accomplished by at most doubling the number of moves:

Theorem 6. *Nine synchronous agents can monotonically disinfect a 3-dimensional toroidal mesh of $m \times n \times p$ vertices. For m and n even the agents make at most $2m \times n \times p + 8$ moves.*

Generalizing the previous designs, we have the following result:

Theorem 7. *2^k synchronous agents (resp., $2^k + 1$ asynchronous agents) can monotonically disinfect a k-dimensional toroidal mesh.*

The detailed description of the algorithm proving the result will be in the final version of the paper.

A matching lower bound still has still to be proved for $k > 3$; we do pose the following

Conjecture 1. *To disinfect a k-dimensional toroidal mesh, with $k \geq 1$, 2^k agents are necessary.*

4 Disinfecting Tree Networks

Tree structured networks have been carefully studied without local immunity [1, 2, 14]. As one may expect, the results in our model are different. Let us first establish a lower bound on the number of moves, same for all trees. Let δ be longest path between two leaves. $|\delta|$ (number of edges in δ) is the *diameter* of the tree. We have:

Theorem 8. *For disinfecting a tree of n vertices and diameter $|\delta|$, at least $2(n-1)-|\delta|$ moves are necessary.*

We study tree disinfecting starting from binary trees.

4.1 Decontaminating trees with vertex degree ≤ 3.

The number of agents needed for disinfecting a tree T of n vertices depends on the maximum vertex degree *d-max* $= max_{v \in T}(d(v))$. The case *d-max* $= 1$ is non interesting because it implies $n = 1, 2$. For *d-max* $= 2$, 3 we shall prove that one agent suffices, while for *d-max* ≥ 4 the number of agents is a function of the tree structure.

The case *d-max* $= 2$ is elementary, because the tree is a chain and can be monotonically visited by an agent starting at one exterme. The diameter is $n - 1$, and the agent performs $n - 1$ moves consistently with Theorem 8. The case *d-max* $= 3$ is much more interesting, because it includes the whole family of binary trees (e.g., rooted or unrooted, ordered or unordered, etc.). The results here are very different from the ones of [1] where $\log_2 n$ agents were needed, in fact, we now show how a single agent can disinfect one such a tree.

The agent A traverses T starting from a leaf. For each vertex v reached by A, let $F(v)$ be the vertex from which A has come, and $T1(v)$, $T2(v)$ be the two other vertices connected to v, in any order, to which A may go. If anyone of these vertices does not exist, we give it value *null*. We pose:

Algorithm 3. *Disinfecting a tree T of n vertices with* d-max $=$ *3 (i.e., a binary tree of any sort) using one agent A.*

1. **let** x be a leaf of T; /x is the starting vertex
   $v \leftarrow x$; $c \leftarrow 0$; /c is the number of visited vertices
2. $REP(v)$; /procedure call
3. **procedure** $REP(v)$
   if $v = null$ **then return**;
   move A to v; $c \leftarrow c + 1$; **if** $c = n$ **then halt**;
   $REP(T1(v))$; $REP(T2(v))$; /recursive calls
   move A to $F(v)$.

The functioning of Algorithm 3 can be followed in Figure 3, where the numbers on the edges indicate the ordering of the moves in the procedure REP. The number to the left corresponds to the move: **move** A to v; the number to the right, if any, corresponds to the inverse move: **move** A to $F(v)$. The latter move is not done on some edges, because the algorithm halts when the n-th vertex is reached. We have:

Theorem 9. *One agent can monotonically disinfect a tree T of n vertices with* d-max $=$ *3, making $2(n-1)-|\lambda|$ moves, where λ is a path connecting two arbitrary leaves of T.*

If the path λ of Theorem 9 is chosen as the longest path between two leaves, then the lower bound of Theorem 8 is met. This is the case of Figure 3 where $n = 14$, $|\lambda| = 7$, and 21 moves are done.

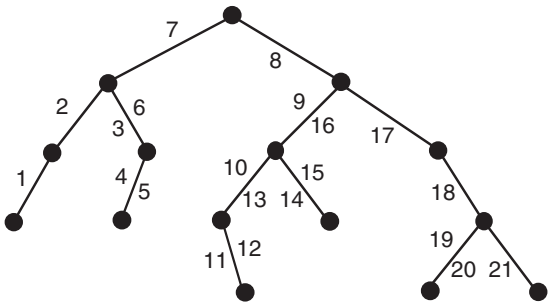

Figure 3. Edge traversal in Algorithm 3. The path λ of Theorem 9 is traversed with the moves 1, 2, 7, 8, 17, 18, 21.

4.2 Disinfecting General trees.

For trees with *d-max* > 3 the analysis is much more difficult than in the binary case. For one such a tree T, let v be any vertex, and consider a group of agents entering v for the first time. The agents may come from outside, i.e. v is the starting vertex, or from an edge e of T. In both cases v is the root of a subtree $T' \subseteq T$ whose vertices are all white except for v (in particular $T' = T$ if v is the starting vertex). Following the approach of [1], we assign to v an *agent-number* $N(v)$ indicating the minimum number of agents needed for disinfecting T' if entering from v. We shall see that $N(v)$ is independent of the origin of the agents, that is, external or e.

Due to the requirements of our computational model, the agents in v can disinfect T' using the following principle. First note that the other extreme of e, if any, is clean because the agents came from it. Assume that v has r children $u_1, ..., u_r$ in T' with agent-numbers $N(u_1) \geq N(u_2) \geq ... \geq N(u_r)$ (leaves have agent-number 1). As in [1], if $N(u_1) > N(u_2)$ then we set $N(v) = N(u_1)$. In fact one agent can be kept in v while the others traverse the subtrees rooted in $u_2, ..., u_r$; then all the $N(v)$ agents can traverse the subtree rooted in u_1. In the present case, however, we can use $N(v) = N(u_1)$ agents at v even if $N(u_1)$ is equal to the agent-number of other children of v, provided a majority of children u_i has $N(u_i) < N(u_1)$. In fact all such subtrees can be disinfected by less than $N(v)$ agents, keeping one agent in v; then all the agents can move to the other subtrees, leaving v protected by the clean vertices u_i already visited. Without such a majority $N(u_1) + 1$ agents are needed, where the extra agent must be always kept in v.

For implementing the above principle we first give a new Algorithm 4 for computing the agent-numbers. Then we shall use this numbering in the disinfecting process. Algorithm 4 starts assigning value 1 to the leaves, and initial value 0 to the internal vertices. Then procedes assigning increasing values $k = 1, 2, ...$ to the internal vertices, in consecutive rounds. At any round k the vertices in the following four sets are processed:

$S_1 = \{v \in T \,|\, N(v) = 0,\ d(v) = 2,\ v$ has neighbours u_1, u_2
with $N(u_1) = k,\ N(u_2) = 0\}$

$S_2 = \{v \in T \,|\, N(v) = 0,\ d(v) = 3,\ v$ has neighbours u_1, u_2, u_3
with $k = N(u_1) \geq N(u_2) > 0,\ N(u_3) = 0\}$

$S_3 = \{v \in T \,|\, N(v) = 0,\ d(v) \geq 4,\ v$ has neighbours $u_1, ..., u_{d(v)}$
with $k = N(u_1) \geq N(u_2) \geq ... \geq N(u_{d(v)-1}) > 0,\ N(u_{d(v)}) = 0\}$;

$S_4 = \{v \in T \,|\, N(v) = 0,\ d(v) \geq 2,\ v$ has neighbours $u_1, ..., u_{d(v)}$
with $k = N(u_1) \geq N(u_2) \geq ... \geq N(u_{d(v)}) > 0\}$.

These sets change within the same round because the new value k is assigned to some of their vertices. A vertex v getting $N(v) = k$ is then eliminated from its set (where $N(v) = 0$ is required), but may bring in another vertex w of which v is a neighbour. We have:

Algorithm 4. *Assigning agent-numbers to the vertices of a tree T with* d-*max* > 3.

1. $\forall v \in T$ **do if** $d(v) = 1$ **then** $N(v) \leftarrow 1$ **else** $N(v) \leftarrow 0$;
2. $k \leftarrow 0$;
3. **repeat** $k \leftarrow k + 1$;
   while $\exists v \in S_1 \cup S_2 \cup S_3 \cup S_4$ **do**
   if $v \in S_1 \cup S_2$ **then** $N(v) \leftarrow k$;
   if $v \in S_3 \cup S_4$ **then let** $s = \lceil d(v)/2 \rceil + 1$;
   /condition $v \in S_4$ occurs when the last agent-number is assigned
   if $N(u_1) > N(u_s)$ **then** $N(v) \leftarrow k$ **else** $N(v) \leftarrow *$;
   $\forall v \in T$ with $N(v) = *$ **do** $N(v) \leftarrow k + 1$
   until all $v \in T$ have $N(v) > 0$.

We are now ready to present the disinfecting algorithm, using a number of agents equal to the maximal value of $N(v)$. The agents start from a vertex chosen as the root of T. For any vertex v except the root, $F(v)$ denotes the vertex from which the visiting agents came.

Algorithm 5. *Disinfecting a tree T of n vertices with* d-$max > 3$ *after the agent-numbers have been computed, using $N = max_{v \in T}(N(v))$ agents $A_1, ..., A_N$.*

1. **transform** T into an ordered rooted tree as follows:

   choose a vertex r with $N(r) = N$ as the root;

   for each internal vertex v, order its s children $u_1, ..., u_s$ such that

   $$N(u_1) \leq N(u_2) \leq ... \leq N(u_s);$$

2. $c \leftarrow 0$; /c is the number of visited vertices

3. $REP2(r)$; /recursive procedure call starting from the root

4. **procedure** $REP2(v)$

   if $v = r$ **then start** with $A_1, ..., A_N$ in v **else move** $A_1, ..., A_{N(v)}$ to v;

   $c \leftarrow c + 1$; **if** $c = n$ **then halt**;

   if v has s children $u_1, ..., u_s$ **then for** $i \leftarrow 1$ **to** s **do** $REP2(u_i)$;

   move $A_1, ..., A_{N(v)}$ to $F(v)$.

Note that in Algorithm 5 we could have started from a leaf x instead of the root r, as done for binary trees. However, starting from x would increase the total number of agent moves, because all the N agents necessary at r should travel from x to r, while just one agent is needed at x. As before we save on the last moves because the algorithm halts when an agent reaches the rightmost leaf ℓ, so that the last command, **move** $A_1, ..., A_{N(v)}$ to $F(v)$, is not executed on the returning path from ℓ to r. With the agent-numbers $N(v)$ defined as before, the root r and the rightmost leaf ℓ determined with the tree ordering of Algorithm 5, we immediately have:

Theorem 10. *$N = max_{v \in T}(N(v))$ agents can monotonically disinfect a tree T of n vertices with* d-$max > 3$*, making $2\sum_{v \in T - \{r\}} N(v) - \sum_{v \in \lambda - \{r\}} N(v)$ moves, where λ is the path connecting r to ℓ.*

On one hand Theorem 10 is very general; on the other hand the given numbers of agents and moves depend on a previous computation of all $N(v)$, so that no immediate result can be stated on a generic tree. It is not even easy to establish lower bounds for this problem, as shown in the example of Figure 4 where an unexpected situation occurs. Although we have $N(r) = 4$ at the root, we could start with only three agents A_1, A_2, A_3 there. Keep A_1 at r and move A_2, A_3 to the child a; keep A_2 at a and move A_3 to e and back; move A_2, A_3 back to r. Vertices a and e are clean, although the subtree rooted at f has not been visited yet. Repeat the same moves on the subtree rooted at d, so that d and g become clean and the agents are brought back to r. Now the three agents can all be moved to any of the subtrees of r, that require three agents each to be disinfected, because vertex r is protected by the clean vertices a and d. Saving one agent is attained by interrupting subtree traversals, then completed at later steps. Obviously this is not always possible, however, the example shows that the bound of Theorem 10 is not strict in general.

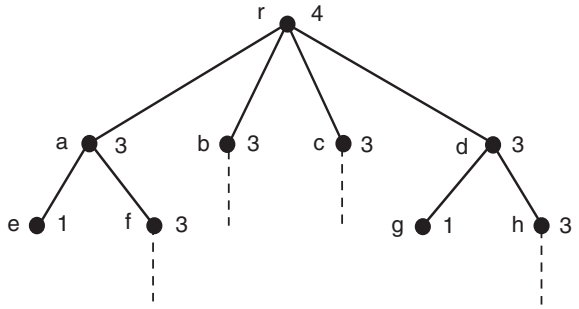

Figure 4. Portion of a tree with $d\text{-}max = 4$, that can be repaired with $N(r) - 1 = 3$ agents at the root. Agent-numbers are shown on the vertices.

More stringent results can be given for specific classes of trees with *d-max* > 3. Let us examine the case of *complete k-ary trees* with $k \geq 3$, that is rooted trees where all the internal vertices have k children, and all the leaves are at the same level $h = \lfloor \log_k n \rfloor$ (h is the height of the tree; the root is at level 0). Applying Algorithm 4 we have that the agent-numbers are increasing from 1 to $N = \lceil \log_k n \rceil$ passing from the leaves to the root. Then we immediately have from Theorem 10:

Corollary 1. *$N = \lceil \log_k n \rceil$ agents can monotonically disinfect a complete k-ary tree T of n vertices, with $k \geq 3$.*

However, one agent less can be used for $k = 3$. We have:

Corollary 2. *$N = \lceil \log_3 n \rceil - 1$ agents can monotonically disinfect a complete 3-ary tree of n vertices.*

We now prove that the upper bounds of Corollaries 1 and 2 are strict, due to the constrained structure of k-ary trees. For this purpose we prove a preliminary result:

Lemma 3. *Let T be a k-ary tree, $k \geq 3$, and let x be the root of a subtree $X \subseteq T$ composed of white vertices. If $M < N(x)$ agents enter in x from its parent, or start from x, then:*
(1) not all the vertices of X can be visited by these agents;
(2) not all the agents can eventually leave from X.

We then have:

Theorem 11. *For disinfecting a complete k-ary tree T of n vertices, with $k \geq 3$, at least $\lceil \log_k n \rceil - 1$ agents are necessary.*

Theorem 11 proves that the upper bound of Corollary 2 for 3-ary trees is strict. However, also the general bound of Corollary 1 is strict, if limited to k-ary trees with $k \geq 4$. In fact we have:

Corollary 3. *For disinfecting a complete k-ary tree T of n vertices, with $k \geq 4$, at least $\lceil \log_k n \rceil$ agents are necessary.*

Using the traversal orders implicitly assumed by Corollaries 1 and 2, the total number of steps made by the agents follow immediately. No significant lower bound on these numbers have been found as yet.

References

[1] L. Barriere, 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. Barriere, P. Fraignaud, N. Santoro, and D. Thilikos. Searching is not jumping. *Proc. 29th Int. Workshop on Graph-Theoretic Concepts in Computer Science* (WG03), 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] R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers* VI(5): 72–78, 1967.

[6] 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.

[7] P. Flocchini, M.J. Huang, and F.L. Luccio. Decontamination of chordal rings and tori. These Proceedings, 2006.

[8] 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), 200-206, 2005.

[9] F. Fomin, D. Thilikos, and I. Todineau. Connected graph searching in outerplanar graphs. *Proc. 7th Int. Conf. on Graph Theory* (ICGT 05), 2005.

[10] M.S. Greenberg, J.C. Byington, and D. G. Harper. Mobile agents and security, *IEEE Communication Magazine*, 36(7): 76-85, 1998.

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

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

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

[14] 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.

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

[16] R. Oppliger. Security issues related to mobile code and agent-based systems. *Computer Communications*, 22(12): 1165-1170, 1999.

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

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

[19] 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.

[20] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, 1992.

[21] M. Yamamoto, K. Takahashi, M. Hagiya, and S.-Y. Nishizaki. Formalization of graph search algorithms and its applications. *LNCS* 1479, Springer, 1998.