Size Optimal Strategies for Capturing an Intruder in Mesh Networks *

Paola Flocchini
School of Information Technology and Engineering
University of Ottawa, Canada
flocchin@site.uottawa.ca

Flaminia L. Luccio
Dipartimento di Matematica e Informatica
Università di Trieste, Italy
luccio@dsm.univ.trieste.it

Lisa Xiuli Song
School of Computer Science
Carleton University, Ottawa, Canada
lxsong@scs.carleton.ca

Abstract

In this paper we consider the problem of capturing an intruder in a networked environment. The intruder is defined as a mobile entity that moves inside the network and escapes from a team of collaborative agents. The agents travel asynchronously and try to coordinate their moves in order to capture the intruder. We consider as a topology a $m \times n$-Mesh; we first show some lower bounds on the number of agents, number of moves, and time required to capture the intruder. We then consider two models: one in which the only knowledge that an agent has is local to the node it resides, the other in which agents have visibility, i.e., are allowed to "see" the state of their neighbours. One of our goals is to understand what is the impact of visibility on complexity. All our strategies, in both models, are optimal in terms of number of agents employed and number of moves performed; moreover, the visibility power enables the agents to drastically speed up computation, and thus we also obtain an optimal time bound.

Keywords: Mobile agents, Intruder Capture, Graph Search, Mesh.

1. Introduction

1.1. The Problem

Consider a network where nodes represent hosts and edges represent connections between hosts. An *intruder* is a mobile entity that moves from host to host. The problem is to devise a strategy for deploying a team of collaborative agents to capture the intruder.

We assume that the agents start from the same node (called the *home base*) and behave *asynchronously*, i.e., all actions they perform take a finite but unpredictable amount of time. Possible actions are agent communication through a *whiteboard* located at each node (i.e., read and write operations), local computations (each agent is provided with an $O(\log n)$ local memory), and movements from a host to a neighbouring host.

Since we are interested in worst case behaviors, we assume that the intruder is very powerful; in fact it can " see" where the other agents are located, and can then escape from them by moving arbitrarily fast inside the network.

It is easy to see that the intruder capture problem can be equivalently formulated in terms of a *decontamination* problem. If the intruder can "hide" only on nodes, the node-decontamination can be defined as follows: each node of the network can be in one of three possible states: *clean*, *contaminated*, *guarded*. Initially all nodes are *contaminated* except for the homebase (which is *guarded*, i.e., an agent lies on it). A node is *clean* when an agent passes by it and all its neighbouring nodes are clean or guarded, *contaminated* otherwise. The node-decontamination problem consists of having all the nodes simultaneously *clean*. The edge-decontamination can be defined similarly and corresponds to the case when the intruder can "hide" on edges.

In this paper we are interested in solving both variations of the problem: node and edge decontamination.

We are also interested in complexity issues, i.e., devising efficient strategies, where efficiency is measured in terms of the size of the agent team, the number of moves agents have to perform and time.

* Work partially supported by NSERC and by Abstract Interpretation Design and Applications, MIUR COFIN Project.

1.2. Related Work

A variation of our problem has been widely studied in the literature under the name of *graph search* (or *decontamination*) problem. This problem was first introduced by Breish [3] and Parson [10, 11]; in that context it consisted of having a system of tunnels represented by edges of a graph and these tunnels were initially *contaminated* by a gas, and had to be simultaneously decontaminated by a sequence of clean operations (or moves) executed by a (minimal) set of *searchers*.

Several variations of the problem have been studied: among them, node search and edge search (see, e.g., [4, 6, 7, 8, 9, 12]). The aim is to find a strategy that minimizes the number of searchers and leads the graph to a state in which all nodes (or nodes and edges) are simultaneously decontaminated. The *size* of the searching team is called *node-search number* $ns(G)$ or *edge-search number* $es(G)$) and the determination of the optimal size is an NP-complete problem in most topologies.

It is easy to see that graph search, intruder detection, and decontamination are equivalent problems.

In all the graph search variations studied in the literature searchers may be placed and removed from any node of the searching graph, i.e., they are allowed to *"jump"* while they perform the searching task. However, in a networked environment searchers may be contaminated when they *"jump"* as during the execution they may traverse a contaminated environment. Therefore, it will be impossible to perform the network cleaning task by using those damaged searchers.

A variation of the node/edge-search problem is called the *contiguous search problem* [1, 2] and adds the requirement the searchers perform *contiguous* searches in a given network instead of making any *jump* moves. This problem is harder than the non-contiguous one as in [2] it has been proved that the contiguous searching number is always greater or equal to the non-contiguous searching number. Finding the contiguous searching number is still an NP-complete problem for general graphs; few specific topologies have been studied, for example, it has been shown that it can be solved in linear time in trees [1], moreover, strategies and upper bounds have been studied only in hypercubes [5].

Formally, the *contiguous search problem* defines two additional stricter conditions: (1) searchers may move only to neighbouring nodes; (2) at any time all cleaned links form a connected subgraph of the original graph.

Our Results. In this paper we consider the contiguous node and edge decontamination problem in a $m \times n$-Mesh (without loss of generality, let $m \leq n$). We consider two different models and accordingly propose two strategies. In the first model the only knowledge that an agent has is the information written on the local whiteboard at its current location; in the second model agents can "see" also the state of their neighbours; i.e., they can see whether a neighbouring node is guarded by an agent, whether it is clean, or contaminated. Notice that, in some mobile agent systems, the visibility power could be easily achieved by "probing" the state of neighbouring nodes before making a decision. Our first strategy relies on a coordinator that organizes the moves of the other agents, in the second one instead, the agents take advantage of their visibility power and can move autonomously.

Table 1 Search in a $m \times n$-Mesh

Model		*Synchronizer*	*Visibility*
Node	Agents	$m+1$	m
	Moves	$\frac{m^2+4mn-5m-2}{2}$	$\frac{m^2+2mn-3m}{2}$
	Time	$mn-2$	$m+n-2$
Edge	Agents	$m+1$	$m+1$
	Moves	$\frac{m^2+4mn-3m-2}{2}$	$\frac{m^2+4mn-5m}{2}$
	Time	$mn+m-2$	$mn-m$

We also show that, regardless of the visibility assumption and regardless of the type of search (node versus edge), at least m agents, mn moves and $m+n-2$ time units are needed to solve the problem. Our upper bounds are summarized in Table 1.

One of our goals is to understand the power of visibility, i.e., to determine whether such an assumption can indeed improve the performances of solutions to the problem and how. In this respect, we have observed that: with our strategies the visibility assumption allows to drastically decrease the time complexity only in the case of node search, obtaining an optimal time complexity; while it does not help much in the case of edge search. Furthermore, in the visibility model the size of the team is matching exactly the theoretical lower bound, while in the case of the other setting, it differs by one agent.

Due to space limitations, some of the proofs are only sketched and some omitted, for full details see [13].

2. Definitions and Terminology

Definition 1 *A* $m \times n$ mesh *is a graph* $M_{m,n} = (V, E)$ *with* $v_{i,j} \in V$, $0 \leq i \leq m-1$, $0 \leq j \leq n-1$, *and* E *contains exactly the edges* $(v_{i,j}, v_{i,j+1})$, $j \neq n-1$, *and* $(v_{i,j}, v_{i+1,j})$, $i \neq m-1$.

Without loss of generality, in the following we will always assume $m \leq n$. We denote with r_i, $0 \leq i \leq m-1$ the i-th row (i.e., $v_{i,j}$, $0 \leq j \leq n-1$), and with c_j, $0 \leq j \leq n-1$, the j-th column (i.e., $v_{i,j}$, $0 \leq i \leq m-1$). The value $v_{i,j}$ represents the node label.

Directions follow the standard compass-labelling, i.e., to move from $v_{i,j}$ to $v_{i-1,j}$ (for $i-1 \geq 0$) we follow the $NORTH$ direction, to $v_{i+1,j}$ (for $i+1 \leq m-1$) we follow the $SOUTH$ direction, to $v_{i,j-1}$ (for $j-1 \geq 0$) we follow the $WEST$ direction, $v_{i,j+1}$ (for $j+1 \le n-1$) we follow the $EAST$ direction.

Agents and nodes. The agents are processes that move autonomously inside the mesh from a host to a neighbouring one. They have computing capabilities, $O(\log n)$ bits of local memory, and obey the same set of behavioral rules. They are asynchronous in the sense that movements and computations take a finite but otherwise unpredictable amount of time. We consider two models: in the first one searchers do not have visibility power, i.e., cannot see whether their neighboring nodes and edges are clean, hence are coordinated by special searcher called *Synchronizer*; in the second one searchers have visibility power hence can autonomously decide where to move. Initially all nodes are identical except a special one called the *Home Base*, which is the upper-left most corner in the mesh; during execution a node x is *guarded* if an agent is on x, *clean* if an agent passed by x and all its neighbours are either clean or guarded, *contaminated* otherwise. An edge (x, y) is *clean* if it has been traversed by an agent and x and y are either clean or guarded, else it is *contaminated*. At the end the intruder lying on a node or edge is captured if it is surrounded by agents. All nodes have a local $O(\log n)$ bits storage called *Whiteboard* accessed in mutual exclusion by agents in order to read and write messages. Initially the whiteboard contains the node label and the label of the incident ports.

The strategy. We will consider only *monotone* strategies, i.e., the ones for which when a node is clean, it is never going to be contaminated again.

Our complexity measures will be: The *number of searchers*, i.e., the total number of agents that are needed to perform the searching task; the *number of searching moves*, i.e., the total number of moves performed by searchers during the cleanup of the graph; the *time*, i.e., the time units spent to clean the graph. Note that as the system is asynchronous, we will measure ideal time, that is, when measuring time we will assume that it takes one unit of time for an agent to traverse an edge.

3. Lower bounds

In this section we provide lower bounds on the number of agents, on the time, and on number of moves required to search the mesh.

We assume we have a $m \times n$-Mesh with $m \le n$ and that the mesh is labelled with Compass-Labelling. The network and mobile agents are reliable, moreover all agents start from the upper-left most node, the *Home Base*.

Theorem 1 *The solution of the contiguous node-search problem in a $m \times n$-Mesh ($m \le n$) requires at least m searchers.*

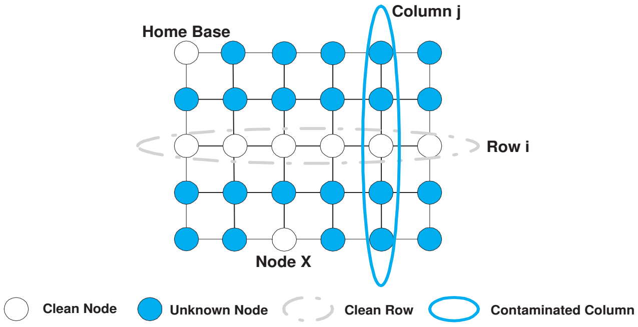

Figure 1. Proof of Theorem 1

PROOF By contradiction let us assume that $m-1$ searching agents are sufficient to solve the contiguous node-search problem in a $m \times n$-Mesh ($m \le n$) starting from the left-upper most corner.

Consider the first time, during the cleanup of a mesh, that an agent reaches a arbitrary node x in the last row, i.e. x is now guarded but all the other nodes in that last row are still contaminated. (See, e.g., node x in Figure 1.)

At most $m-1$ searching agents are available and the mesh is composed of m rows, thus there must be at least an empty row r_i $(0 < i < m-1)$. This row has to be clean, otherwise, if at least a node is contaminated, then the whole row r_i is contaminated as well. This however contradicts the contiguous assumption. In fact r_i would divide the mesh into two disconnected regions, one containing at least a clean (or guarded) node (the *Home Base*), and the other one containing at clean node (x) and at least a contaminated node (any node in the last row different from node x). Thus row r_i must be clean.

As the number of available agents is $m-1 < n$ then there must be at least an empty column c_j, where $(0 < j < n)$ and thus this column cannot contain x. However, at this instant of time c_j is contaminated as the node in its last row is contaminated (by construction the only clean node in this last row is x). Thus, the entire column c_j is contaminated but c_j intersects r_i, thus r_i cannot be a clean row. This contradicts what we have concluded before. Therefore, the contiguous node-search problem in a $m \times n$-Mesh requires at least m searchers. ∎

Theorem 2 *The solution of the contiguous node-search problem in a $m \times n$-Mesh ($m \le n$) requires at least $n+m-2$ time units.*

PROOF It directly follows from the fact that the agents are all initially located at the upper-left most corner of the mesh and at least one of them has to move to the bottom-right most corner in order to clean the entire mesh. Assuming one time unit per move, this agent will take at least $m+n-2$ time units to terminate. ∎

Theorem 3 *The solution of the contiguous node-search problem in a $m \times n$-Mesh ($m \leq n$) requires at least mn moves.*

PROOF It directly follows from the fact that the agents are all initially located at the upper-left most corner of the mesh and all nodes have to be visited by some agent. ∎

4. Node and Edge-Search without Visibility

We first consider the case when agents have only local knowledge and cannot "see" the state of their neighbouring nodes. We now illustrate the Searching Strategy.

We assume that, initially, all the searchers are located at node $v_{0,0}$ (the upper left most corner).

Initialization Phase: The *Synchronizer* starts from $v_{0,0}$; the other m searchers move to their starting point, i.e., a different position along the first column.

Cleaning Phase: The main idea is to coordinate the moves in such a way that all agents move from one column to the next one without creating "holes". This operation is divided into four phases followed one after the other until the whole mesh is clean.

Phase 1): The *Synchronizer* moves *SOUTH* from r_0 to r_n and forces the encountered searchers to execute one move *EAST*. Phase 2): The *Synchronizer* executes one move *EAST*. Phase 3): The *Synchronizer* moves *NORTH* from r_n to r_0 and forces the encountered searchers to execute one move *EAST*. Phase 4): The *Synchronizer* executes one move *EAST*.

Note that these steps perform node-search. For edge-search the only additional operation is the traversal of the last column (thus the cleaning of the related edges) *NORTH* or *SOUTH* depending, respectively, on the evenness or oddness of m. Note also that each node of the last column is guarded by an agent.

The formal description of the algorithm is given by algorithm SEARCH SYNCH in Table 2. The input is a contaminated mesh and the output is the same mesh after the cleaning. There are $m+1$ agents, initially all located at node $v_{0,0}$: the synchronizer is labelled as S, agent i, $0 \leq i \leq m-1$ as s_i.

Table 2 Search with a Synchronizer.

```
ALGORITHM SEARCH SYNCH

INITIALIZATION
for (i = m − 1; i ≤ 0; i − −) S tells s_i to move to v_{i,0};

CLEAN-UP
j = 0; i = 0;
while (j < (n − 2)) do while (i ≤ (m − 1)) do S moves to v_{i,j};
                                              S tells s_i move to v_{i,j+1};
                                              i+ = 1;
                   j+ = 1; i = m − 1;
                   while i ≥ 0 do S moves to v_{i,j};
                              S tells s_i move to v_{i,j+1};
                              i− = 1;
                   j+ = 1; i = 0;
if (j == (n − 2)) do while (i ≤ (m − 1)) do S moves to v_{i,j};
                                              S tells s_i move to v_{i,j+1};
                                              i+ = 1;
if edge-search then
    if (i == m) then S moves to v_{m−1,n−1};
                 for (k = m − 2; k ≥ 0; k − −) S moves to v_{k,n−1};
               else S moves to v_{0,n−1};
                 for (k = 1; k ≤ m − 1; k + +) S moves to v_{k,n−1};
```

Theorem 4 *Algorithm* SEARCH SYNCH *correctly solves the node/edge-search problem in a Mesh.*

PROOF To prove the correctness of the Algorithm SEARCH SYNCH , we need to prove that the intruder is captured and that the cleaning result is contiguous and monotone. We prove it by induction on the number of cleaned nodes.

1) *Base Case: the first column is clean.* Initially all the agents are located at node $v_{0,0}$ and this node is clean. During the initialization steps, *Synchronizer* S forces one at a time and by consecutive positions all the searchers to move *SOUTH* along the first column so to guard it entirely. Thus, the initialization (or cleaning) result is contiguous and monotone.

2) *Assumption Step:* Let's assume that at step i ($i > 0$) of the cleaning phase, the cleaning result is contiguous and monotone. In other words, at step i, all the clean nodes are connected and all the agents guard the contiguous border of the clean nodes, thus no recontamination occurs. By construction, this clean border may lie in one single column c_j ($0 < j < n$), or two consecutive columns c_j and c_{j+1} ($0 < j < n-1$). Note that nodes on the last column are clean, but not the edges connecting them.

3) *Induction Step:* By construction, at step $(i+1)$, the *Synchronizer* moves at a node x thus cleaning a row or column edge, and forces the agent located at x to execute one move *EAST* so to clean and guard a new node, and clean the connecting edge. Thus the cleaned area is increased by one more node and one or two edges (the row and eventually the column edge), and no recontamination occurs. All the other agents still guard their nodes and do not move. The node from which the agent has moved during step $(i+1)$ is now guarded by the *Synchronizer*, thus no recontamination occurs and the cleaning remains connected and monotone.

Finally, for the edge-search we only need to prove that the extra m moves for the column cleaning are contiguous and monotone. According to our algorithm, the last column is cleaned as S moves along the edges on the last column, while all the other m agents guard the m nodes of the last column. Thus, no recontamination can occur and the cleaning result is contiguous and monotone. Therefore, the intruders are captured as S is able to synchronize all the moves of the other m searchers. ∎

Theorem 5 *The number of searchers required by algorithm*

SEARCH SYNCH *in a* $m \times n$*-Mesh* ($m \leq n$) *is* $m+1$. *During algorithm* SEARCH SYNC *, for the node (edge) search, the searchers perform* $\frac{m^2+4mn-5m-2}{2}$ *(respectively,* $\frac{m^2+4mn-3m-2}{2}$*) moves, and spend* $mn-1$ *(respectively,* $mn+m-2$*) units of time.*

PROOF The number of moves can be computed as follows. During the *INITIALIZATION*, the searchers globally perform $1+2+...+m-1 = \frac{m(m-1)}{2}$ moves to reach their starting points. During the *CLEAN-UP*, all the m searchers move $(n-1)$ moves *EAST* for a total number of $m(n-1)$ moves. Therefore, the total number of moves consumed by searchers is $\frac{m(m-1)}{2} + m(n-1) = \frac{m^2+2mn-3m}{2}$. On the other hand, during the *CLEAN-UP*, S moves $(n-2)$ moves *EAST*, $\lceil \frac{n-1}{2} \rceil$ times *SOUTH* on a column, and $\lfloor \frac{n-1}{2} \rfloor$ times *NORTH* on a column, where moving on each column requires $(m-1)$ moves. So, the total number of moves performed by S is $(n-2)+(m-1)(n-1) = mn-m-1$. Therefore, the total number of moves for the node-search on a $m \times n$-Mesh with a *Synchronizer* is $\frac{m^2+2mn-3m}{2} + (mn-m-1) = \frac{m^2+4mn-5m-2}{2}$.

For the edge-search S also executes the cleaning of the last column, i.e., $1+m-1 = m$ extra moves. Globally, $\frac{m^2+4mn-5m-2}{2} + m = \frac{m^2+4mn-3m-2}{2}$ moves.

Based on the ideal time assumption, the cleanup time can be calculated as follows. During the *INITIALIZATION*, agents move one by one to their starting points, hence it takes $(m-1)$ time units for them to line up along the first column. During the *CLEAN-UP*, S requires $(m-1)$ time units to move *SOUTH* or *NORTH* along the $(n-1)$ columns in the Mesh. At the same time, all the searchers move *EAST*. Moreover S needs $(n-2)$ time units to move *EAST* to the next adjacent column after searchers have moved to the next column. The last move $EAST$ executed by the last searcher requires an extra time unit. Therefore, the total cleanup time for the node-search is $(m-1)+(m-1)(n-1)+(n-2)+1 = mn-1$ time units. For the edge-search the cleaning of the last column requires extra $m-1$ time units, therefore globally $mn-1+m-1 = mn+m-2$ time units. ∎

5. Node and Edge-Search with Visibility

In this model, we assume that agents have visibility power, i.e., can see the other agents located at their adjacent nodes and may autonomously coordinate their searching operations. Thus, no *"Synchronizer"* is required and agents communicate with each other through a *Whiteboard* associated to each node.

Without needing a *"Synchronizer"*, we will exactly match the lower bound by using m searching agents to capture the intruder in the case of node-search. However, we will need one extra *Helper* agent in the edge-search variation to clean the column edges.

5.1. Node-Search with Visibility

We now illustrate the Searching Strategy.

Initialization Phase: We assume that initially, all the m agents are located at node $v_{0,0}$.

As for the case of agents without visibility, there is a first phase in which the m agents place themselves on the first column and a second phase in which they move *EAST*. However, in this case, the two phases overlap in the sense that some agent might be able to start moving *EAST* while the first column is not completely occupied by agents. Intuitively, each agent will move *EAST* to continue the cleaning as soon as it realizes that it is "safe" to leave the current node. The cleaning steps are illustrated in Figure 2.

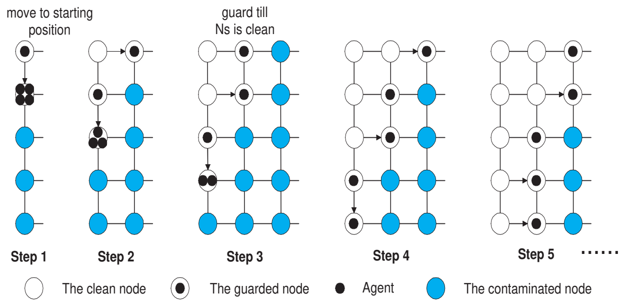

Figure 2. Some steps of Node-Search with Visibility. Black dots represent agents, initially they are together at the home base.

Cleaning Phase: Each searcher s independently performs the following operations.

1) if the *Whiteboard* at the node where s is currently located is empty, s writes "CLEAN" on it. This means that s is the one that will eventually proceed with the cleaning *EAST* from this location, while all the other agents passing by this node will have to proceed *SOUTH*. In fact, s guards the current node until it can see that it is safe to move (i.e., when its neighboring nodes N_N, N_W, and N_S except N_E are all clean (or guarded)), then it moves *EAST* to the next column and it repeats this operation until it reaches the last column.

2) if the *Whiteboard* at the node where s is currently located "CLEAN" message, s moves *SOUTH*. This means that we are in the first column, and some other searcher is going to move *EAST* from this location.

The formal description of the algorithm is provided in Algorithm NODE SEARCH WITH VISIBILITY in Table 3. The input is a contaminated mesh and m agents, the output is the same mesh after the cleaning. WB represents the *Whiteboard*.

We now show that the algorithm is correct and we measure its complexity.

Table 3 Node-Search with Visibility

ALGORITHM NODE SEARCH WITH VISIBILITY

INITIALIZATION
$WB = \emptyset$;
All m agents are at $v_{0,0}$ and *Wake-Up* to be the *Searchers* s;

CLEAN-UP

```
for each s do i = 0; j = 0; //i, j are local variables for each s
    s reads messages from WB at node v_{i,j};
    while (WB == CLEAN) do
        i+ = 1;
        s moves to node v_{i,j} ; // move SOUTH
        s reads messages from WB at node v_{i,j};
    while (WB == ∅) && (j < n − 1) do
            WB = CLEAN;
            while (N_N && N_W && N_S)! = clean do
                    s guards node v_{i,j};
            j+ = 1;
            s moves to node v_{i,j} ; // move EAST
            s reads messages from WB at node v_{i,j};
```

Theorem 6 *Algorithm* NODE SEARCH WITH VISIBILITY *correctly solves the node-search problem in a Mesh.*

PROOF To prove the correctness of the Algorithm NODE SEARCH WITH VISIBILITY , we need to prove that the intruder is captured and that the cleaning result is contiguous and monotone. We prove it by induction on the number of cleaned nodes.

1) *Base Case: the node $v_{0,0}$ is clean.* Initially all the agents are located at the node $v_{0,0}$ and thus this node is clean.

2) *Assumption Step:* Let's assume that at step i $(i > 0)$, the cleaning result is contiguous and monotone. In other words, at step i, all the clean nodes are connected and all the agents guard the contiguous border of the clean nodes, thus no recontamination occurs. By construction, this clean border may line in one single column c_j $(0 < j < n)$ or on several consecutive columns c_j, c_{j+1}, ..., c_k $(0 < j < k < n)$ depending on the state of the cleaning steps. The clean border is contiguous and the cleaning result is monotone.

3) *Induction Step:* By construction, if all agents guard one single column c_j at cleaning step i then at the following cleaning step $(i + 1)$, all searchers move *EAST* to clean and guard the nodes on column c_{j+1}. This is because all searchers can see their neighboring nodes except the N_E that are all clean. Therefore, the cleaned area is increased by one more column and no recontamination occurs. On the other hand, if the agents guard several consecutive columns c_j, c_{j+1}, ..., c_k $(0 < j < k < n)$ at the cleaning step i then at step $(i + 1)$, there will be three possibilities: 1) If a searcher sees that its neighboring nodes except N_E are all clean or guarded, then it moves *EAST* to the next adjacent column. 2) If a searcher s sees a "CLEAN" message on the *Whiteboard*, then it knows this node is has been cleaned by another searcher (that has seen guarded or cleaned nodes around), thus it move *SOUTH* to the next adjacent row. 3) Otherwise, searchers continue to guard the current node and do not move.

The clean border is contiguous after the cleaning step $(i + 1)$. In both case 1) and 2), the node from which the agent has moved during step $(i + 1)$, is totally surrounded by clean or guarded nodes and thus no recontamination occurs.

Thus, the cleaning result is contiguous and monotone. ∎

Theorem 7 *Algorithm* NODE SEARCH WITH VISIBILITY *in a $m \times n$-Mesh $(m \leq n)$ optimally requires m searchers and $(m + n - 2)$ units of time. The number of moves agents have to perform is $\frac{m^2+2mn-3m}{2}$.*

PROOF The fact that m searchers are employed comes directly from the definition of the algorithm, and this number matches the lower bound.

Based on the ideal time assumption, the cleanup time is trivially computed on the slowest searcher as follows. This searcher has to move from $v_{0,0}$ to $v_{m-1,0}$ (i.e., the last row) and then to $v_{m-1,n-1}$ (i.e., the last column) for a total of $(m-1) + (n-1) = m + n - 2$ time units, which is optimal.

The number of moves can be computed as follows. During the *CLEAN-UP*, all the m searchers require $1 + 2 + ... + m - 1 = \frac{m(m-1)}{2}$ moves to move *SOUTH* to place themselves on the first column. Then, all the m searchers move only $(n - 1)$ moves *EAST* for a total number of $m(n - 1)$ moves. So, the total number of moves for the searchers are $\frac{m(m-1)}{2} + m(n-1) = \frac{m^2+2mn-3m}{2}$. ∎

5.2. Edge-Search with Visibility

Initially, all the agents are located at $v_{0,0}$. The idea is similar to the one used for node-search, however a *Helper* agent is required in order to clean the edges. This new agent will be the first to wake-up at $v_{0,0}$. Then, if an agent arrives at first on any other node it writes a 'CLEAN" message and guards the node until it sees all its $NORTH, SOUTH, WEST$ neighbouring nodes and edges clean, then it moves $EAST$, and repeats this up to when it reaches the last column. Otherwise, it moves $SOUTH$. At the same time, the *Helper* H will move following a path similar to the one of the synchronizer for the node-search (i.e., $EAST$, then from r_0 to r_n, then $EAST$, then from r_n to r_0). Whenever it moves to a new node it either stops and waits for an agent (if the node is not guarded) or it moves to the next node.

The formal description of the algorithm is provided in Algorithm EDGE SEARCH WITH VISIBILITY in Table 4. The input is a contaminated mesh and $m+1$ agents, the output is the same mesh after the cleaning. WB represents the *Whiteboard*, D a direction, H the helper.

For the correctness and complexity analysis (see also [13]), we obtain that:

Table 4 Edge-Search with Visibility

```
ALGORITHM EDGE SEARCH WITH VISIBILITY

INITIALIZATION
WB = ∅;D = ∅;
All (m + 1) agents are at node v_{0,0} and Wake-Up to be the Searchers s;

CLEAN-UP
for each s do i = 0;j = 0;//i, j are local variables for each s
    s reads messages of WB at node v_{i,j};
    if Current Node is node v_{0,0} and AgentAssigned is not on WB
            then s becomes a H and writes "Agent Assigned" on WB;
    while (WB == CLEAN) do i+ = 1;
                        s moves to v_{i,j} ; // move SOUTH
                        s reads messages of WB at node v_{i,j};
    while (WB == ∅) && (j < n − 1) do
        WB = CLEAN;
        while (N_N && N_W && N_S && E_N && E_W && E_S) ! = clean do
                        s guards node v_{i,j};
        j+ = 1;
        s moves to v_{i,j} ; // move EAST
        s reads messages from WB at node v_{i,j};
for H do x = 0; y = 1; D = SOUTH;
    while (y < n) do H moves to node v_{x,y};
        if (D == SOUTH) then
                        while H can see v_{x,y} is guarded && (x < (m − 1))
                          do x+ = 1; H moves to v_{x,y};
                        if (x == m − 1) then y+ = 1; D = NORTH;
        else if (D == NORTH && y < n) then
                        while H can see v_{x,y} is clean && x > 0
                          do x− = 1; H moves to v_{x,y};
                        if (x == 0) then y+ = 1; D = SOUTH;
```

Theorem 8 *Algorithm* EDGE SEARCH WITH VISIBILITY *correctly solves the edge-search problem in a Mesh.*

Theorem 9 *Algorithm* EDGE SEARCH WITH VISIBILITY *in a* $m \times n$*-Mesh* ($m \leq n$) *requires* $m + 1$ *searchers,* $(mn - m)$ *units of time and* $\frac{m^2+4mn-5m}{2}$ *moves.*

6. Conclusions and Future Work

Table 5 summarizes our complexity results emphasizing the asymptotic optimality of each strategy. Further notice that in the case of agents with visibility, both the number of agents and the time units are optimal also non asymptotically (exactly m for the number of agents and exactly $m + n - 2$ for the time complexity).

It has been previously observed in the case of the hypercube [5] that the more powerful visibility model allows a drastic decrease in time complexity, while it does not seem to help much in terms of size of the team of agents. This observation was not however supported by any theoretical explanation. In this paper we have observed the same phenomenon in the case of node search only. In fact, in such a case the time complexity drops from $O(mn)$ to $\Theta(m + n)$, which is optimal. On the other hand, we have not been able to design a strategy that can exploit visibility to decrease the time complexity in the case of edge search. Finding a better strategy or proving that it cannot be done is an open problem. In other words, it would be interesting to either prove or disprove that a model with agent visibility is indeed more powerful than the model without agent visibility.

Table 5 Search in a $m \times n$**-Mesh**.

Model		*Synchronizer*	*Visibility*
Node	Agents	$\Theta(m)$	$\Theta(m)$
	Moves	$\Theta(mn)$	$\Theta(mn)$
	Time	$O(mn)$	$\Theta(m + n)$
Edge	Agents	$\Theta(m)$	$\Theta(m)$
	Moves	$\Theta(mn)$	$\Theta(mn)$
	Time	$O(mn)$	$O(mn)$

References

[1] L. Barrière and P. Flocchini and P. Fraignaud and N. Santoro. Capture of an Intruder by Mobile Agents. *Proc. 14-th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 200-209, 2002.

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

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

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

[5] P. Flocchini, M. J. Huang, F. L. Luccio. Contiguous Search in the Hypercube for Capturing an Intruder. *Proc. 19th IEEE International Parallel and Distributed Processing Symposium* (IPDPS 2005), to appear.

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

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

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

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

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

[11] T. Parson. The search number of a connected graph. *Proc. of 9-nth Southeastern Conference on Combinatorics, Graph Theory and Computing*, Utilitas Mathematica, 549-554, 1978.

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

[13] L.X. Song. Intruder Capture by Mobile Agents in Mesh Networks. *Master Thesis*, Carleton University, 2005.