Distributed Security Algorithms for Mobile Agents

Paola Flocchini[*] Nicola Santoro[†]

February 1, 2008

Abstract

The use of *mobile agents* is becoming increasingly popular when computing in networked environments, ranging from Internet to the Data Grid, both as a theoretical computational paradigm and as a system-supported programming platform. In spite of this, mobile agents systems have been largely ignored by the mainstream distributed computing community. It is only recently that several researchers have started to systematically explore this new and exciting distributed computational universe. In this paper we describe some of interesting problems and solution techniques developed in this investigations in the context of security. In fact, at a practical level, in systems supporting mobile agents, *security* is the most pressing concern, and possibly the most difficult to address. In particular, specific severe security threats are those posed to the network site by *harmful agents*, and those posed to the mobile agents by *harmful hosts*. 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. 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.

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, web crawlers, etc. Further, they allow to express immediately notions such as selfish behaviour,

[*]SITE, University of Ottawa, email:flocchin@site.uottawa.ca
[†]SCS, Carleton University, email:santoro@scs.carleton.ca

negotiation, cooperation, etc 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 that of object-oriented programming. As a computational paradigm, mobile agents 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 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. Example of basic tasks are *wakeup*, *traversal*, *rendez-vous*, *election*. The coordination of the agents necessary to perform these tasks is not necessarily simple or easy to achieve. In fact, the computational problems related to these operations are definitely non trivial, 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; e.g., see [1, 2, 4, 5, 6, 7, 25, 27, 29, 63].

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 *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 of the amount of effort, must still be effectively addressed (e.g., see [16, 19, 53, 56, 71, 80]).

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 non-cooperative settings such as Internet (e.g., e-mail transmitted viruses). The latter not only exists in those settings, but also in environments with regulated access and where agents cooperate towards 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 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; i.e., 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 finite time, at least one agent survives and knows the location of the black hole.

We then consider (in Section 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 anti-viral 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* (e.g., see [1, 9, 25, 28, 29, 48, 49, 50, 72, 73]). 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* (e.g. see [2, 6, 7, 23, 24, 64]). The *network decontamination* problem is instead related to the classical problem known as *graph search* (e.g., see [39, 59, 65, 69, 75]), 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., see [13, 58, 68, 78]).

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.

2 Black Hole Search

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 (e.g., see [55, 77, 87]). Obviously, the first step in any solution to such a problem must be to *identify*, if possible, the harmful host; i.e., 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* (shortly 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 a 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), the *cost* (i.e., the number of moves performed by the agents executing a size-optimal solution protocol). Sometimes also bounded *time* complexity is 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; i.e., the system is *asynchronous*. Moreover, it is usually assumed that each node of the network provides a storage area called *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 has been visited and are found to be safe. Clearly, in this exploration process some agents may disappeared 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.

2.2 A 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 *labelled* graphs typically emphasize minimizing the number of moves or the amount of memory used by the agent (e.g., see [1, 25, 28, 72, 73]). 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 [29, 48]). 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 [9] 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 [10] 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 [49] for exploring directed graphs and by Fraigniaud et al. [48] for exploring trees. In [29, 49, 50] 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 [10], whereas Fraigniaud et al. [48] showed how k agents can explore a tree. In both these cases, the agents start from same node and they have distinct identities. In [7] 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 are co-prime 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, like *Agent Election*, *Labelling* and *Rendezvous*. Among them rendezvous is probably the most investigated; for a recent account see [2, 64].

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.

2.3.1 Cautious Walk

We now describe a basic tool (from [30]) 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*.

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, i.e. the impossibility to distinguish the BH from a slow node). For example [30]:

- 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 black hole if the size of G is not known.
- For asynchronous agents it is impossible to verify if there is a back hole.

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 black hole.

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.

2.4 Impact of Knowledge

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 Bh; that is, within finite time, at least one agent must terminate with a map of the entire graph where the home-base, 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 $d(\text{Bh})$ 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 can not be resolved in finite time nor 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 need to be wrong only on at most $\Delta - d(\text{Bh})$ links. Hence, we require from a solution algorithm $\mathcal{P}$ 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 [32]. 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 [32].

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 neighbours. During this process, the home base 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 home base before starting to look for a new node to expand. The whiteboard on the homebase is used to store information about the nodes that have been already 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.

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 [44]). In this case, two agents suffice to locate the black hole, regardless of the (unknown) topological structure of G. The proof of [32] 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

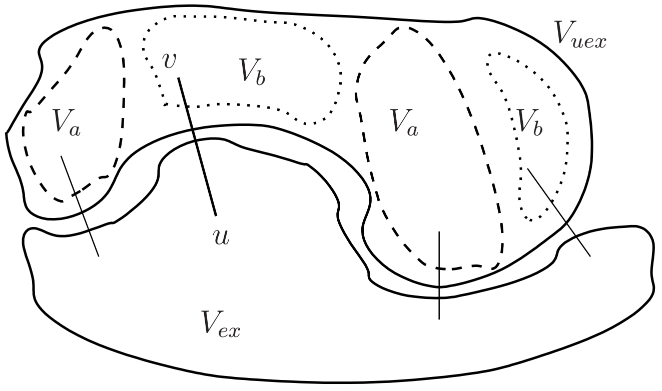

Figure 1: Splitting the unexplored subgraph G_{uex} into G_a and G_b.

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 insuring that one of the two will eventually succeed.

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 then the more common *topological awareness* (i.e., knowledge of the class of the network, but not of its size nor of the source location – e.g. being in a mesh, starting from an unknown position).

Also in this case, two agents suffice [32]; 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.

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; finally reaching the last safe node w' reached by b.

Agent a then computes the new subgraph G_{uex} containing all non-safe nodes. 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 destroyed by the black hole.

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 [34]. 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 and (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 in 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 $O(n/logn)$ diameter in which the previous protocols [32, 33] 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.

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 [36].

Let C be a set of simple cycles such that each vertex of G is covered by a cycle from

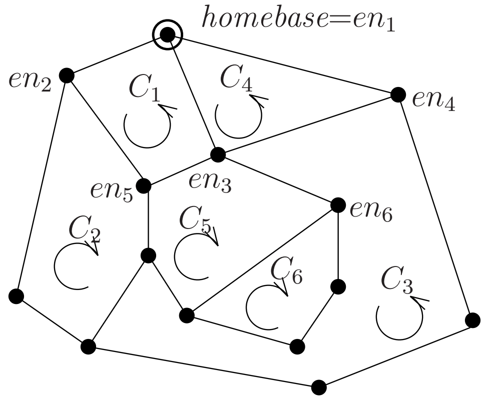

Figure 2: Example of *Cycle-DFT Sequence* for graph G and $= \{C_1, C_2, C_3, C_4, C_5, C_6\}$. The cycle directions are shown, as well as the resulting entry nodes for each cycle. The resulting *Cycle-DFT Sequence*: $L = \{1, 2, 5, 3, 6, 3, 4\}$.

C. Such a set of cycle, with some connectivity constraint is called *Open Vertex Cover* by cycles. The algorithm is based on the pre-computation of such an open vertex cover by cycles of a graph. The idea is to explore the graph G by exploring the cycles 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 three and more (e.g., hypercubes, multi-dimensional tori, etc,), as well as many non-Abelian cube graphs (e.g., CCC, butterfly, wrapped-butterfly networks, etc.). For some of these networks, this is the only algorithm achieving such a bound.

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.

2.5.1 Rings

The problem has been investigated and its solutions characterized for *ring* networks [30]. A $Omega(n \log n)$ lower bound holds since $\Omega(n \log n)$ moves are needed by any two-agents solution [30].

An agent and move optimal solution exists, based on a partitioning of the ring and on a non-overlapping exploration by the agent. The solution is similar (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 exploring (cautiously) roughly half of the ring. One of the two agents will necessarily succeed (say agent A), while the other (agent

B) might be moving slowly or be trapped into the black hole. The successful agent follows the safe trace of the other one; at the last safe node reached by following the traces, A writes a message on the whiteboard for B indicating that it will now take charge of half of the area already to be explored. In this way, if B comes back during its cautious walk, it will find the message and will act accordingly.a Notice that the size of the ring must be known for the algorithm to work, but notice also that without knowing the size the problem is unsolvable. The key point of the algorithm's correctness is that the agents are always exploring disjoint areas and that there is a single black hole. The time complexity of this solution is $O(n \log n)$.

Interestingly, increasing the number of agents the number of moves cannot decrease, but the time to finish the exploration does [30]. 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 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 precede u counterclockwise. Only one node 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 agent 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, in spite of the presence of a black hole. The problem is studied in [31] 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 non-trivial 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 [30] .

2.5.2 Interconnection Networks

The negative result for rings does not generalizes. 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*, *star graphs* [33].

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 [33] there is also the algorithm for constructing a traversal pair of a biconnected graph.

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 inter-agent 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. [35, 37]. Some natural questions immediately arise: is the Bhs problem is 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 [35] 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 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 details in [37]. There it has been shown that the 2-agents $\Theta(n \log n)$-moves 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 use 0(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 same size and cost) that uses only whiteboards of constant size; the converse is not true.

These results indicate that, although tokens are 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.

An open problem, in the case of unknown topologies, is whether the problem is solvable when the tokens can be placed only on nodes (like in classical exploration algorithms with pebbles).

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, one of the major problem when designing an algorithm for the asynchronous case 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 neighbouring 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 2-node connected; thus, the 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 [21]. In the case of general networks the problem of finding the optimal strategy is shown to be NP-hard [22, 61] and approximation algorithms are given in [21] and subsequently improved in [60, 61]. The case of multiple black holes have been very recently investigated in [20] where a lower bound on the cost and close upper bounds are given.

3 Intruder Capture and Network Decontamination

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 "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 (e.g., see [3, 47, 81]).

Assume the nodes of the network are initially *contaminated* and we want to deploy a team of agents to *clean* (or decontaminate) the whole network. The cleaning of a node occurs when an agent transits on the node; however, when a node is left without protection (no agents on it) it might become *re-contaminated* according to a recontamination rule. The most common recontamination rule is that as soon as a node without an agent on it has a contaminated neighbour, it will become contaminated again.

3.1 A Background Problem: Graph Search

A variation of the decontamination problem described above has been extensively studied in the literature under the name of *graph search* (e.g., see [39, 59, 65, 69, 75]).

The graph search problem has been first discussed by Breisch [14], and by Parson [74, 75]. In the graph-searching problem, we are given a "contaminated" network, i.e., 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, (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 are depending 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*.

Megiddo, Hakimi, Garey, Johnson and Papadimitriou [69] proved that determining whether $s(G) \leq k$ is NP-complete. They gave an $O(n)$-time algorithm to determine the search number of n-node trees, and an $O(n \log n)$-time algorithm to determine a minimal search strategy in n-node trees. Ellis, Sudborough and Turner [39] linked $s(G)$ with the vertex separation $\mathrm{vs}(G)$ of G (known to be equal to the pathwidth of G [57]). Given an n-node network $G = (V, E)$, $\mathrm{vs}(G)$ is defined as the minimum, taken over all (one-to-one) linear layouts $L : V \to \{1, \dots, n\}$, of $\mathrm{vs}_L(G)$, the latter being defined as the maximum, for $i = 1, \dots, n$, 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 that $s(G) = \mathrm{vs}(G')$ where G' is the 2-augmentation of G, i.e., 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.

Beside network security [54], the graph-searching problem has many other applications, including pursuit-evasion problems in a labyrinth [74], decontamination problems in a system of tunnels, and mobile computing problems in which agents or robots [40] are looking for an hostile intruder [83]. Moreover, the graph-searching problem also arises in VLSI design through its equivalence with the gate matrix layout problem [57]. 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 (node-search, mixed-search, t-search, etc.), is closely related to standard graph parameters and concepts, including tree-width, cut-width, path-width, and, last but not least, graph minors [13]. For instance, Makedon and Sudborough [68] showed that $s(G)$ is equal to the cutwidth 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 [58], and to its vertex separator plus one [59]. Seymour and Thomas [78] showed that the t-search number is equal to the tree-width plus one. Takahashi, Ueno and Kajitani [85] showed that the mixed-search number is equal to the proper path-width. In [12], Bienstock and Seymour simplified the proof of Lapaugh's result [65] stating that there is a minimal search strategy that does not recontaminate any link (see also [11]). Thilikos [86] 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 [18, 26, 46, 79, 82]. Contributions to related search problems can be found in [17, 66, 70, 83, 84, 88, 89] and the references therein.

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. This additional constraint has been introduced and first studied in [5] resulting in a *contiguous, monotone, node search* or *intruder capture* problem. With the contiguous assumption the nature of the problem changes considerably and the classical results on node and edge search do not generally apply. The problem of finding the optimal number of agents is still NP-complete for arbitrary graphs. 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 [45] and a move-exponential optimal solution has been given in [15]. Investigations on the relationship between the contiguous model and the classical one for graph search (where the agents can "jump") have been studied, for example, in [8, 51, 52].

In this Chapter we use the term *decontamination* to refer to contiguous monotone node search as defined in [5].

3.2 The 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 [5] (called *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, like the state of the node, the labels of the incident links, the other agents present at the node.

2. *Visibility* is the capability of the agent to "see" the state of its neighbors; i.e., an agent can see whether a neighboring node is guarded, 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, 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. All the results reported here apply for monotone strategies.

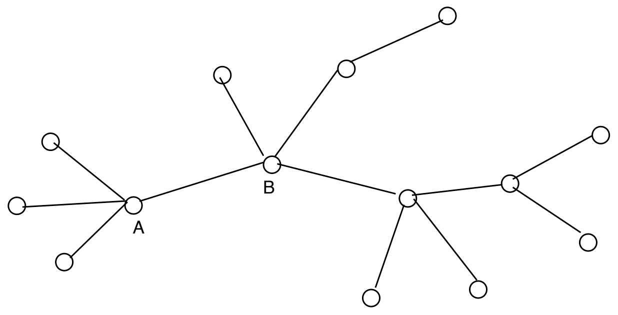

Figure 3: The number of needed agents depends on the starting node.

3.3 Results in Specific Topologies

3.3.1 Trees

The tree has been the first topology to be investigated in the Local Model [5]. First of all notice that, for a give tree T, the minimum number of agents needed depends on the node from which the team of agents start. Consider for example the tree shown in Figure 3. If the team starts from node A then two agents suffice. However, the reader can verify that at least three agent are needed if they start from node B.

In [5], 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 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

$$m(A, e) = m_1 \text{ if } m_1 > m_2 \text{ and}$$
$$m(A, e) = m_1 + 1 \text{ if } m_1 = m_2$$

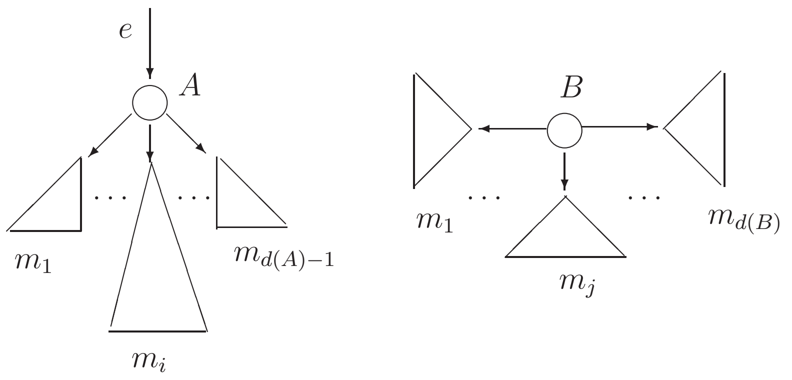

Figure 4: Determining the minimum number of cleaners.

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 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

$$m(B) = m_1 \text{ if } m_1 > m_2 \text{ and}$$
$$m(B) = m_1 + 1 \text{ if } m_1 = m_2$$

Based on these two properties, the authors show in [5] how the determination of the optimal number of agents can be done through a saturation where appropriate information about the structure of the tree are collected from the leaves and propagated along the tree, until the optimal is known for each possible starting point. The most interesting aspects of this strategy is that it yields immediately a a decontamination protocol for trees that uses exactly that minimum number of agents. In other words, the technique of [5] 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.

3.3.2 Hypercubes

It has been shown in [42] that to decontaminate a hypercube of size n, $\Theta(\frac{n}{\sqrt{\log n}})$ 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 $\Theta(\frac{n}{\sqrt{\log n}})$ is the search number for the hypercube in the classical model, i.e., 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

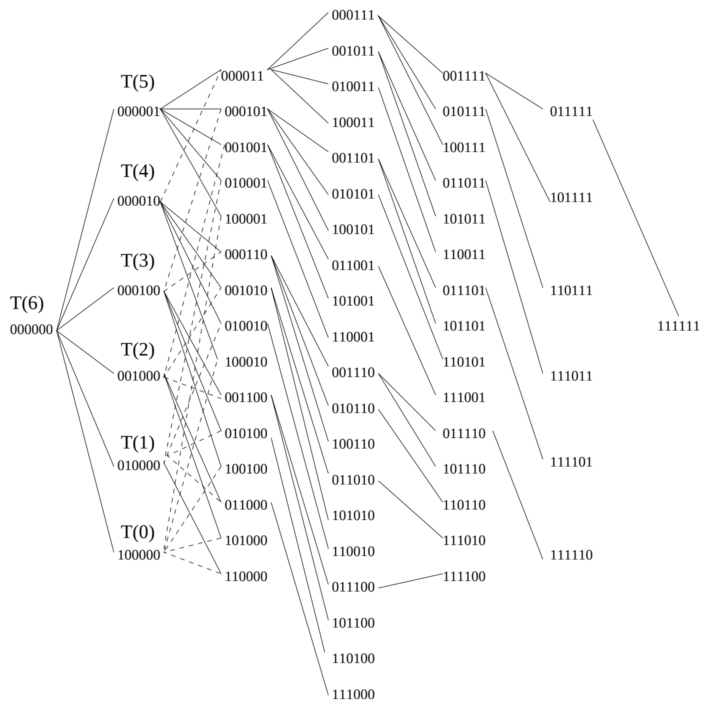

Figure 5: The broadcast tree T of the hypercube H_6. Normal lines represent edges in T, dotted lines (only partially shown) the remaining edges of H_6.

hypercube. 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 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 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 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 expenses of employing more agents and it is not clear whether this increment is necessary.

		Agents	**Time**	**Moves**
Local	**Local**	$(\star)\ O(\frac{n}{\sqrt{\log n}})$	$O(n \log n)$	$O(n \log n)$
	Local, Cloning, Synchronicity	$n/2$	$(\star)\ \log n$	$(\star)\ n-1$
Visibility	**Visibility**	$n/2$	$(\star)\ \log n$	$O(n \log n)$
	Visibility and Cloning	$n/2$	$(\star)\ \log n$	$(\star)\ n-1$

Table 1: Decontamination of the Hypercube. The star indicates an optimal bound.

3.3.3 Chordal Rings

The Local and the Visibility Models have been subject of investigation also in the Chordal Ring topology in [43].

Let $C(\langle d_1 = 1, d_2, ..., d_k \rangle)$ be a chordal ring network with n nodes and link structure $\langle d_1 = 1, d_2, ..., d_k \rangle$, where $d_i < d_{i+1}$ and $d_k \leq \lfloor \frac{n}{2} \rfloor$. In [43] it is first shown that the smallest number of agents needed for the decontamination does not depend 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 $C(\langle d_1 = 1, d_2, ..., d_k \rangle)$ with $4 \leq d_k \leq \sqrt{n}$, requires at least $2 \cdot d_k$ searchers ($2 \cdot d_k + 1$ 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 move to clean a neighbour only when this is the only contaminated neighbour.

Figure 6 shows a possible execution of the algorithm in a portion of a chordal ring $C(\langle 1, 2, 4 \rangle)$. Figure 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 neighbour. Figure 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 neighbour).

The complexity results in the two Models are summarized in Table 2.

Consistently to 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

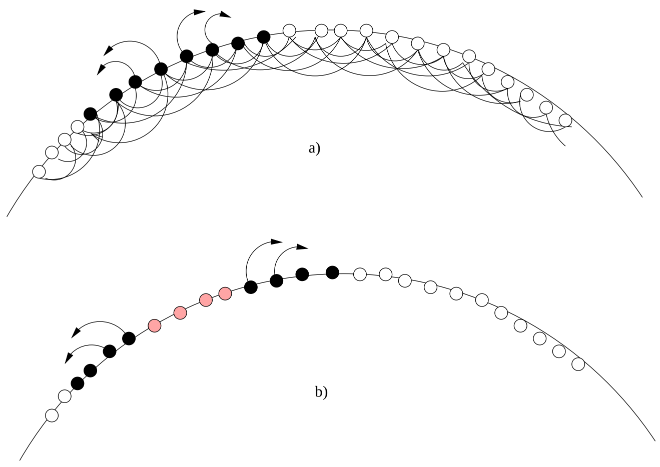

Figure 6: A chordal ring $C(\langle 1,2,4\rangle)$. *a*) The agents are deployed and four of them (the ones pointed by an arrow) could move to clean the neighbour. *b*) Four agents have moved to clean their only contaminated neighbour and four more (the ones pointed by an arrow) could now move.

Chordal Ring	**Agents**	**Time**	**Moves**
Local	$2d_k + 1$ $(\star)$	$3n - 4d_k - 1$	$4n - 6d_k - 1$
Visibility	$2d_k$ $(\star)$	$\left\lceil \frac{n-2d_k}{2(d_k - d_{k-1})} \right\rceil$	$n - 2d_k$ $(\star)$

Table 2: Results for the Chordal Ring. The star indicates an optimal bound.

are optimal both in terms of number of agents and in terms of 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).

3.3.4 Tori

A lower bound for the torus has beed derived in [43]. Any solution of the decontamination problem in a torus $T(h,k)$ with $h,k \geq 4$ requires at least $2 \cdot min\{h,k\}$ agents; in the *Local model* it requires at least $2 \cdot min\{h,k\} + 1$ 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. 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.

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

Table 3: Results for the 2-dimensional Torus with dimensions $h, k,\ h \leq k$. The star indicates an optimal bound.

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 $h_1 \leq h_2 \leq \ldots \leq h_d$. Let N be the number of nodes in the torus and let $H = \frac{N}{h_d}$. The resulting complexities are reported below.

d-dim Torus	**Agents**	**Time**	**Moves**
Local	$2\frac{N}{h_d}+1$	$N-2\frac{N}{h_d}$	$2N-4\frac{N}{h_d}-1$
Visibility	$2\frac{N}{h_d}$	$(\lceil h_d-2\rceil)/2$	$N-2\frac{N}{h_d}$

Table 4: Results for a d-dimensional Torus $T(h_1, h_2, \ldots, h_d)$.

3.4 Different Contamination Rules

In [67] the network decontamination problem has been considered under a new model of *immunity* to recontamination: a clean node, after the cleaning agent has gone, becomes recontaminated only if a weak majority of its neighbours are infected. This recontamination rule is called *local immunization.* The paper studies the effects of this level of immunity on the nature of the problem in tori and trees. More precisely, it establishes 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 it proposes 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 e.g. the $2\min\{n, m\}$ agents required to decontaminate a $n \times m$ torus without local immunization [43]. Interestingly, among tree networks, binary trees were the worst to decontaminate without local immunization, requiring $\Omega(\log n)$ agents in the worst case [5]. Instead, with local immunization, they can be decontaminated by a *single* agent.

4 Conclusions

Mobile agents represent a novel powerful paradigm for algorithmic solutions to distributed problems; unlike the message-passing paradigm, mobile agents 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 software engineering and the 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 describe some of interesting problems and solution techniques developed in this investigations 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*, i.e., 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.

References

[1] S. Albers, M. Henzinger. “Exploring unknown environments”. *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 416–425, 1997.

[2] S. Alpern, S. Gal. *The Theory of Search Games and Rendezvous.* Kluwer, 2003.

[3] M. Asaka, S. Okazawa, A. Taguchi, S. Goto. “A method of tracing intruders by use of mobile agent”. *INET*, www.isoc.org, 1999.

[4] B. Awerbuch, M. Betke, M. Singh. "Piecemeal graph learning by a mobile robot". *Information and Computation* 152, 155–172, 1999.

[5] L. Barrière, P. Flocchini, P. Fraigniaud, N. Santoro. "Capture of an intruder by mobile agents". *Proc. 14th ACM-SIAM Symp. on Parallel Algorithms and Architectures (SPAA)*, 200-209, 2002.

[6] L. Barrière, P. Flocchini, P. Fraigniaud, N. Santoro. "Can we elect if we cannot compare?" In *Proc. 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 200–209, 2003.

[7] L. Barrière, P. Flocchini, P. Fraigniaud, N. Santoro. "Election and rendezvous in fully anonymous systems with sense of direction". In *Theory of Computer System*, to appear.

[8] L. Barrière, P. Fraigniaud, N. Santoro, D.M. Thilikos. "Searching is not jumping". *Proc. 29th Int. Workshop on Graph Theoretic Concepts in Computer Science (WG)*, LNCS 2880, 34-45, 2003.

[9] M. Bender, A. Fernandez, D. Ron, A. Sahai, S. Vadhan. "The power of a pebble: Exploring and mapping directed graphs". In *Proc. 30th ACM Symp. on Theory of Computing (STOC)*, 269–287, 1998.

[10] M. Bender, D. K. Slonim. "The power of team exploration: two robots can learn unlabeled directed graphs". In *Proc. 35th Symp. on Foundations of Computer Science (FOCS)*, 75–85, 1994.

[11] D. Bienstock. "Graph searching, path-width, tree-width and related problems". *DIMACS Series in Disc. Maths. and Theo. Comp. Sc.*, Vol. 5, 33–49, 1991.

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

[13] D. Bienstock, M. Langston. "Algorithmic implications of the graph minor theorem". *Hanbooks in OR & MS*, Vol. 7, Chapter 8, 481–502, Elsevier Science, 1995.

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

[15] L. Blin, P. Fraigniaud, N. Nisse, S. Vial. " Distributed chasing of network intruders by mobile agents". *Proc. of the 13th Int. Coll. on Structural Information and Communication Complexity (SIROCCO)*, 70–84, 2006.

[16] N. Borselius. "Mobile agent security". *Electronics and Communication Engineering Journal*, 14(5):211–218, October 2002.

[17] H. Buhrman, M. Franklin, J. Garay, J.-H. Hoepman, J. Tromp, and P. Vitányi. "Mutual search". *Journal of the ACM* 46(4), 517–536, 1999.

[18] R. Chang. "Single step graph search problem". *Information Processing Letters*, 40(2):107–111, 1991.

[19] D.M. Chess. "Security issues in mobile code systems". *Proc. Conf. on Mobile Agent Security*, LNCS 1419, 1–14,1998.

[20] C. Cooper, R. Klasing, T. Radzik "Searching for black-hole faults in a network using multiple agents". *Proc. 10th Int. Conf. on Principle of Distributed Systems* (OPODIS), 2006.

[21] J. Czyzowicz, D. Kowalski, E. Markou, A. Pelc. "Searching for a black hole in tree networks". *Proc. 8th Int. Conf. on Principle of Distributed Systems* (OPODIS), 35-45, 2004.

[22] J. Czyzowicz, D. Kowalski, E. Markou, A. Pelc. "Complexity of searching for a black hole". *Fundamenta Informaticae*, 71(2-3), 229-242, 2006. 35-45, 2004.

[23] S. Das, P. Flocchini, A. Nayak, N. Santoro. "Exploration and labelling of an unknown graph by multiple agents" *Proc. 12th Int. Coll. on Structural Information and Communication Complexity, (SIROCCO)*, 99-114, 2005.

[24] S. Das, P. Flocchini, A. Nayak, N. Santoro. "Effective elections for anonymous mobile agents". *Proc. 17th Int. Symp. on Algorithms and Computation (ISAAC)*, 2006.

[25] X. Deng, C. H. Papadimitriou, "Exploring an unknown graph". *J. of Graph Theory* 32(3), 265–297, 1999.

[26] N. Dendris, L. Kirousis, D. Thilikos. "Fugitive-search games on graphs and related parameters". *Theoretical Computer Science*, 172(1–2):233–254, 1997.

[27] A. Dessmark, P. Fraigniaud, A. Pelc. "Deterministic rendezvous in graphs". In *Proc. 11th European Symp. on Algorithms (ESA)*, 184–195, 2003.

[28] A. Dessmark, A. Pelc. "Optimal graph exploration without good maps". In *Proc. 10th European Symp. on Algorithms (ESA)*, 374–386, 2002.

[29] K. Diks, P. Fraigniaud, E. Kranakis, A. Pelc. "Tree exploration with little memory". *Journal of Algorithms*, 51:38–63, 2004.

[30] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro. "Mobile search for a black hole in an anonymous ring". *Algorithmica*, to appear.

[31] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro. "Multiple agents rendezvous in a ring in spite of a black hole". *Proc. 6th Int. Symp. on Principles of Distributed Systems* (*OPODIS*) 34-46, 2003.

[32] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro. "Searching for a black hole in arbitrary networks: optimal mobile agents protocols". *Distributed Computing*, to appear.

[33] S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, N. Santoro. "Optimal search for a black hole in common interconnection networks". *Networks*, 47 (2), p. 61-71, 2006.

[34] S. Dobrev, P. Flocchini, N. Santoro. "Improved bounds for optimal black hole search in a network with a map". *Proc. 10th Int. Coll. on Structural Information and Communication Complexity* (), 111-122, 2004.

[35] S. Dobrev, P. Flocchini, R. Kralovic, N. Santoro. "Exploring a dangerous unknown graph using tokens". *Proc. 5th IFIP Int. Conf. on Theoretical Computer Science* (*TCS*), 131-150, 2006.

[36] S. Dobrev, P. Flocchini, N. Santoro. "Cycling through a dangerous network: a simple efficient strategy for black hole search". *Int. Conf. on Distributed computing Systems* (*ICDCS*), 2006.

[37] S. Dobrev, R. Kralovic, N. Santoro, W. Shi. "Black hole search in asynchronous rings using tokens". *Proc. 6th Conf. on Algorithms and Complexity* (*CIAC*), 139-150, 2006.

[38] G. Dudek, M. Jenkin, E. Milios, D. Wilkes. "Robotic exploration as graph construction". *Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[39] J. Ellis, H. Sudborough, J. Turner. "The vertex separation and search number of a graph". *Information and Computation*, 113(1):50–79, 1994.

[40] J. Fernandez and J. Gonzalez. "Hierarchical graph search for mobile robot path planning". In *Int. Conf. on Robotics and Automation* (*ICRA*), IEEE, 656–661, 1998.

[41] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk. "Multiple mobile agent rendezvous in a ring". Proc. *6th Latin American Theoretical Informatics Symp.* (*LATIN*), 599–608, 2004.

[42] P. Flocchini, M.J. Huang, F.L. Luccio. "Contiguous search in the hypercube for capturing an intruder" *Proc. 18th IEEE Int. Parallel and Distributed Processing Symp.* (*IPDPS*), 2005.

[43] P. Flocchini, M.J. Huang, F.L. Luccio. "Decontamination of chordal rings and tori". *Proc. 8th Workshop on Advances in Parallel and Distributed Computational Models (APDCM)*, 2006.

[44] P. Flocchini, B. Mans, N. Santoro. "Sense of direction in distributed computing". *Theoretical Computer Science*, vol. 291, 29-53, 2003.

[45] P. Flocchini, A. Nayak, A. Shulz. " Cleaning an arbitrary regular network with mobile agents" *Proc. 2nd Int. Conf. on Distributed Computing & Internet Technology (ICDCIT)*, 132-142, 2005.

[46] F. Fomin and P. Golovach. "Graph searching and interval completion". *SIAM Journal on Discrete Mathematics* 13(4), 454–464, 2000.

[47] N. Foukia,J. G. Hulaas, J. Harms. "Intrusion Detection with Mobile Agents". *Proc. 11th Annual Conference of the Internet Society* (*INET*), 2001.

[48] P. Fraigniaud, L. Gasieniec, D. Kowalski, A. Pelc. "Collective tree exploration". *Networks*, to appear.

[49] P. Fraigniaud, D. Ilcinkas, "Digraph exploration with little memory". *Proc. 21st Symp. on Theoretical Aspects of Computer Science* (*STACS*), 246–257, 2004.

[50] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, D. Peleg. "Graph exploration by a finite automaton". *Theoretical Computer Science*, to appear.

[51] P. Fraigniaud, N. Nisse. "Monotony properties of connected visible graph searching". *Proc. 32nd Int. Workshop on Graph-Theoretic Concepts in Computer Science* (*WG*) 22-24, 2006.

[52] P. Fraigniaud, N. Nisse. "Connected Treewidth and Connected Graph Searching". *Proc. 7th Latin American Theoretical Informatics Symposium* (LATIN) 479-490, 2006.

[53] M.S. Greenberg, J.C. Byington, D. G. Harper. "Mobile agents and security. *IEEE Commun. Mag.* 36(7), 76 – 85, 1998.

[54] S. Hansen and M. Eldredge. "Intruder isolation and monitoring". In *Proc. 1st Security Workshop*, USENIX, 63–64, 1988.

[55] F. Hohl. "Time limited blackbox security: Protecting mobile agents from malicious hosts". In *Proc. Conf. on Mobile Agent Security*, LNCS 1419, pages 92–113, 1998.

[56] W. Jansen. "Countermeasures for mobile agent security". *Computer Communications*, Nov. 2000.

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

[58] L. Kirousis and C. Papadimitriou. "Interval graphs and searching". *Discrete Mathematics* 55, 181–184, 1985.

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

[60] R. Klasing, E. Markou, T. Radzik, F. Sarracco. "Approximation bounds for black hole search problems". *Proc. 9th Int. Conf. on Principle of Distributed Systems* (*OPODIS*), 2005.

[61] R. Klasing, E. Markou, T. Radzik, F. Sarracco. "Hardness and approximation results for black hole search in arbitrary graphs". *Proc. 12th Int. Coll. on Structural Information and Communication Complexity* (*SIROCCO*), 200-215, 2005.

[62] E. Korach, D. Rotem and N. Santoro. "Distributed algorithms for finding centers and medians in networks". *ACM Trans. on Programming Languages and Systems*, 6(3): 380–401, 1984.

[63] E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk. "Mobile agent rendezvous in a ring". *Int. Conf. on Distibuted Computing Systems (ICDCS)*, 592–599, 2003.

[64] E. Kranakis, D. Krizanc, S. Rajsbaum. "Mobile agent rendezvous". *Proc. 13th Int. Coll. on Structural Information and Communication Complexity* (*SIROCCO*), 1–9, 2006.

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

[66] T. Lengauer. "Black-white pebbles and graph separation". *Acta Informatica*, 16(4):465–475, 1981.

[67] F. Luccio, L. Pagli, N. Santoro. "Network decontamination with local immunization". *Proc. 8th Workshop on Advances in Parallel and Distributed Computational Models (APDCM)*, 2006.

[68] F. Makedon and H. Sudborough. "Minimizing width in linear layout". *Proc. 10th Int. Coll. on Automata, Languages, and Programming* (ICALP '83), LNCS 154, Springer-Verlag, 478–490, 1983.

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

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

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

[72] P. Panaite, A. Pelc, "Exploring unknown undirected graphs". *Journal of Algorithms*, 33 281-295, 1999.

[73] P. Panaite, A. Pelc. "Impact of topographic information on graph exploration efficiency". *Networks*, 36, 96–103, 2000.

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

[75] T. Parson. "The search number of a connected graph". *Proc. 9th Southeastern Conf. on Combinatorics, Graph Theory and Computing*, Utilitas Mathematica, 549–554, 1978.

[76] S. M. Ruiz. "A result on prime numbers." *Math. Gaz.* 81, 269, 1997.

[77] T. Sander, C. F. Tschudin. "Protecting mobile agents against malicious hosts". *Proc. of Conf on Mobile Agent Security*, LNCS 1419, pages 44–60, 1998.

[78] P. Seymour and R. Thomas "Graph searching, and a min-max theorem for treewidth". *Jour. Combin. Theory, Ser. B*, 22-33, 1993.

[79] J. Smith. "Minimal trees of given search number". *Discrete Mathematics* 66, 191–202, 1987.

[80] K. Schelderup and J. Ones. "Mobile agent security - Issues and directions". *Proc. 6th Int. Conf. on Intelligence and Services in Networks*, LNCS 1597, 155–167, 1999.

[81] E. H. Spafford, D. Zamboni. "Intrusion detection using autonomous agents". *Computer Networks*, 34(4):547–570, 2000.

[82] Y. Stamatiou and D. Thilikos. "Monotonicity and inert fugitive search games". In 6th *Twente Workshop on Graphs and Comb. Opt.*, Elsevier, 1999.

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

[84] 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". *Information Processing Letters*, 66(1):49–52, 1998.

[85] A. Takahashi, S. Ueno, and Y. Kajitani. "Mixed searching and proper-path-width". *Theoretical Computer Science*, 137(2):253–268, 1995.

[86] D. Thilikos. "Algorithms and obstructions for linear-width and related search parameters". *Discrete Applied Mathematics* 105, 239–271, 2000.

[87] J. Vitek, G. Castagna. "Mobile computations and hostile hosts". In D. Tsichritzis, editor, *Mobile Objects*, pages 241–261. University of Geneva, 1999.

[88] B. von Stengel and R. Werchner. "Complexity of searching an immobile hider in a graph". *Discrete Applied Mathematics*, 78, 235 - 249,1997.

[89] M. Yamamoto, K. Takahashi, M. Hagiya, and S.-Y. Nishizaki. "Formalization of graph search algorithms and its applications". *Proc. 11th Int. Conf. on Theorem Proving in Higher Order Logics*, 479 - 496, 1998.