3 Distributed Security Algorithms for Mobile Agents

PAOLA FLOCCHINI

School of Electrical Engineering and Computer Science, University of Ottawa, Canada.

NICOLA SANTORO

School of Computer Science, Carleton University, Canada.

3.1 INTRODUCTION

Mobile agents have been extensively studied for several years by researchers in artificial intelligence and in software engineering. They offer a simple and natural way to describe distributed settings where mobility is inherent and an explicit and direct way to describe the entities of those settings, such as mobile code, software agents, viruses, robots, and Web crawlers. Further, they allow to immediately express notions such as selfish behavior, negotiation, and cooperation arising in the new computing environments. As a programming paradigm, they allow a new philosophy of protocol and software design, bound to have an impact as strong as that caused by object-oriented programming. As a computational paradigm, mobile agent systems are an immediate and natural extension of the traditional message-passing settings studied in distributed computing.

For these reasons, the use of mobile agents is becoming increasingly popular when computing in networked environments, ranging from the Internet to the data grid, both as a theoretical computational paradigm and as a system-supported programming platform.

In networked systems that support autonomous mobile agents, a main concern is how to develop efficient agent-based *system protocols*, that is, to design protocols that will allow a team of identical simple agents to cooperatively perform (possibly complex) system tasks. Examples of basic tasks are *wakeup, traversal, rendezvous,* and *election.* The coordination of the agents

Mobile Agents in Networking and Distributed Computing, First Edition.
Edited by Jiannong Cao and Sajal K. Das.

necessary to perform these tasks is not necessarily simple or easy to achieve. In fact, the computational problems related to these operations are definitely nontrivial, and a great deal of theoretical research is devoted to the study of conditions for the solvability of these problems and to the discovery of efficient algorithmic solutions [1–10].

At an abstract level, these environments can be described as a collection of autonomous mobile *agents* (or *robots*) located in a graph G. The agents have limited computing capabilities and private storage, can move from node to neighboring node, and perform computations at each node according to a predefined set of behavioral rules called *protocol,* the same for all agents. They are *asynchronous,* in the sense that every action they perform (computing, moving, etc.) takes a finite but otherwise unpredictable amount of time. Each node of the network, also called a *host*, may provide a storage area called *whiteboard* for incoming agents to communicate and compute, and its access is held in fair mutual exclusion. The research concern is on determining what tasks can be performed by such entities, under what conditions, and at what cost. In particular, a central question is to determine what minimal hypotheses allow a given problem to be solved.

At a practical level, in these environments, *security* is the most pressing concern and possibly the most difficult to address. Actually, even the most basic security issues, in spite of their practical urgency and the amount of effort, must still be effectively addressed [11–15].

Among the severe security threats faced in distributed mobile computing environments, two are particularly troublesome: *harmful agent* (that is, the presence of malicious mobile processes) and *harmful host* (that is, the presence at a network site of harmful stationary processes).

The former problem is particularly acute in unregulated noncooperative settings such as the Internet (e.g., e-mail-transmitted viruses). The latter exists not only in those settings but also in environments with regulated access and where agents cooperate toward common goals (e.g., sharing of resources or distribution of a computation on the grid). In fact, a local (hardware or software) failure might render a host harmful. In this chapter we consider security problems of both types and concentrate on two security problems, one for each type: *locating a black hole* and *capturing an intruder*. For each we discuss the computational issues and the algorithmic techniques and solutions.

We first focus (in Section 3.2) on the issue of *host attacks*, that is, the presence in a site of processes that harm incoming agents. A first step in solving such a problem should be to identify, if possible, the harmful host, that is, to determine and report its location; following this phase, a "rescue" activity would conceivably be initiated to deal with the destructive process resident there. The task to identify the harmful host is clearly dangerous for the searching agents and, depending on the nature of the harm, might be impossible to perform. We consider a highly harmful process that disposes of visiting agents upon their arrival, leaving no observable trace of such a destruction. Due to its nature, the site where such a process is located is called a *black hole*.

The task is to unambiguously determine and report the location of the black hole. The research concern is to determine under what conditions and at what cost mobile agents can successfully accomplish this task. The searching agents start from the same safe site and follow the same set of rules; the task is successfully completed if, within a finite time, at least one agent survives and knows the location of the black hole.

We then consider (in Section 3.3) the problem of *agent attacks*, that is, the presence of a harmful mobile agent in the system. In particular, we consider the presence of a *mobile virus* that infects any visited network site. A crucial task is clearly to decontaminate the infected network; this task is to be carried out by a team of antiviral system agents (the *cleaners*), able to decontaminate visited sites, avoiding any recontamination of decontaminated areas. This problem is equivalent to the one of capturing an intruder moving in the network.

Although the main focus of this chapter is on security, the topics and the techniques have a much wider theoretical scope and range. The problems themselves are related to long-investigated and well-established problems in automata theory, computational complexity, and graph theory. In particular, the *black-hole search* problem is related to the classical problems of *graph exploration* and *map construction* [1, 7, 9, 16–22]. With whiteboards, in the case of dispersed agents (i.e., when each starts from a different node), these problems are in turn computationally related (and sometimes equivalent) to the problems of *rendezvous* and *election* [2, 5, 6, 23–25]. The *network decontamination* problem is instead related to the classical problem known as *graph search* [e.g., 26–30], which is in turn closely related to standard graph parameters and concepts, including tree width, cut width, path width, and, last but not least, graph minors [e.g., 31–34].

The chapter is organized as follows. In the next section we will discuss the black-hole search problem, while the network decontamination and intruder capture problems will be the subject of Section 3.3.

3.2 BLACK-HOLE SEARCH

3.2.1 The Problem and Its Setting

The problem posed by the presence of a harmful host has been intensively studied from a programming point of view [35–37]. Obviously, the first step in any solution to such a problem must be to *identify*, if possible, the harmful host, that is, to determine and report its location; following this phase, a "rescue" activity would conceivably be initiated to deal with the destructive process resident there. Depending on the nature of the danger, the task to identify the harmful host might be difficult, if not impossible, to perform.

Consider the presence in the network of a black hole (BH): a host where resides a stationary process that *disposes* of visiting agents upon their arrival, leaving *no observable* trace of such a destruction. Note that this type of highly harmful host is not rare; for example, the undetectable crash failure of a site in an asynchronous network turns such a site into a black hole. The task is to unambiguously determine and report the location of the black hole by a team of mobile agents. More precisely, the black-hole search (shortly BHS) problem is solved if at least one agent survives and all surviving agents know the location of the black hole.

The research concern is to determine under what conditions and at what cost mobile agents can successfully accomplish this task. The main complexity measures for this problem are the *size* of the solution (i.e., the number of agents employed) and the *cost* (i.e., the number of moves performed by the agents executing a size-optimal solution protocol). Sometimes bounded *time* complexity is also considered.

The searching agents usually start from the same safe site (the *homebase*). In general, no assumptions are made on the time for an agent to move on a link, except that it is finite; that is, the system is asynchronous. Moreover, it is usually assumed that each node of the network provides a storage area called a whiteboard for incoming agents to communicate and compute, and its access is held in fair mutual exclusion.

One can easily see that the black-hole search problem can also be formulated as an *exploration* problem; in fact, the black hole can be located only after all the nodes of the network but one have been visited and are found to be safe. Clearly, in this exploration process some agents may disappear in the black hole). In other words, the black-hole search problem is the problem of exploring an unsafe graph. Before proceeding we will first (briefly) discuss the problem of *safe exploration*, that is, of exploring a graph without any black hole.

3.2.2 Background Problem: Safe Exploration

The problem of exploring and mapping an unknown but *safe* environment has been extensively studied due to its various applications in different areas (navigating a robot through a terrain containing obstacles, finding a path through a maze, or searching a network).

Most of the previous work on exploration of unknown graphs has been limited to single-agent exploration. Studies on exploration of *labeled* graphs typically emphasize minimizing the number of moves or the amount of memory used by the agent [1, 7, 17, 21, 22]. Exploration of *anonymous* graphs is possible only if the agents are allowed to mark the nodes in some way, except when the graph has no cycles (i.e., the graph is a tree [9, 18]). For exploring arbitrary anonymous graphs, various methods of marking nodes have been used by different authors. Pebbles that can be dropped on nodes have been proposed first in [16], where it is shown that any strongly connected directed graph can be explored using just one pebble (if the size of the graph is known), and using $O(\log \log n)$ pebbles otherwise. Distinct markers have been used, for example, in [38] to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [39] was to employ two cooperating agents, one of which would stand on a node, while the other explores new edges. Whiteboards have been used by Fraigniaud and Ilcinkas [19] for exploring directed graphs and by Fraigniaud et al. [18] for exploring trees. In [9, 19, 20] the authors focus on minimizing the amount of memory used by the agents for exploration (they however do not require the agents to construct a map of the graph).

There have been few results on exploration by more than one agent. A two-agent exploration algorithm for directed graphs was given in [39], whereas Fraigniaud et al. [18] showed how k agents can explore a tree. In both these cases, the agents start from the same node and they have distinct identities. In [6] a team of dispersed agents explores a graph and constructs a map. The graph is anonymous but the links are labeled with sense of direction; moreover the protocol works if the size n of the network or the number of agents k is coprime and it achieves a move complexity of $O(km)$ (where m is the number of edges). Another algorithm with the same complexity has been described in [23], where the requirement of sense of direction is dropped. In this case the agents need to know either n or k, which must be coprime. The solution has been made "effective" in [24], where effective means that it will always terminate, regardless of the relationship between n and k reporting a solution whenever the solution can be computed, and reporting a failure message when the solution cannot be computed.

The map construction problem is actually equivalent to some others basic problems, such as *agent election, labeling,* and *rendezvous*. Among them rendezvous is probably the most investigated; for a recent account see [2, 25].

3.2.3 Basic Properties and Tools for Black-Hole Search

We return now to the black-hole search problem and discuss first some basic properties and techniques.

3.2.3.1 Cautious Walk

We now describe a basic tool [40] that is heavily employed when searching for a black hole. In order to minimize the number of agents that can be lost in the black hole, the agents have to move *cautiously*. More precisely, we define as *cautious walk* a particular way of moving on the network that prevents two different agents to traverse the same link when this link potentially leads to the black hole.

At any time during the search for the black hole, the ports (corresponding to the incident links) of a node can be classified as *unexplored* (no agent has been sent/received via this port), *explored* (an agent has been received via this port), or *dangerous* (an agent has been sent through this port but no agent has been received from it). Clearly, an explored port does not lead to a black hole; on the other hand, both unexplored and dangerous ports might lead to it.

The main idea of cautious walk is to avoid sending an agent over a dangerous link while still achieving progress. This is accomplished using the following two rules:

1. No agent enters a dangerous link.
2. Whenever an agent a leaves a node u through an unexplored port p (transforming it into dangerous), upon its arrival to node v and before proceeding somewhere else, a returns to u (transforming that port into explored).

Similarly to the classification adopted for the ports, we classify nodes as follows: At the beginning, all nodes except the homebase are unexplored; the first time a node is visited by an agent, it becomes explored. Note that, by definition, the black hole never becomes explored. Explored nodes and edges are considered safe.

3.2.3.2 Basic Limitations

When considering the black-hole search problem, some constraints follow from the asynchrony of the agents (arising from the asynchrony of the system, that is, the impossibility to distinguish the BH from a slow node). For example [40]:

- If G has a cut vertex different from the homebase, then it is impossible for asynchronous agents to determine the location of the BH.
- It is impossible for asynchronous agents to determine the location of the BH if the size of G is not known.
- For asynchronous agents it is impossible to verify if there is a BH.

As a consequence, the network must be 2-connected; furthermore, the existence of the black hole and the size of G must be common knowledge to the agents.

As for the number of searching agents needed, since one agent may immediately wander into the black hole, we trivially have:

- At least two agents are needed to locate the BH.

How realistic is this bound? How many agents suffice? The answers vary depending on the a priori knowledge the agents have about the network and on the consistency of the local labelings.

3.2.4 Impact of Knowledge

3.2.4.1 Black-Hole Search without a Map

Consider first the situation of *topological ignorance*, that is, when the agents have no a priori knowledge of the topological structure of G (e.g., do not have a map of the network). Then any generic solution needs at least $\Delta + 1$ agents, where Δ is the maximal degree of G, even if the agents know Δ and the number n of nodes of G.

The goal of a black-hole search algorithm $\mathcal{P}$ is to identify the location of the BH; that is, within finite time, at least one agent must terminate with a map of the entire graph where the homebase, the current position of the agent, and the location of the black hole are indicated. Note that termination with an exact map in finite time is actually impossible. In fact, since an agent is destroyed upon arriving to the BH, no surviving agent can discover the port numbers of the black hole. Hence, the map will have to miss such an information. More importantly, the agents are asynchronous and do not know the actual degree d (BH) of the black hole (just that it is at most Δ). Hence, if an agent has a local map that contains $N - 1$ vertices and at most Δ unexplored edges, it cannot distinguish between the case when all unexplored ports lead to the black hole and the case when some of them are connected to each other; this ambiguity cannot be resolved in finite time or without the agents being destroyed. In other words, if we require termination within finite time, an agent might incorrectly label some links as incident to the BH; however, the agent needs to be wrong only on at most $\Delta - d(\text{BH})$ links. Hence, from a solution algorithm $\mathcal{P}$ we require termination by the surviving agents within finite time and creation of a map with just that level of accuracy.

Interestingly, in any *minimal* generic solution (i.e., using the minimum number of agents), the agents must perform $\Omega(n^2)$ moves in the worst case [41]. Both these bounds are *tight*. In fact, there is a protocol that correctly locates the black hole in $O(n^2)$ moves using $\Delta + 1$ agents that know Δ and n [41].

The algorithm essentially performs a collective "cautious" exploration of the graph until all nodes but one are considered to be safe. More precisely, the agents cooperatively visit the graph by "expanding" all nodes until the black hole is localized, where the expansion of a node consists of visiting all its neighbors. During this process, the homebase is used as the cooperation center; the agents must pass by it after finishing the expansion of a node and before starting a new expansion. Since the graph is simple, two agents exploring the links incident to a node are sufficient to eventually make that node "expanded." Thus, in the algorithm, at most two agents cooperatively expand a node; when an agent discovers that the node is expanded, it goes back to the homebase before starting to look for a new node to expand. The whiteboard on the homebase is used to store information about the nodes that already have been explored and the ones that are under exploration. If the black hole is a node with maximum degree, there is nothing to prevent Δ agents disappearing in it.

3.2.4.2 *Black-Hole Search with Sense of Direction*

Consider next the case of topological ignorance in systems where there is *sense of direction* (SD); informally, sense of direction is a labeling of the ports that allows the nodes to determine whether two paths starting from a node lead to the same node using only the labels of the ports along these paths (for a survey on sense of direction see [42]). In this case, two agents suffice to locate the black hole, regardless of the (unknown) topological structure of G. The proof of [41] is constructive, and the algorithm has a $O(n^2)$ cost. This cost is optimal; in fact, it is shown that there are types of sense of direction that, if present, impose an $\Omega(n^2)$ worst-case cost on any generic two-agent algorithm for locating a black hole using SD. As for the topological ignorance case, the agents perform an exploration. The algorithm is similar to the one with topological ignorance (in fact it leads to the same cost); sense of direction is however very useful to decrease the number of casualties. The exploring agents can be only two: A node that is being explored by an agent is considered "dangerous" and, by the properties of sense of direction, the other agent will be able to avoid it in its exploration, thus ensuring that one of the two will eventually succeed.

3.2.4.3 Black-Hole Search with a Map

Consider the case of *complete topological knowledge* of the network; that is, the agents have a complete knowledge of the edge-labeled graph G, the correspondence between port labels and the link labels of G, *and* the location of the source node (from where the agents start the search). This information is stronger than the more common *topological awareness* (i.e., knowledge of the class of the network, but not of its size or of the source location, e.g., being in a mesh, starting from an unknown position).

Also in this case two agents suffice [41]; furthermore, the cost of a minimal protocol can be reduced in this case to $O(n \log n)$, and this cost is worst-case optimal. The technique here is quite different and it is based on a partitioning of the graph in two portions which are given to the two agents to perform the exploration. One will succeed in finishing its portion and will carefully move to help the other agent finishing its own.

Informally, the protocol works as follows. Let G_{ex} be the explored part of the network (i.e., the set of safe nodes); initially it consists only of the homebase h. Agents a and b partition the unexplored area into disjoint subgraphs G_a (the working set for a) and G_b (the working set for b) such that for each connected component of G_a and G_b there is a link connecting it to G_{ex} (this partitioning can always be done). Let T_a and T_b be trees spanning G_a and G_b, respectively, such that $T_a \cap G_b = T_b \cap G_a = \emptyset$. (The graphs G_a and G_b are not necessarily connected—the trees T_a and T_b are obtained from the spanning forests of G_a and G_b by adding edges from G_{ex} as necessary but avoiding the vertices of the opposite working set.)

Each agent then traverses its working set using cautious walk on the corresponding spanning tree. In this process, it transforms unexplored nodes into safe ones.

Let a be the first agent to terminate the exploration of its working set; when this happens, a goes to find b. It does so by first going to the node w where the working sets were last computed using an optimal path and avoiding G_b, then following the trace of b, and finally reaching the last safe node w' reached by b.

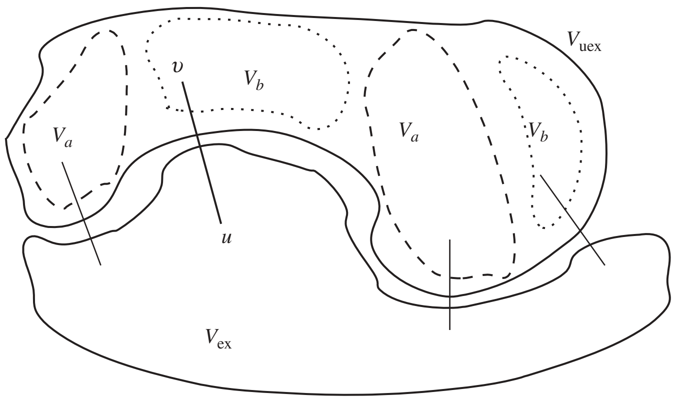

FIGURE 3.1 Splitting the unexplored subgraph G_{uex} into G_a and G_b.

Agent a then computes the new subgraph G_{uex} containing all nonsafe nodes (Figure 3.1). If G_{uex} contains a single node, that node is the black hole. Otherwise a computes the new working sets for itself and b; it leaves a note for b at the current node w' indicating the new working set G_b for b and goes to explore its new assigned area avoiding the (new) working set of b. When (if) b returns to w', it finds the note and starts exploring its new working set. Note that, at any time, an agent is either exploring its working set or looking for the other agent to update the workload or is destroyed by the black hole.

3.2.4.4 Topology-Sensitive Universal Protocols

Interestingly, it is possible to considerably improve the bound on the number of moves without increasing the team size. In fact, there is a recent *universal* protocol, *Explore and Bypass,* that allows a team of *two* agents with a map of the network to locate a black hole with cost $O(n + d \log d)$, where d denotes the diameter of the network [43]. This means that, without losing its universality and without violating the worst-case $\Omega(n \log n)$ lower bound, this algorithm allows two agents to locate a black hole with $\Theta(n)$ cost in a very large class of (possibly unstructured) networks: those where $d = O(n/\log n)$.

The algorithm is quite involved. The main idea is to have the agents explore the network using cooperative depth-first search of a spanning tree T. When further progress using only links of T is blocked, the blocking node is appropriately bypassed and the process is repeated. For efficiency reasons, the bypass is performed in different ways depending on the structure of the unexplored set U and on the size of its connected components. The overall exploration is done in such a way that:

1. The cost of the cooperative depth-first search is linear in the number of explored vertices.
2. Bypassing a node incurs an additional overhead of $O(d)$ which can be charged to the newly explored vertices if there are enough of them.

3. If there are not enough unexplored vertices remaining for bypassing to be viable, the remaining unexplored graph is so small [$O(d)$] that applying the general $O(n \log n)$ algorithm would incur an $O(d \log d)$ additional cost [which is essentially optimal, due to the lower bound of $\Theta(n \log n)$ for rings].

Importantly, there are many networks with $O(n/\log n)$ diameter in which the previous protocols [41, 44] fail to achieve the $O(n)$ bound. A simple example of such a network is the *wheel*, a ring with a central node connected to all ring nodes, where the central node is very slow: Those protocols will require $O(n \log n)$ moves.

3.2.4.5 Variations with a Map

A very simple algorithm that works on any topology (a priori known by the agents) is shown in [45].

Let $\mathcal{C}$ be a set of simple cycles such that each vertex of G is covered by a cycle from $\mathcal{C}$ (Figure 3.2). Such a set of cycles with some connectivity constraint is called *open-vertex cover* by cycles. The algorithm is based on the precomputation of such an open-vertex cover by cycles of a graph. The idea is to explore the graph G by exploring the cycles $\mathcal{C}$.

The algorithm uses the optimal number of agents (two). If an agent is blocked on an edge e (because either the transmission delay on e is very high or it leads to the BH), the other agent will be able to bypass it, using the cycle containing e, and continue the exploration. The number of moves depends on the choice of the cover and it is optimal for several classes of networks. These classes include all Abelian Cayley graphs of degree 3 and more (e.g., hypercubes, multidimensional

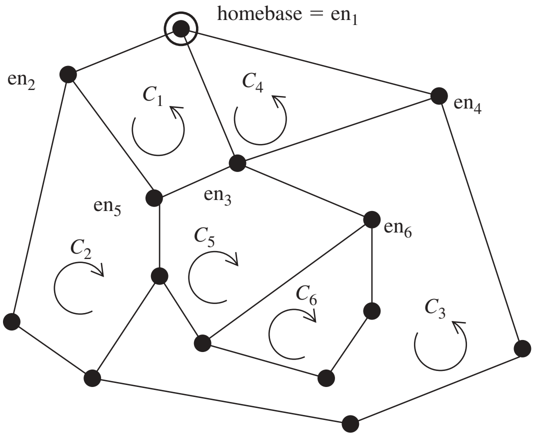

FIGURE 3.2 Example of an open vertex cover $\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5, C_6\}$, for graph G. The cycle directions are shown, as well as the entry nodes en_i for each cycle C_i.

tori, etc,), as well as many non-Abelian cube graphs (e.g., cube-connected-cycles, butterfly, wrapped-butterfly networks). For some of these networks, this is the only algorithm achieving such a bound.

3.2.5 Special Topologies

A natural question to ask is whether the bounds for arbitrary networks with full topological knowledge can be improved for networks with special topologies by topology-dependent proptocols.

3.2.5.1 Rings

The problem has been investigated and its solutions characterized for *ring* networks [40]. For these networks, at least two agents are needed, and a $\Omega(n \log n)$ lower bound holds on the number of moves regardless of the number of agents [40].

An agent and move optimal solution exists based on a partitioning of the ring and a nonoverlapping exploration by the agents. The solution is similar to (and simpler than) the one for the known arbitrary topology case. Initially the agents use the whiteboard to differentiate their tasks: each taking charge of (cautiously) exploring roughly half of the ring. One of the two agents will necessarily succeed (say agent A); by that time, the other (agent B) is either still exploring or trapped into the black hole. The successful agent A follows the safe trace of B; at the last safe node found following the trace, A writes on the whiteboard a message for B, indicating that it will now take charge of half of the area still to be explored. In this way, if B is not in the black hole and returns to the node (as imposed by cautious walk), it will find the message and will act accordingly. Notice that the size of the ring must be known for the algorithm to work; furthermore, without knowledge of the size, the problem is unsolvable. The key point of the algorithm's correctness is that the agents are always exploring disjoint areas; hence one of them will always complete its exploration. Since the unexplored area to be explored is halved in each stage, the total number of stages is O (log n); the amount of moves in each stage is linearly proportional to the size of the explored area; therefore, the total number of moves is $O(n \log n)$. The time complexity of this solution is also $O(n \log n)$.

Interestingly, increasing the number of agents, the number of moves cannot decrease, but the time to finish the exploration does [40]. For example, suppose n agents $x_1, x_2, \ldots, x_n$ are available. By accessing the whiteboard they can assign to themselves different tasks: for example, agent x_i could take care of exploring the node at distance i (clockwise: if there is no orientation, a similar trick would work). To explore node u at distance i, agent x_i moves to visit the nodes that precede u clockwise and the one that precedes u counterclockwise. Only one agent will be successful because all the others will terminate in the black hole either when moving clockwise or when moving counterclockwise. Notice that, in their exploration, the agents do not need to move with cautious walk. Clearly the agents can perform their tasks concurrently and the time complexity is $\Omega(n)$. Indeed, there exists an optimal trade-off between time complexity and number of agents.

Notice that the lower bound for rings implies an $\Omega(n \log n)$ lower bound on the worst-case cost complexity of any *universal* protocol.

The ring has been investigated also to perform another task: rendezvous of k anonymous agents dispersed in the ring in spite of the presence of a black hole. The problem is studied in [46] and a complete characterization of the conditions under which the problem can be solved is established. The characterization depends on whether k or n is unknown (at least one must be known for any nontrivial rendezvous). Interestingly, it is shown that, if k is unknown, the rendezvous algorithm also solves the black-hole location problem, and it does so with a bounded-time complexity of $\Theta(n)$; this is a significant improvement over the $O(n \log n)$ bounded-time complexity of [40].

3.2.5.2 Interconnection Networks

The $\Omega(n \log n)$ lower bound for rings does not necessarily generalize to other topologies. Sometimes the network has special properties that can be exploited to obtain a lower cost network-specific protocol. For example, two agents can locate a black hole with only $O(n)$ moves in a variety of highly structured interconnection networks such as *hypercubes*, square *tori* and *meshes*, *wrapped butterflies*, and *star graphs* [44].

The protocol achieving such a bound is based on the novel notion of *traversal pairs* of a network which describes how the graph will be explored by each agent and will be used by an agent to avoid "dangerous" parts of the network. The algorithm proceeds in logical rounds. In each round the agents follow a usual cooperative approach of dynamically dividing the work between them: The unexplored area is partitioned into two parts of (almost) equal size. Each agent explores one part without entering the other one; exploration and avoidance are directed by the traversal pair. Since the parts are disjoint, one of them does not contain the black hole and the corresponding agent will complete its exploration. When this happens, the agent (reaches the last safe node visited by the other agent and there) partitions whatever is still left to be explored leaving a note for the other agent (should it be still alive). This process is repeated until the unexplored area consists of a single node: the black hole. In addition to the protocol and its analysis, in [44] there is also the algorithm for constructing a traversal pair of a biconnected graph.

3.2.6 Using Tokens

As we have seen, the problem of asynchronous agents exploring a dangerous graph has been investigated assuming the availability of a whiteboard at each node: Upon gaining access, the agent can write messages on the whiteboard and can read all previously written messages, and this mechanism has been used by the agents to communicate and mark nodes or/and edges. The whiteboard is indeed a powerful mechanism of interagent communication and coordination.

Recently the problem of locating a black hole has been investigated also in a different, weaker model where there are no whiteboards at the nodes. Each agent has instead a bounded number of tokens that can be carried, placed on a node or on a port, or removed from it; all tokens are identical (i.e., indistinguishable) and no other form of marking or communication is available [47–49]. Some natural questions immediately arise: Is the BHS problem still solvable with this weaker mechanism and if so under what conditions and at what cost? Notice that the use of tokens introduces another complexity measure: the number of tokens. Indeed, if the number of tokens is unbounded, it is possible to simulate a whiteboard environment; hence the question immediately arises of how many tokens are really needed.

Surprisingly, the black-hole search problem in an unknown graph can be solved using only this weaker tool for marking nodes and communicating information. In fact, it has been shown [47] that $\Delta + 1$ agents with a single token each can successfully solve the black-hole search problem; recall that this team size is *optimal* when the network is unknown. The number of moves performed by the agents when executing the protocol is actually polynomial. Not surprisingly, the protocol is quite complex. The absence of a whiteboard, in fact, poses serious limitations to the agents, which have available only a few movable bits to communicate with each other.

Special topologies have been studied as well, and in particular, the case of the *ring* has been investigated in detail in [48]. There it has been shown that the two-agent $\Theta(n \log n)$ move strategies for black-hole search in rings with whiteboards can be successfully employed also without whiteboards by carefully using a bounded number of tokens. Observe that these optimal token-based solutions use only $O(1)$ tokens in total, whereas the protocols using whiteboards assumed at least $O(\log n)$ dedicated bits of storage at each node. Further observe that any protocol that uses only a constant number of tokens implies the existence of a protocol (with the same size and cost) that uses only whiteboards of constant size; the converse is not true.

In the case of arbitrary networks, it has been recently shown that, with a map of the graph, a $\Theta(n \log n)$-moves solution exists for 2 agents with a single token each, which can be placed only on nodes (like in classical exploration algorithms with pebbles) [49]. These results indicate that, although tokens appear as a weaker means of communication and coordination, their use does not negatively affect solvability and it does not even lead to a degradation of performance.

3.2.7 Synchronous Networks

The black-hole search problem has been studied also in synchronous settings where the time for an agent to traverse a link is assumed to be unitary.

When the system is synchronous, the goals and strategies are quite different from the ones reviewed in the previous sections. In fact, when designing an algorithm for the asynchronous case, a major problem is that an agent cannot wait at a node for another agent to come back; as a consequence, agents must always move and have to do it carefully. When the system is synchronous, on the other hand, the strategies are mostly based on waiting the right amount of time before performing a move. The algorithm becomes the determination of the shortest traversal schedule for the agents, where a traversal schedule is a sequence of actions (move to a neighboring node or stay at the current node). Furthermore, for the black-hole search to be solvable, it is no longer necessary that the network is two-node connected; thus, black-hole search can be performed by synchronous agents also in trees.

In synchronous networks tight bounds have been established for some classes of trees [50]. In the case of general networks the decision problem corresponding to the one of finding the optimal strategy is shown to be nondeterministic polynomial time (NP) hard [51, 52] and approximation algorithms are given in [50] and subsequently improved in [52, 53]. The case of multiple black holes has been very recently investigated in [54], where a lower bound on the cost and close upper bounds are given.

3.2.8 Rendezvous in Spite of Black Hole

In networks where a black hole is present, the primary task is clearly that of determining the location of the black hole. In addition to the BHS problem, the research has also focused on the computability of other tasks in the presence of a black hole, that is, on the design of *black-hole resilient* solutions to problems. In particular, the focus has been on the rendezvous problem: k anonymous agents are dispersed in the network and must gather at the same node; the location of the rendezvous is not fixed a priori. As mentioned in Section 3.2.2, rendezvous (with whiteboards) is equivalent to the exploration and leader election problems, and it has been extensively studied in networks without black holes. In the presence of a black hole, the problem changes drastically. In fact, it is impossible to guarantee that all agents will gather since some agents might enter the black hole. Thus the main research concern is to determine how many agents can be guaranteed to rendezvous and under what conditions.

A solution strategy would be to first determine the location of the black hole and then perform a rendezvous in the safe part of the network. This strategy requires solving the BHS problem in a setting quite different from that examined by the investigators so far; in fact, the agents would not start from the same safe node, the homebase, but are instead *scattered* in the network. To date, the only solutions to the BHS problem when the agents are scattered are for ring networks using whiteboards [40] or tokens [48].

Interestingly, the overall problem can be solved without necessarily having to locate the black hole. The overall problem has been first investigated when the k anonymous agents are dispersed in a ring, and a complete characterization of the conditions under which the problem can be solved is established in [46]. The characterization depends on whether k or n is unknown (at least one must be known for any nontrivial rendezvous). Interestingly, it is shown that, if k is unknown, the rendezvous algorithm also solves the black-hole location problem, and it does so with a bounded-time complexity of $\Theta(n)$; this is a significant improvement over the $O(n \log n)$ bounded-time complexity of [40].

The problem has recently been investigated in arbitrary networks in the presence not only of one or more black holes but also of *black links,* that is, edges that destroy traversing agents without leaving any trace [55]. A complete characterization of the conditions necessary for solvability has been provided, and a protocol has been designed that allows rendezvous if these conditions hold and otherwise determines impossibility [40].

3.3 INTRUDER CAPTURE AND NETWORK DECONTAMINATION

3.3.1 The Problem

A particularly important security concern is to protect a network from unwanted and possibly dangerous intrusions. At an abstract level, an intruder is an alien process that moves on the network to sites unoccupied by the system's agents, possibly "contaminating" the nodes it passes by. The concern for the severe damage intruders can cause has motivated a large amount of research, especially on detection [56–58]. From an algorithmic point of view, the concern has been almost exclusively on the intruder capture problem, that is, the problem of enabling a team of system agents to stop the dangerous activities of the intruder by coming in direct contact with it. The goal is to design a protocol, to be executed by the team of agents, enabling them to capture the intruder. To make the protocol as flexible as possible, the intruder is assumed to be very powerful: arbitrarily fast and possibly aware of the positions of all the agents; on the contrary, the agents could be arbitrarily slow and unable to sense the intruder presence except when encountering it (i.e., on a node or on a link).

In this setting, the intruder capture problem is equivalent (both from a computational and a complexity point of view) to the problem of network decontamination. In this problem, the nodes of the network are initially *contaminated* and the goal is to *clean* (or decontaminate) the infected network. The task is to be carried out by a team of antiviral 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 *recontaminated* (e.g., if the virus returns to that site in the absence of a cleaner); the specifications of the conditions under which this can happen is called a recontamination rule. The most common recontamination rule is that, if a node without an agent on it has a contaminated neighbor, it will become (re-)contaminated. 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, possibly avoiding any recontamination.

3.3.2 Background Problem: Graph Search

A variation of the decontamination problem described above has been extensively studied in the literature under the name graph search [26–30].

The graph search problem has been first discussed by Breisch [59] and Parson [30, 60]. In the graph-searching problem, we are given a "contaminated" network, that is, whose links are all contaminated. Via a sequence of operations using "searchers," we would like to obtain a state of the network in which all links are simultaneously clear. A *search step* is one of the following operations: (1) place a searcher on a node, (2) remove a searcher from a node, and (3) move a searcher along a link. There are two ways in which a contaminated link can become clear. In both cases, a searcher traverses the link from one extremity u to the other extremity v. The two cases depend on the way the link is preserved from recontamination: Either another searcher remains in u or all other links incident to u are clear. The goal is to use as few searchers as possible to decontaminate the network. A *search strategy* is a sequence of search steps that results in all links being simultaneously clear. The *search number* $s(G)$ of a network G is the smallest number of searchers for which a search strategy exists. A search strategy using $s(G)$ searchers in G is called minimal.

The decision problem corresponding to the computation of the search number of a graph is NP hard [29] and NP completeness is shown in [28, 61]. In particular, Megiddo et al. [29] gave a $O(n)$ time algorithm to determine the search number of n-node trees and a $O(n \log n)$ time algorithm to determine a minimal search strategy in n-node trees. Ellis, Sudborough, and Turner [26] linked $s(G)$ with the vertex separation $\mathrm{vs}(G)$ of G (known to be equal to the pathwidth of G [62]). Given an n-node network $G=(V, E)$, $\mathrm{vs}(G)$ is defined as the minimum taken over all (one-to-one) linear layouts $L : V \rightarrow \{1, \ldots, n\}$, of $\mathrm{vs}_L(G)$, the latter being defined as the maximum, for $i = 1, \ldots, n$, of the number of vertices $x \in V$ such that $L(x) \leq i$ and there exists a neighbor y of x such that $L(y) > i$. Ellis et al. showed that $\mathrm{vs}(G) \leq s(G) \leq \mathrm{vs}(G) + 2$ and $s(G) = \mathrm{vs}(G')$, where G' is the 2-augmentation of G, that is, the network obtained from G by replacing every link $\{x, y\}$ by a path $\{x, a, b, y\}$ of length 3 between x and y. They also showed that the vertex separation of trees can be computed in linear time, and they gave an $O(n \log n)$ time algorithm for computing the corresponding layout. It yields another $O(n)$ time algorithm returning the search number of trees and an $O(n \log n)$ *time* algorithm returning a minimal search strategy in trees.

Graph searching has many other applications (see [63]), including pursuit–evasion problems in a labyrinth [60], decontamination problems in a system of tunnels, and mobile computing problems in which agents are looking for a hostile intruder [64]. Moreover, the graph-searching problem also arises in very large scale integration (VLSI) design through its equivalence with the gate matrix layout problem [62]. It is hence not surprising that it gave rise to numerous papers. Another reason for this success is that the problem and its several variants (nodesearch, mixedsearch, 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 [31]. For instance, Makedon and Sudborough [33] showed that $s(G)$ is equal to the cut width of G for all networks of maximum degree 3. Similarly, Kiroussis and Papadimitriou showed that the node search number of a network is equal to its interval width [32] and to its vertex separator plus 1 [27]. Seymour and Thomas [34] showed that the t-search number is equal to the tree width plus 1. Takahashi, Ueno, and Kajitani [65] showed that the mixed-search number is equal to the proper path width. Bienstock and Seymour [61] simplified the proof of Lapaugh's result [28] stating that there is a minimal search strategy that does not recontaminate any link (see also [66]). Thilikos [67] used graph minors to derive a linear-time algorithm that checks whether a network has a search number at most 2. For other results on graph searching, the reader is referred to the literature [68–72]. Contributions to related search problems can be found elsewhere [64, 73–78].

In graph searching, there has been a particular interest in monotone strategies [28, 61, 79]. A strategy is monotone if after a node (link) is decontaminated it will not be contaminated again. An important result from Lapaugh has shown that monotonicity does not really change the difficulty of the graph search problem; in fact, it has been shown in [28] that for any graph there exists a monotone search strategy that uses the minimum number of agents.

Let us stress that in the classical graph search problem the agents can be arbitrarily moved from a node "jumping" to any other node in the graph.

The main difference in the setting described in this chapter is that the agents, which are pieces of software, *cannot be removed from the network;* they can only move from a node to a *neighboring* one (*contiguous search*). This additional constraint was introduced and first studied in [4] resulting in a *connected-node search* where (i) the removal of agents is not allowed and (ii) at any time of the search strategy the set of clean nodes forms a connected subnetwork. With the connected assumption the nature of the problem changes considerably and the classical results on node and edge search do not generally apply.

In the case of connected graph search usually more agents are required to decontaminate a network G. It has been shown that for any graph G with n nodes the ratio between the connected search number $\mathrm{csn}(G)$ and the regular search number $\mathrm{sn}(G)$ is always bounded. More precisely, it is known that $\mathrm{csn}(G)/\mathrm{sn}(G) \le \log n + 1$ [80], and, for a tree T, $\mathrm{csn}(T)/\mathrm{sn}(T) \le 2$ [81]. Monotonicity also plays an important role in connected graph searches. It has been shown that, as in the more general graph search problem, a solution allowing recontamination of nodes cannot reduce the optimal number of agents required to decontaminate trees [4]. On the other hand, unlike the classical graph search, there exist graphs for which any given monotone-connected graph search strategy requires more searchers than the optimal non-monotone-connected search strategy [82]. The decision problem corresponding to the computation of the connected search number of a graph is NP hard, but it is not known whether there exists a yes certificate that is checkable in polynomial time.

As we will survey below, the problem has been studied mostly in specific topologies. Also the arbitrary topology has been considered; in this case, some heuristics have been proposed [83] and a move-exponential optimal solution has been given in [84].

In this chapter we use decontamination to refer to the connected monotone node search as defined in [4].

3.3.3 Models for Decontamination

Initially, all agents are located at the same node, the homebase, and all the other nodes are contaminated; a decontamination strategy consists of a sequence of movements of the agents along the edges of the network. The agents can communicate when they reside on the same node.

Starting from the classical model employed in [4] (called the local model), additional assumptions have sometimes been added to study the impact that more powerful agents' or system's capabilities have on the solutions of our problem:

1. In the *local model* an agent located at a node can "see" only local information, such as the state of the node, the labels of the incident links, and the other agents present at the node.
2. *Visibility* is the capability of the agent to see the state of its neighbors; that is, an agent can see whether a neighboring node is guarded and 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 neighboring nodes before making a decision.
3. *Cloning* is the capability, for an agent, to clone copies of itself.
4. *Synchronicity* implies that local computations are instantaneous, and it takes one unit of time (one step) for an agent to move from a node to a neighboring one.

The efficiency of a strategy is usually measured in terms of number of agents, amount of decontamination operations, number of moves performed by the agents, and ideal time.

We say that a cleaning strategy is *monotone* if, once a node is clean, it will never be contaminated again. 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 performed is the smallest possible, one per network site. All the results reported here are for monotone strategies.

3.3.4 Results in Specific Topologies

3.3.4.1 Trees

The tree has been the first topology to be investigated in the local model [4]. Notice that, for a give tree T, the minimum number of agents needed depends

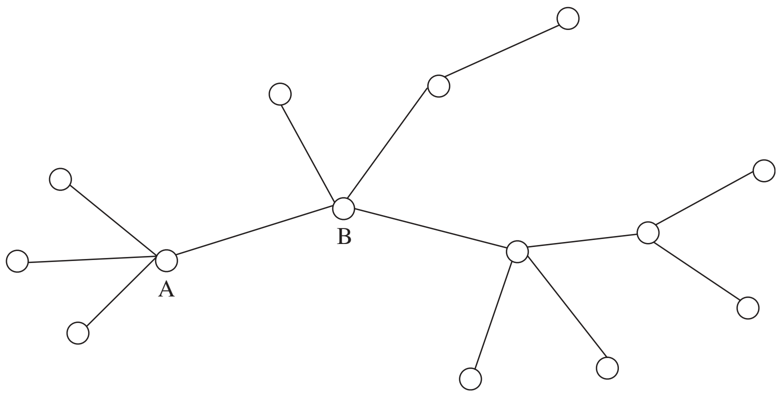

FIGURE 3.3 The number of needed agents depends on the starting node.

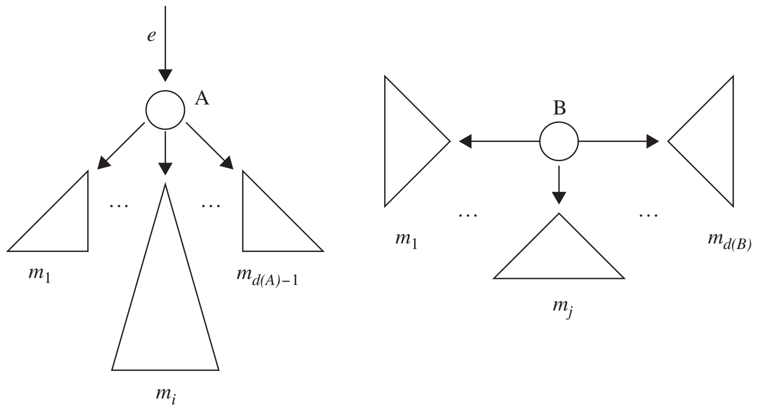

FIGURE 3.4 Determining the minimum number of cleaners.

on the node from which the team of agents starts. Consider for example the tree shown in Figure 3.3. If the team starts from node A, then two agents suffice. However, the reader can verify that at least three agents are needed if they start from node B.

In [4], the authors describe a simple and efficient strategy to determine the minimum number of agents necessary to decontaminate an arbitrary given tree from any initial starting node. The strategy is based on the following two observations.

Consider a node A. If A is not the starting node, the agents will arrive at A for the first time from some link e (see Figure 3.4). Let $T_1(A), \ldots, T_i(A), \ldots, T_{d(A)-1}$ be the subtrees of A from the other incident links, where $d(A)$ denotes the degree of A; let m_i denote the number of agents needed to decontaminate $T_i(A)$ once the agents are at A, and let $m_i \leq m_{i+1}$, $1 \leq i \leq d(A)-2$. The first observation is that to decontaminate A and all its other subtrees without recontamination the number $m(A, e)$ of agents needed is

$$m(A,e) = \begin{cases} m_1 & \text{if } m_1 > m_2 \\ m_{1+1} & \text{if } m_1 = m_2 \end{cases}$$

Consider now a node B and let $m_j(B)$ be the minimum number of agents needed to decontaminate the subtree $T_j(B)$ once the agents are at B and let $m_j \leq m_{j+1}$, $1 \leq j \leq d(B)$. The second observation is that to decontaminate the entire tree starting from B the number $m(B)$ of agents needed is

$$m(B) = \begin{cases} m_1 & \text{if } m_1 > m_2 \\ m_{1+1} & \text{if } m_1 = m_2 \end{cases}$$

Based on these two properties, the authors show in [4] how determination of the optimal number of agents can be done through a saturation where appropriate information about the structure of the tree is collected from the leaves and propagated along the tree until the optimal is known for each possible starting point. The most interesting aspect of this strategy is that it yields immediately a decontamination protocol for trees that uses exactly that minimum number of agents. In other words, the technique of [4] allows to determine the minimum number of agents and the corresponding decontamination strategy for every starting network, and this is done exchanging only $O(n)$ short messages [or, serially, in $O(n)$ time].

The trees requiring the largest number of agents are *complete binary trees,* where the number of agent is $O(\log n)$; by contrast, in the *line* two agents are always sufficient.

3.3.4.2 Hypercubes

It has been shown in [85] that to decontaminate a hypercube of size n, $\Theta(n/\sqrt{\log n})$ agents are necessary and sufficient. The employ of an optimal number of agents in the local model has an interesting consequence; in fact, it implies that $\Theta(n/\sqrt{\log n})$ is the search number for the hypercube in the classical model, that is, where agents may "jump."

In the algorithm for the local model one of the agents acts as a *coordinator* for the entire cleaning process. The cleaning strategy is carried out on the broadcast tree of the hypercube. The cleaning strategy broadcast tree of the hypercube; see Figure 3.5. The main idea is to place enough agents on the homebase and to have them move, level by level, on the edges of the broadcast tree, leaded by the coordinator in such a way that no recontamination may occur. The number of moves and the ideal time complexity of this strategy are indicated in Table 3.1.

The visibility assumption allows the agents to make their own decision regarding the action to take solely on the basis of their local knowledge. In fact, the agents are still moving on the broadcast tree, but they do not have to follow the order imposed by the coordinator. The agents on node x can proceed to

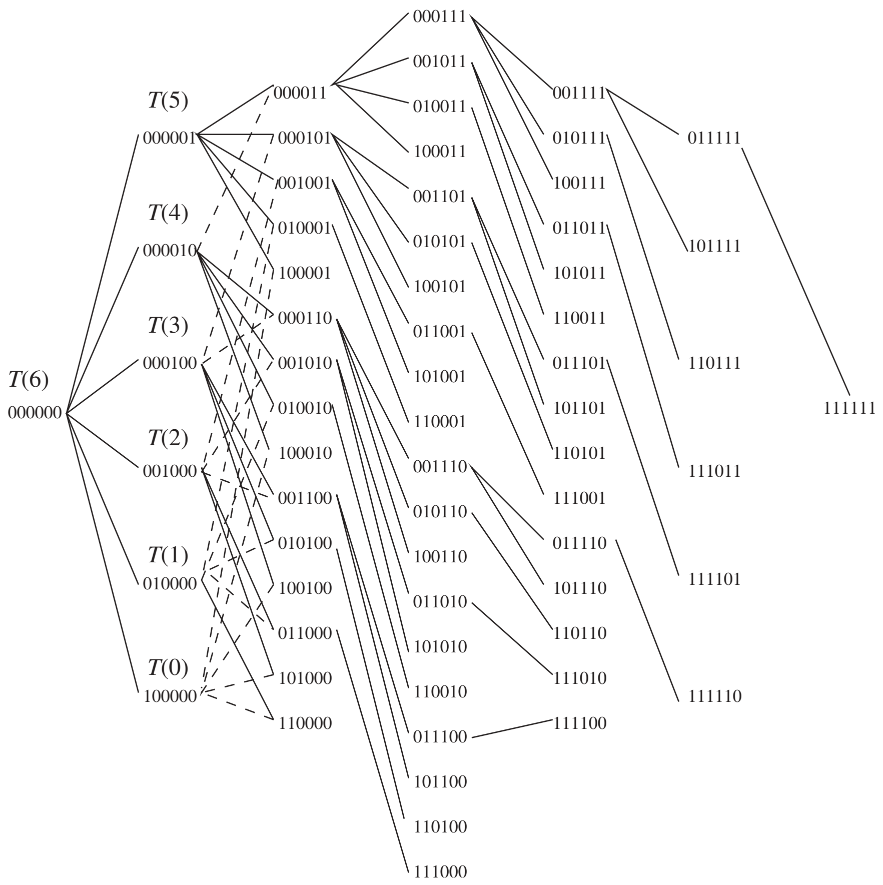

FIGURE 3.5 The broadcast tree T of the hypercube H_6. Normal lines represent edges in T, dotted lines (only partially shown) the remaining edges of H_6.

TABLE 3.1 Decontamination of Hypercube

Model	Agents	Time	Moves
Local	$(\star)O(n/\sqrt{\log n})$	$O(n \log n)$	$O(n \log n)$
Local, cloning, synchronicity	$n/2$	$(\star) \log n$	$(\star)\ n-1$
Visibility	$n/2$	$(\star) \log n$	$O(n \log n)$
Visibility and cloning	$n/2$	$(\star) \log n$	$(\star)\ n-1$

Note: The star indicates an optimal bound.

clean the children of x in the broadcast tree when they see that the other neighbors of x are either clean or guarded. With this strategy the time complexity is drastically reduced (since agents move concurrently and independently) but the number of agents increases. Other variations of those two models have been studied and summarized in Table 3.1.

A characterization of the impact that these additional assumptions have on the problem is still open. For example, an optimal move complexity in the local model with cloning has not been found, and it is not clear whether it exists; when the agents have visibility, synchronicity has not been of any help, although it has not been proved that it is indeed useless; the use of an optimal number of agents in the weaker local model is obtained at the expense of employing more agents, and it is not clear whether this increment is necessary.

3.3.4.3 Chordal Rings

The local and the visibility models have been the subject of investigation also in the chordal ring topology in [86].

Let $C(\langle d_1 = 1, d_2, \ldots, d_k \rangle)$ be a chordal ring network with n nodes and link structure $\langle d_1 = 1, d_2, \ldots, d_k \rangle$, where $d_i < d_{i+1}$ and $d_k \leq \lfloor n/2 \rfloor$. In [86] it is first shown that the smallest number of agents needed for the decontamination depends not on the size of the chordal ring but solely on the *length* of the longest chord. In fact, any solution of the contiguous decontamination problem in a chordal ring $C(\langle d_1 = 1, d_2, \ldots, d_k \rangle)$ with $4 \leq d_k \leq \sqrt{n}$ requires at least $2d_k$ searchers ($2d_k + 1$ in the visibility model).

In both models, the cleaning is preceded by a deployment stage after which the agents have to occupy $2d_k$ consecutive nodes. After the deployment, the decontamination stage can start. In the local model, nodes x_0 to x_{d_k-1} are constantly guarded by one agent each, forming a window of d_k agents. This window of agents will shield the clean nodes from recontamination from one direction of the ring while the agents of the other window are moved by the coordinator (one at a time starting from the one occupying node x_{d_k}) along their longest chord to clean the next window in the ring. Also in the case of the chordal ring, the visibility assumption allows the agents to make their own decision solely on the basis of their local knowledge: An agent moves to clean a neighbor only when this is the only contaminated neighbor.

Figure 3.6 shows a possible execution of the algorithm in a portion of a chordal ring $C(\langle 1, 2, 4 \rangle)$. Figure 3.6*a* shows the guarded nodes (in black) after the deployment phase. At this point, the nodes indicated in the figure can independently and concurrently start the cleaning phase, moving to occupy their only contaminated neighbor. Figure 3.6*b* shows the new state of the network if they all move (the arrows indicate the nodes where the agents could move to clean their neighbor).

The complexity results in the two models are summarized in Table 3.2.

Consistent with the observations for the hypercube, also in the case of the chordal ring the visibility assumption allows to drastically decrease the time complexity (and in this case also the move complexity). In particular, the strategies for the visibility model are optimal in terms of both number of agents and number of moves; as for the time complexity, visibility allows some concurrency (although it does not bring this measure to optimal as was the case for the hypercube).

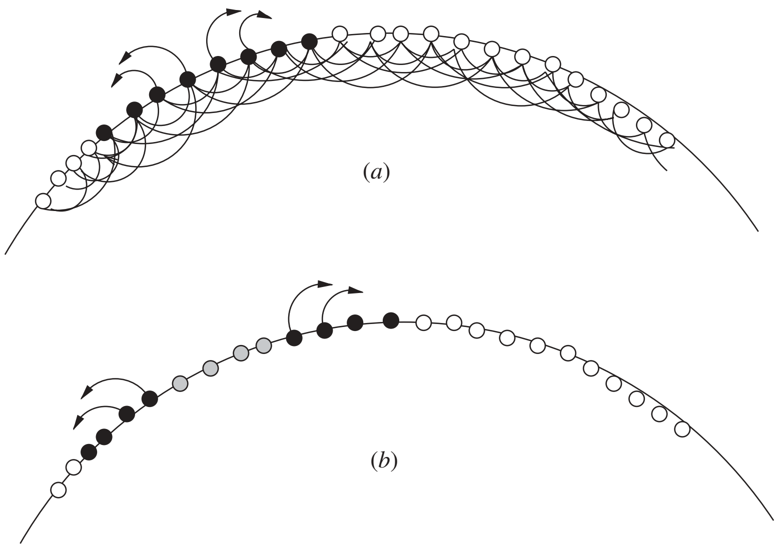

FIGURE 3.6 A chordal ring $C(\langle 1, 2, 4\rangle)$. (*a*) The agents are deployed and four of them (the ones pointed by an arrow) could move to clean the neighbor. (*b*) Four agents have moved to clean their only contaminated neighbor and four more (the ones pointed by an arrow) could now move.

TABLE 3.2 Results for Chordal Ring

Model	Agents	Time	Moves
Local	$2d_k + 1$ ($\star$)	$3n - 4d_k - 1$	$4n - 6d_k - 1$
Visibility	$2d_k$ ($\star$)	$\left\lceil \dfrac{n - 2d_k}{2(d_k - d_k - 1)} \right\rceil$	$n - 2d_k$ ($\star$)

Note: The star indicates an optimal bound.

3.3.4.4 Tori

A lower bound for the torus has beed derived in [86]. Any solution of the decontamination problem in a torus $T(h, k)$ with $h, k \geq 4$ requires at least 2 min $\{h, k\}$ agents; in the local model it requires at least 2 min$\{h, k\} + 1$ agents. The strategy that matches the lower bound is very simple. The idea is to deploy the agents to cover two consecutive columns and then keep one column of agents to guard from decontamination and have the other column move along the torus. The complexity results are summarized in Table 3.3. As for the other topologies, visibility decreases time and slightly increases the number of agents. In the case of the torus it is interesting to notice that in the visibility model all three complexity measures are optimal.

TABLE 3.3 Results for Two-Dimensional Torus with Dimensions $h, k, h \leq k$

Model	Agents	Time	Moves
Local	$2h+1$ $(\star)$	$hk-2h$	$2hk-4h-1$
Visibility	$2h$ $(\star)$ $(\star)$	$\left\lceil \frac{1}{2}\frac{k-2}{2} \right\rceil (\star)$	$hk-2h$ $(\star)$ $(\star)$

Note: The star indicates an optimal bound.

TABLE 3.4 Results for d-Dimensional Torus $T(h_1, h_2, \ldots, h_d)$

Model	Agents	Time	Moves
Local	$2(N/h_d)+1$	$N-2(N/h_d)$	$2N-4(N/h_d)-1$
Visibility	$2(N/h_d)$	$(\lceil h_d-2 \rceil)/2$	$N-2(N/h_d)$

Finally, these simple decontamination strategies can be generalized to d-dimensional tori (although the lower bounds have not been generalized). Let $T(h_1, \ldots, h_d)$ be a d-dimensional torus and let $h_1 < h_2 \leq \cdots \leq h_d$. Let N be the number of nodes in the torus and let $H = N/h_d$. The resulting complexities are reported in Table 3.4.

3.3.5 Different Contamination Rules

In [87] the network decontamination problem has been considered under a new model of *neighborhood-based immunity* to recontamination: a clean node, after the cleaning agent has gone, becomes recontaminated only if a weak majority of its neighbors are infected. This recontamination rule is called *local immunization*. Luccio et al. [87] studied the effects of this level of immunity on the nature of the problem in tori and trees. More precisely, they establish lower bounds on the number of agents necessary for decontamination and on the number of moves performed by an optimal-size team of cleaners and propose cleaning strategies. The bounds are tight for trees and for synchronous tori; they are within a constant factor of each other in the case of asynchronous tori. It is shown that with local immunization only $O(1)$ agents are needed to decontaminate meshes and tori, regardless of their size; this must be contrasted with, for example, the $2\min\{n, m\}$ agents required to decontaminate an $n \times m$ torus without local immunization [86]. Interestingly, among tree networks, binary trees were the worst to decontaminate without local immunization, requiring $\Omega(\log n)$ agents in the worst case [4]. Instead, with local immunization, they can be decontaminated by a *single* agent starting from a leaf or by two agents starting from any internal node.

A different kind of immunity has been considered in [88]: one disinfected, a node is immune to contamination for a certain amount of time. This *temporal immunity* has been investigated and characterized for tree networks [88].

3.4 CONCLUSIONS

Mobile agents represent a novel powerful paradigm for algorithmic solutions to distributed problems; unlike the message-passing paradigm, mobile agent solutions are naturally suited for dynamic environments. Thus they provide a unique opportunity for developing simple solutions to complex control and security problems arising in ever-changing systems such as dynamic networks. While mobile agents per se have been extensively investigated in the artificial intelligence, software engineering, and specification and verification communities, the algorithmic aspects (problem solving, complexity analysis, experimental evaluation) are very limited. It is only recently that researchers have started to systematically explore this new and exciting distributed computational universe. In this chapter we have described some interesting problems and solution techniques developed in this investigation in the context of security. Our focus has been on two security problems: locating a black hole and capturing an intruder. For each we have described the computational issues and the algorithmic techniques and solutions. These topics and techniques have a much wider theoretical scope and range. In particular, the problems themselves are related to long-investigated and well-established problems in automata theory, computational complexity, and graph theory.

Many problems are still open. Among them:

- The design of solutions when the harmful host represents a *transient danger*, in other words, when the harmful behavior is not consistent and continuous but changes over time.
- The study of *mobile harm,* that is, of pieces of software that are wandering around the network possibly damaging the mobile agents encountered in their path.
- The study of *multiple attacks*, in other words, the general harmful host location problem when dealing with an arbitrary, possibly unknown, number of harmful hosts present in the system. Some results have been recently obtained [54, 55]

ACKNOWLEDGMENTS

This work has been supported in part by the Natural Sciences and Engineering Research Council of Canada under the Discovery Grant program, and by Dr. Flocchini's University Research Chair.

REFERENCES

1. S. Albers and M. Henzinger, Exploring unknown environments, *SIAM J. Comput.*, 29:1164–1188, 2000.
2. S. Alpern and S. Gal, *The Theory of Search Games and Rendezvous*, Springer, New York, 2002.
3. B. Awerbuch, M. Betke, and M. Singh, Piecemeal graph learning by a mobile robot, *Inform. Comput.*, 152:155–172, 1999.
4. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro, Capture of an intruder by mobile agents, in *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Winnipeg, 2002, pp. 200–209.
5. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro, Can we elect if we cannot compare? in *Proceedings of the 15th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, San Diego, 2003, pp. 200–209.
6. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro, Rendezvous and election of mobile agents: Impact of sense of direction, *Theory Comput. Syst.*, 40(2): 143–162, 2007.
7. X. Deng and C. H. Papadimitriou, Exploring an unknown graph, *J. Graph Theory*, 32(3):265–297, 1999.
8. A. Dessmark, P. Fraigniaud, and A. Pelc, Deterministic rendezvous in graphs, *Algorithmica*, 46:69–96, 2006.
9. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc, Tree exploration with little memory, *J. Algorithms*, 51:38–63, 2004.
10. E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk, Mobile agent rendezvous in a ring, in *Proceedings of the 23rd International Conference on Distibuted Computing Systems (ICDCS)*, Providence, 2003, pp. 592–599.
11. N. Borselius, Mobile agent security, *Electron. Commun. Eng. J.*, 14(5):211–218, 2002.
12. D. M. Chess, Security issues in mobile code systems, in *Mobile Agent Security*, G. Vigna (Ed.), Lecture Notes in Computer Science, Vol. 1419, Springer, London, 1998, pp. 1–14.
13. M. S. Greenberg, J. C. Byington, and D. G. Harper, Mobile agents and security, *IEEE Commun. Mag.*, 36(7):76–85, 1998.
14. R. Oppliger, Security issues related to mobile code and agent-based systems, *Computer Commun.*, 22(12):1165–1170, 1999.
15. K. Schelderup and J. Ones, Mobile agent security—Issues and directions, in *Proceedings of the 6th International Conference on Intelligence and Services in Networks*, Lecture Notes in Computer Science, Vol. 1597, Barcelona, 1999, pp. 155–167.
16. M. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan, The power of a pebble: Exploring and mapping directed graphs, *Inform. Comput.*, 176(1):1–21, 2002.
17. A. Dessmark and A. Pelc, Optimal graph exploration without good maps, *Theor. Computer Sci.*, 326:343–362, 2004.
18. P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc, Collective tree exploration, *Networks*, 48(3):166–177, 2006.
19. P. Fraigniaud, and D. Ilcinkas, Digraph exploration with little memory, in *Proceedings of the 21st Symposium on Theoretical Aspects of Computer Science (STACS)*, Montpellier, 2004, pp. 246–257.

20. P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg, Graph exploration by a finite automaton, *Theor. Computer Sci.*, 345(2–3):331–344, 2005.
21. P. Panaite and A. Pelc, Exploring unknown undirected graphs, *J. Algorithms*, 33:281–295, 1999.
22. P. Panaite and A. Pelc, Impact of topographic information on graph exploration efficiency, *Networks*, 36:96–103, 2000.
23. S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro, Map construction of unknown graphs by multiple agents, *Theor. Computer Sci.*, 385(1–3):34–48, 2007.
24. S. Das, P. Flocchini, A. Nayak, and N. Santoro, Effective elections for anonymous mobile agents, in *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC)*, Kolkata, 2006, pp. 732–743.
25. E. Kranakis, D. Krizanc, and S. Rajsbaum, Mobile agent rendezvous, in *Proceedings of the 13th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Chester, 2006, pp. 1–9.
26. J. Ellis, H. Sudborough, and J. Turner, The vertex separation and search number of a graph, *Inform. Comput.*, 113(1):50–79, 1994.
27. L. Kirousis and C. Papadimitriou, Searching and pebbling, *Theor. Computer Sci.*, 47(2):205–218, 1986.
28. A. Lapaugh, Recontamination does not help to search a graph, *J. ACM*, 40(2): 224–245, 1993.
29. N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou, The complexity of searching a graph, *J. ACM*, 35(1):18–44, 1988.
30. T. Parson, The search number of a connected graph, in *Proceedings of the 9th Southeastern Conference on Combinatorics, Graph Theory and Computing*, Utilitas Mathematica, Boca Raton, 1978, pp. 549–554.
31. D. Bienstock and M. Langston, Algorithmic implications of the graph minor theorem, in M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser (Eds.), *Handbook of Operations Research and Management Science: Network Models*, Vol. 7, Elsevier, Amsterdam, 1995, pp. 481–502.
32. L. Kirousis and C. Papadimitriou, Interval graphs and searching, *Discrete Math.*, 55:181–184, 1985.
33. F. Makedon and H. Sudborough, Minimizing width in linear layout, in *Proceedings of the 10th International Colloquium on Automata, Languages, and Programming* (*ICALP '83*), Barcelona, 1983, pp. 478–490.
34. P. Seymour and R. Thomas, Graph searching, and a min-max theorem for treewidth, *J. Comb. Theory, Ser. B*, 58(1):22–33, 1993.
35. F. Hohl, Time limited blackbox security: Protecting mobile agents from malicious hosts, in G. Vigna (Ed.), *Mobile Agent Security*, Lecture Notes in Computer Science, Vol. 1419, Springer, London, 1998, pp. 92–113.
36. T. Sander, and C. F. Tschudin, Protecting mobile agents against malicious hosts, in G. Vigna (Ed.), *Mobile Agent Security*, Lecture Notes in Computer Science, Vol. 1419, Springer, London, 1998, pp. 44–60.
37. J. Vitek, and G. Castagna, Mobile computations and hostile hosts, in D. Tsichritzis (Ed.), *Mobile Objects*, University of Geneva, Geneva, 1999, pp. 241–261.
38. G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, Robotic exploration as graph construction, *Trans. Robot. Autom.*, 7(6):859–865, 1991.

39. M. Bender, and D. K. Slonim, The power of team exploration: Two robots can learn unlabeled directed graphs, in *Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS)*, Santa Fe, 1994, pp. 75–85.
40. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro, Mobile search for a black hole in an anonymous ring, *Algorithmica*, 48(1):67–90, 2007.
41. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro, Searching for a black hole in arbitrary networks: Optimal mobile agents protocols, *Distrib. Comput.* 19(1):1–19, 2006.
42. P. Flocchini, B. Mans, and N. Santoro, Sense of direction in distributed computing, *Theor. Computer Sci.*, 291:29–53, 2003.
43. S. Dobrev, P. Flocchini, and N. Santoro, Improved bounds for optimal black hole search in a network with a map, in *Proceedings of the 10th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Smolenice Castle, 2004, pp. 111–122.
44. S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, and N. Santoro, Optimal search for a black hole in common interconnection networks, *Networks*, 47(2):61–71, 2006.
45. S. Dobrev, P. Flocchini, and N. Santoro, Cycling through a dangerous network: A simple efficient strategy for black hole search, in *Proceedings of the 26th International Conference on Distributed computing Systems (ICDCS)*, Lisboa, 2006, p. 57.
46. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro, Multiple agents rendezvous in a ring in spite of a black hole, in *Proceedings of the 6th International Symposium on Principles of Distributed Systems (OPODIS)*, La Martinique, 2003, pp. 34–46.
47. S. Dobrev, P. Flocchini, R. Kralovic, and N. Santoro, Exploring a dangerous unknown graph using tokens, in *Proceedings of the 5th IFIP International Conference on Theoretical Computer Science (TCS)*, Santiago, 2006, pp. 131–150.
48. S. Dobrev, N. Santoro, and W. Shi, Using scattered mobile agents to locate a black hole in an unoriented ring with tokens, *Intl. J. Found. Comput. Sci.*, 19(6):1355–1372, 2008.
49. P. Flocchini, D. Ilcinkas, and N. Santoro, Ping pong in dangerous graphs: Optimal black hole search with pebbles, *Algorithmica*, 62(3–4):1006–1033, 2012.
50. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc, Searching for a black hole in synchronous tree networks, *Combinator. Prob. Comput.*, 16:595–619, 2007.
51. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc, Complexity of searching for a black hole, *Fund. Inform.*, 71(2–3):229–242, 2006; 35–45, 2004.
52. R. Klasing, E. Markou, T. Radzik, and F. Sarracco, Approximation bounds for black hole search problems. *Networks*, 52(4):216–226, 2008.
53. R. Klasing, E. Markou, T. Radzik, and F. Sarracco, Hardness and approximation results for Black Hole Search in arbitrary networks, *Theor. Computer Sci.*, 384(2–3): 201–221, 2007.
54. C. Cooper, R. Klasing, and T. Radzik, Searching for black-hole faults in a network using multiple agents, in *Proceedings of the 10th International Conference on Principle of Distributed Systems (OPODIS)*, Bordeaux, 2006, pp. 320–332.
55. J. Chalopin, S. Das, and N. Santoro, Rendezvous of mobile agents in unknown graphs with faulty links, in *Proceedings of the 21st International Symposium on Distributed Computing (DISC)*, Lemesos, 2007, pp. 108–122.

56. W. Jansen, Intrusion detection with mobile agents, *Computer Communications*, 25(15): 1392–1401, 2002.
57. E. H. Spafford, and D. Zamboni, Intrusion detection using autonomous agents, *Computer Networks*, 34(4):547–570, 2000.
58. D. Ye, Q. Bai, M. Zhang, and Z. Ye, Distributed intrusion detections by using mobile agents, in *Proceedings of the 7th IEEE/ACIS International Conference on Computer and Information Science (ICIS)*, Portland, 2008, pp. 259–265.
59. R. Breisch, An intuitive approach to speleotopology, *Southwestern Cavers* VI(5): 72–78, 1967.
60. T. Parson, Pursuit-evasion in a graph, in *Proceedings of Conference on Theory and Applications of Graphs*, Lecture Notes in Mathematics, Springer, Michigan, 1976, pp. 426–441.
61. D. Bienstock and P. Seymour, Monotonicity in graph searching, *J. Algorithms*, 12:239–245, 1991.
62. N. Kinnersley, The vertex separation number of a graph equals its path-width, *Inform. Process. Lett.*, 42(6):345–350, 1992.
63. F. V. Fomin and D. M. Thilikos, An annotated bibliography on guaranteed graph searching, *Theor. Comput. Sci.*, 399(3):236–245, 2008.
64. I. Suzuki and M. Yamashita, Searching for a mobile intruder in a polygonal region, *SIAM J. Comput.*, 21(5):863–888, 1992.
65. A. Takahashi, S. Ueno, and Y. Kajitani, Mixed searching and proper-path-width, *Theor. Comput. Sci.*, 137(2):253–268, 1995.
66. D. Bienstock, Graph searching, path-width, tree-width and related problems, (A Survey), in *Proceedings of DIMACS Workshop on Reliability of Computer and Communication Networks*, Rutgers University, 1991, pp. 33–49.
67. D. Thilikos, Algorithms and obstructions for linear-width and related search parameters, *Discrete Appl. Math.,* 105:239–271, 2000.
68. R. Chang, Single step graph search problem, *Inform. Process. Lett.* 40(2):107–111, 1991.
69. N. Dendris, L. Kirousis, and D. Thilikos, Fugitive-search games on graphs and related parameters, *Theor. Comput. Sci.*, 172(1–2):233–254, 1997.
70. F. Fomin and P. Golovach, Graph searching and interval completion, *SIAM J. Discrete Math.*, 13(4):454–464, 2000.
71. J. Smith, Minimal trees of given search number, *Discrete Math.*, 66:191–202, 1987.
72. Y. Stamatiou and D. Thilikos, Monotonicity and inert fugitive search games, in *Proceedings of the 6th Twente Workshop on Graphs and Combinatorial Optimization*, Electronic Notes on Discrete Mathematics, Vol. 3, Enschede, 1999, pp. 184.
73. H. Buhrman, M. Franklin, J. Garay, J.-H. Hoepman, J. Tromp, and P. Vitányi, Mutual search, *J. ACM*, 46(4), 517–536, 1999.
74. T. Lengauer, Black-white pebbles and graph separation, *Acta Inform.*, 16(4): 465–475, 1981.
75. S. Neufeld, A pursuit-evasion problem on a grid, *Inform. Process. Lett.*, 58(1):5–9, 1996.
76. I. Suzuki, M. Yamashita, H. Umemoto, and T. Kameda, Bushiness and a tight worst-case upper bound on the search number of a simple polygon, *Inform. Process. Lett.*, 66(1):49–52, 1998.

77. B. von Stengel and R. Werchner, Complexity of searching an immobile hider in a graph, *Discrete Appl. Math.*, 78:235–249, 1997.
78. M. Yamamoto, K. Takahashi, M. Hagiya, and S.-Y. Nishizaki, Formalization of graph search algorithms and its applications, in *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics*, Canberra, 1998, pp. 479–496.
79. P. Fraigniaud and N. Nisse, Monotony properties of connected visible graph searching, *Inf. Comput.*, 206(12):1383–1393, 2008.
80. D. Ilcinkas, N. Nisse, and D. Soguet, The cost of monotonicity in distributed graph searching, *Distributed Comput.*, 22(2):117–127, 2009.
81. L. Barrière, P. Fraigniaud, N. Santoro, and D. M. Thilikos, Searching is not jumping, in *Proceedings of the 29th International Workshop on Graph Theoretic Concepts in Computer Science (WG)*, Lecture Notes in Computer Science, Vol. 2880, Elspeet, 2003, pp. 34–45.
82. B. Yang, D. Dyer, and B. Alspach, Sweeping graphs with large clique number, *Discrete Math.*, 309(18):5770–5780, 2009.
83. P. Flocchini, A. Nayak, and A. Shulz, Cleaning an arbitrary regular network with mobile agents, in *Proceedings of the 2nd International Conference on Distributed Computing and Internet Technology (ICDCIT)*, Bhubaneswar, 2005, pp. 132–142.
84. L. Blin, P. Fraigniaud, N. Nisse, and S. Vial, Distributed chasing of network intruders, *Theor. Comput. Sci.*, 399(1–2):12-37, 2008.
85. P. Flocchini, M. J. Huang, and F. L. Luccio, Decontamination of hypercubes by mobile agents, *Networks,* 52(3):167–178, 2008.
86. P. Flocchini, M. J. Huang, and F. L. Luccio, Decontamination of chordal rings and tori using mobile agents, *Int. J. Found. Computer Sci.*, 18(3):547–564, 2007.
87. F. Luccio, L. Pagli, and N. Santoro, Network decontamination in presence of local immunity, *Int. J. Found. Comput. Sci.*, 18(3):457–474, 2007.
88. P. Flocchini, B. Mans, and N. Santoro, Tree decontamination with temporary immunity, in *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC)*, Gold Coast, 2008, pp. 330–341.