Searching for a Black Hole in Arbitrary Networks: Optimal Mobile Agent Protocols*

Stefan Dobrev † Paola Flocchini† Giuseppe Prencipe‡
Nicola Santoro§

Abstract

Consider a networked environment, supporting mobile agents, where there is a black hole: a harmful host that disposes of visiting agents upon their arrival, leaving no observable trace of such a destruction. The black hole search problem is the one of assembling a team of asynchronous mobile agents, executing the same protocol and communicating by means of whiteboards, to successfully identify the location of the black hole; we are concerned with solutions that are generic (i.e., topology-independent). We establish tight bounds on the size of the team (i.e., the number of agents), and the cost (i.e., the number of moves) of a size-optimal solution protocol. These bounds depend on the a priori knowledge the agents have about the network, and on the consistency of the local labellings. In particular, we prove that: with topological ignorance $\Delta + 1$ agents are needed and suffice, and the cost is $\Theta(n^2)$, where Δ is the maximal degree of a node and n is the number of nodes in the network; with topological ignorance but in presence of sense of direction only *two* agents suffice and the cost is $\Theta(n^2)$; and with complete topological knowledge only *two* agents suffice and the cost is $\Theta(n \log n)$. All the upper-bound proofs are constructive.

Keywords: Mobile Agents, Malicious Host, Distributed Search, Black Hole, Topological Knowledge.

---

*A preliminary version of this paper appeared in the Proceedings of the 21^{st} ACM Symposium on Principles of Distributed Computing [21].

†School of Information Technology and Engineering, University of Ottawa, email: {flocchin,sdobrev}@site.uottawa.ca

‡Dipartimento di Informatica, Università di Pisa, email: prencipe@di.unipi.it

§School of Computer Science, Carleton University, email: santoro@scs.carleton.ca

1 Introduction

1.1 The problem

In networked environments that make use of mobile agents, security is a pressing concern of difficult resolution, and even the most basic issues must still be effectively addressed (e.g., see [13, 30, 34, 40, 46]). The causes of this situation are not lack of interest and effort; witness, for example, the large effort to determine how to protect a network site (a *host*) from malicious agents, as well as to protect agents from host attacks. Rather it is due to the difficulties found when developing solutions; the nature of these obstacles is varied, most are technological, some computational.

In this paper we consider the issue of *host attacks*; that is, the presence in a site of processes that harm incoming agents (e.g., see [23, 32, 33, 39, 45, 48]). 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.

In this paper, 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* [20]. The task is to unambiguously determine and report the location of the black hole (Black Hole Search problem). 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.

The setting in which we study the Black Hole Search problem is that of *cooperating asynchronous agents* on an edge-labelled graph: the agents have computing capabilities and storage, can move from node to neighboring node, obey the same set of behavioral rules (the "protocol"), and all their actions (e.g., computation, movement, etc) take a finite but unpredictable amount of time. At each node, the incident edges are distinctly labelled, and there is a limited amount of storage, called *whiteboard*. Agents communicate by reading from and writing on the whiteboards; access to a whiteboard is done in mutual exclusion.

Black Hole Search is a non trivial problem, and its difficulty is aggravated by the simultaneous presence of asynchrony of the agents and absence of any trace of destruction (outside the black hole). We have investigated the problem when the network is an anonymous *ring* (i.e., a loop network of identical nodes sites), characterizing the limits and presenting optimal solutions [20]. These results are however topology-dependent, and in the solution protocols the agents exploit the unique properties of the ring to locate the black hole.

1.2 Our Results

In this paper we are interested in the black hole search problem in general graphs, and we are concerned with *generic* (i.e. topology-independent) solutions. We ask computational questions regarding the *size* of these solutions (i.e., the number of agents in the team), and the conditions for their existence.

Some answers follow from the asynchrony of the agents. For example, as we show, the problem is unsolvable if the graph G representing the network topology is not 2-connected; therefore, we only consider 2-connected graphs. As we prove later, also due to asynchrony, the problem is unsolvable if the number of nodes of G is not known; therefore, we assume n is known. Some answers are immediate. For example, since a single agent is incapable of performing the task, the size of a team is at least two. How realistic is this bound? How many agents suffice? We are also interested in the *cost* of the minimal solutions (i.e., the number of moves performed by the agents executing a size-optimal solution protocol).

In this paper we provide specific answers to these computational questions. We show that the answers vary depending on the a priori knowledge the agents have about the network, and on the consistency of the local labellings.

We consider first the situation of *topological ignorance*; that is when the agents have no a priori knowledge of the topological structure of G. We show that 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. We further prove that, in any *minimal* generic solution, the agents must perform $\Omega(n^2)$ moves in the worst case. Both these bounds are *tight*. In fact we present a protocol that correctly locates the black hole in $O(n^2)$ moves using $\Delta + 1$ agents that know Δ and n.

We then consider the case of topological ignorance, first in systems where there is *sense of direction* (SD); informally, sense of direction is a globally consistent labelling 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. We show that, in this case, two agents suffice to locate the black hole, regardless of the (unknown) topological structure of G. The proof is constructive, and the proposed algorithm has a $O(n^2)$ cost. We further show that this cost is optimal; in fact, we show the existence of 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.

Finally, we 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). We show that, also in this case, two agents suffice. We then constructively prove that the cost of a minimal protocol can be reduced in this setting to $\Theta(n \log n)$, showing a matching lower bound as well.

1.3 Related Work

This study is part of a larger investigation on the algorithmic issues arising in networked environments that support autonomous mobile agents. At an abstract level, these environments can be described as a team of autonomous mobile entities located in a graph G. Depending on the context, the entities are sometimes called *agents* or *robots*. The research concern is on determining what tasks can be performed by such entities, under what conditions, and at what cost.

In terms of topics, the investigations most closely related to the one of this paper are the studies on *collaborative graph-exploration*. In these investigations, a team of autonomous agents must explore (i.e., traverse every link and visit every node) the graph in which they reside. None of these investigations, however, considers the presence of harmful hosts.

The investigations on exploration with *multiple* agents are a generalization of the extensive earlier work on exploration by a *single* agent (e.g., [1, 5, 9, 15, 17, 19, 28, 38, 41, 47]); they focus on different aspects of the problem, each result giving us some knowledge on the impact that factors such as memory size, computation capabilities, and *a priori* knowledge have on the solvability and complexity of the problem. For example, the earlier investigations, by Blum and Kozen [12] and Kozen [35], considered a team of agents that were *finite-automata*. It was later shown by Rollik that, with those type of agents, there is no *generic* protocol regardless of the (finite) number of agents employed [43], proving an earlier conjecture by Rabin [42]. More recently the investigations have focused on collaborative exploration by *Turing machines*; one of the earlier contributions was by Bender and Slonim [10]. Solution algorithms for collective exploration were given by Frederickson et al. for arbitrary graphs [29], by Averbakh and Berman for weighted trees [4], and more recently by Fraigniaud et al. for trees [27].

The impact of *sense of direction* on the exploratory power of the agents was first examined in the work of Blum and Kozen on the power of a *compass* [12]. We mention only in passing the extensive investigations on exploration and navigation in terrains other than graphs (e.g., [11, 14, 22, 44]), where orientation and sense of direction play an important role.

It should be pointed out that, in general, the work on graph exploration assumes *synchronous* agents, and in general employs limited forms of communications between agents; whiteboards are used only in [27]. Furthermore, none of these investigations considers the presence of harmful nodes.

Other investigations on cooperative solutions of problems by mobile agents include the studies on *rendezvous* (e.g., see [2, 3, 16, 36, 49]), *election* (e.g., see [8, 6]), and *decontamination* [7].

2 Definitions and Basic Properties

2.1 Framework and Problem

Let $G = (V, E)$ be a simple 2-connected graph; let $n = |V|$ be the size of G, $E(x)$ be the links incident on $x \in V$, $d(x) = |E(x)|$ denote the degree of x, and Δ denote the maximum degree in G. If $\{x, y\} \in E$ then x and y are said to be neighbors. The nodes of G can be *anonymous* (i.e., without unique names). At each node x there is a distinct label (called port number) associated to each of its incident links (or ports); let $\lambda_x(\{x, z\})$ denote the label associated at x to the link $\{x, z\} \in E(x)$, and λ_x denote the overall injective mapping at x. The set $\lambda = \{\lambda_x | x \in V\}$ of those mappings is called a *labelling* and we shall denote by (G, λ) the resulting edge-labeled graph. Let $P[x]$ denote the set of all paths with x as a starting point, $P[x, y]$ denote the set of paths starting from node x and ending in node y, and let Λ be the extension of the labelling function λ from edges to paths.

Informally, (G, λ) has *sense of direction* if it is possible, for any two paths $\pi_1, \pi_2 \in P[x]$, to determine from the sequence of labels $\Lambda_x(\pi_1)$ and $\Lambda_x(\pi_2)$ whether or not π_1 and π_2 lead to the same node. The formal definition as well as examples of sense of direction is deferred to Section 4.

Operating in (G, λ) is a set of autonomous mobile agents. The agents can move from node to neighboring node in G, have computing capabilities and computational storage, and obey the same set of behavioral rules (the "protocol"). The agents are *asynchronous* in the sense that every action they perform (computing, moving, etc) takes a finite but otherwise unpredictable amount of time. Initially, all agents are at the same node h, called *home base*.

Each node has available a limited amount of storage, called *whiteboard*. Agents communicate by reading from and writing on the whiteboards; access to a whiteboard is gained fairly in mutual exclusion. When an agent is at a node, we assume it can "see" the port numbers associated to the incident links, as well as the information on the whiteboard. Moreover, when an agent moves on a link $\{x, y\}$, it can "see" the labels $\lambda_x(\{x, y\})$ and $\lambda_y(\{y, x\})$ at the two sides of the link (in other words, when arriving to y, the agent can recognize that label $\lambda_y(\{y,x\})$ leads back to x).

In the following we assume the agents have distinct Ids; notice however that, since the whiteboards are accessed in mutual exclusion and the agents start from the same homebase, distinct Ids could be easily constructed if the agents are initially anonymous.

A *black hole* (shortly BH) is a node where resides a stationary process that destroys any agent arriving at that node; no observable trace of such a destruction is evident to the other agents. The location of the black hole is unknown to the agents. The BLACK HOLE SEARCH (shortly BHS) problem is to find the location of the black hole. More precisely, BHS is solved if at least one agent survives, and all surviving agents know, for each edge, whether it leads to the BH [1].

A solution protocol is *generic* if it solves BHS regardless of G; in this paper we only consider generic protocols. The main measure of complexity of a solution protocol $\mathcal{P}$ is the *size*, that is the number of agents used by $\mathcal{P}$. We also consider the total number of moves performed by the agents, and call it the *cost* of $\mathcal{P}$.

We study generic solutions and their complexity depending on the type of topological information the agents might have a priori available; we always assume they all know n. If no additional topological information is available, the agents operate with *topological ignorance*. If the system (G, λ) has sense of direction that is known to the agents, we say the agents operate with *sense of direction*. Finally, the agents have *complete topological knowledge* of (G, λ) if the following information is available to all agents: (1) Knowledge of the labeled graph (G, λ); (2) Correspondence between port labels and the link labels of (G, λ); (3) Location of the home base in (G, λ).

2.2 Basic Tool and Limitations

2.2.1 Cautious Walk

At any moment of the execution of a protocol, the ports are classified as *unexplored* – no agent has left from ar arrive to this port, *explored* – an agent has arrived via this port, or *dangerous* – an agent has left via this port, but no agent has arrived via it. Obviously, an explored port does not lead to a black hole; on the other hand, both unexplored and dangerous ports might lead to it. To minimize the number of casualties (i.e., agents entering the black hole), we do not allow any agent to leave through a dangerous port. To prevent the execution from stalling, in all our algorithms we require any dangerous port not leading to the black hole, to be made explored as soon as possible.

This is accomplished as follows: 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). We call this technique *Cautious Walk*.

Similarly to the classification adopted for the ports, we classify nodes as follows: at the beginning, all nodes except for the home base h are *unexplored*; the first time a node is visited by an agent, it becomes *explored*. Note that, by definition, the BH never becomes explored.

[1] In the particular case of $n - 4 < \Delta < n$, as explained later, we will use a relaxed definition.

2.2.2 Limits

The nature of the problem imposes several basic limits to any solution. As the first move of the single agent might lead to the BH, it follows that at least two agents are needed to locate the BH.

The following lemma is directly implied by the definition of BHS and from asynchrony of the system:

Lemma 2.1. *If there are two or more unexplored nodes, the* BHS *problem is not solved.*

Proof. Let u and v be two such nodes. Because of the asynchrony, for any finite amount of time, the scenario where u is the BH and every agent travelling towards v is very slow is indistinguishable from the scenario where v is the BH and any agent travelling towards u is very slow. Hence no protocol terminating with both u and v unexplored can correctly report the location of the BH. □

By using a similar argument, it follows that it is not possible to verify whether or not there is a black hole in the system – the agents cannot distinguish in finite time whether the last unexplored node is the BH or all agents moving towards it are simply too slow.

Lemma 2.2. *It is not possible to verify whether there is a* BH *in the system.*

The following two corollaries of Lemma 2.1 complete the list of basic limitations:

Corollary 2.3. *If G has a cut vertex different from the* home base*, then it is impossible to determine the location of the* BH*.*

Proof. If the BH is located in the cut vertex, any algorithm terminating in finite time leaves at least two nodes unexplored. Contradiction with Lemma 2.1.

□

Corollary 2.4. *It is impossible to determine the location of the black hole if the size of G is not known.*

Proof. Consider two scenarios: A graph G with a black hole at node v and a graph G' with node v replaced by two nodes v_1 and v_2 connected to each other, and dividing among themselves the edges leading to v in G. With all edges leading to v_1 and v_2 being very slow, the algorithm cannot distinguish between G and G' and would terminate in G' with both v_1 and v_2 unexplored; a contradiction with Lemma 2.1.

□

As a consequence, we assume that there are at least two agents working in a 2-connected network; furthermore, the existence of the black hole and the size of G are common knowledge to the agents.

3 Searching with Topological Ignorance

3.1 Lower Bounds

3.1.1 The Adversary

The lower bounds we are going to prove are for algorithms that know n and an upper bound Δ on the maximum degree of G, and work on all 2-connected graphs of maximal degree at most Δ.

We follow an approach common in lower bound proofs by viewing the execution as a game between an algorithm and an adversary. The goal of the algorithm is to locate the black hole and to terminate. The adversary tries to either force the algorithm to behave incorrectly, or make it costly. The adversary has the power (1) to choose the graph (with port labels), (2) to place the black hole and (3) to set link delays. The first two powers are due to the fact that the algorithm does not know the network nor the location of the black hole. The fact that agents are asynchronous gives the adversary the third power. In particular, we say that the adversary *blocks* a link l when it sets very high delays for that link; as a consequence, we say that an agent traversing l is *blocked*; viceversa, to block an agent traversing a link l, the adversary blocks l.

Notice the adversary's power to 'direct' the agents leaving via unexplored ports: When the algorithm specifies that an agent leaves a node via an unexplored port, it chooses the port based on the port labels. Since the adversary can permute the port labels of the unexplored ports, this means that the adversary can choose via which unexplored port the agent leaves.

Because of asynchrony, we can limit ourselves to reactive algorithms: If an algorithm uses timeouts, the adversary can make all timeouts expire without allowing any agent to arrive to its destination. This means that an agent at a node must either wait for the arrival of an agent, or depart via one of the incident ports.

We further simplify matters by turning a dangerous port p into explored as soon as the agent that departed via p arrives to its destination (provided it is not the black hole). This essentially gives any algorithm the power of cautious walk, without requiring that an agent must return to mark an active port as explored. Clearly, this only strengthens the algorithm and makes the lower bound result stronger. Since every algorithm can be modified to use cautious walk[2], we can limit ourselves to algorithms that avoid sending an agent via a dangerous port.

The above discussion narrows the possible actions the algorithm can specify for available (not in transit and not waiting) agents to: (1) departure via an unexplored port, (2) departure via an explored port, and (3) waiting until another agent arrives, or a port status changes.

The adversary applies its power to set delays by being able to specify at any moment of the execution which agents in transit arrive to their destination, and which continue to travel.

Since an explored link does not lead to a black hole, whenever there are some agents in transit over such links, the adversary must eventually allow all these agents to arrive to their destinations. Call a configuration *stable* if there are no agents in transit over explored links. Then the adversary will use the following rule:

- *If there are agents travelling via explored links, the adversary blocks the agents that left through unexplored ports, until a stable configuration is reached.*

Applying this rule we are allowed to focus on the behavior of the adversary in stable configurations only.

At any moment of the execution the maximal information that the algorithm can have about the underlying graph G can be described by a graph G_{ex} induced by the explored nodes, together with the information about their degree and the labels of the ports incident in G. We denote this graph, together with the additional information, by G_{ex}^{+}.

The power of the adversary to choose the graph and the port labelling comes into play in stable configurations, when the adversary allows some agents to move and to arrive to their destination. At that moment, the adversary chooses a *witness graph* $G_{wit} = (V, E)$ (together with its port labelling): a graph with a Bh *consistent* with the previous execution of the algorithm. Consistency means that G_{ex} is an induced subgraph of G_{wit} and for all nodes in G_{ex}

[2]by increasing the time and number of moves at most three times.

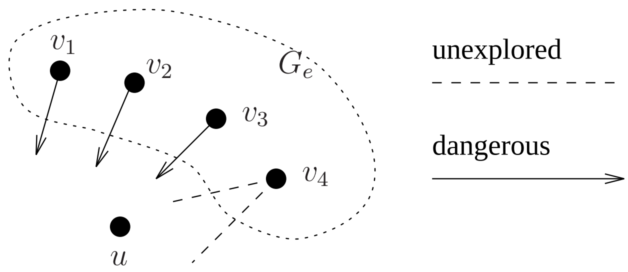


Figure 1: Example of insufficiency of conditions from Observation 3.2.

the information (degree and port labels) contained in G_{ex}^+ matches these values in G_{wit}. Also note that G_{wit} must be 2-connected, of order n and should have maximal degree at most Δ. In the rest of this section we restrict ourselves only to such graphs, without explicitly stating these properties. Observe that there are usually many possible witness graphs consistent with the current knowledge G_{ex}^+, and the adversary's choices of the witness graph can vary during the execution.

The following observation and fact capture the limits of the adversary's powers and give additional intuition behind our choice of graphs in the lower bound proofs of this section.

Since the adversary must eventually unblock links not leading to the BH, we get the following:

Observation 3.1. *If, for a stable configuration, there is no witness graph such that all dangerous ports lead to the* BH*, then the adversary must allow at least one agent to arrive to its destination.*

The following fact describes in detail some conditions under which such a witness graph does not exist.

Observation 3.2. *If any of the following conditions hold in a stable configuration, then the adversary must allow at least one agent to arrive to its destination.*

1. *There is a node u with two incident dangerous ports.*

2. *There are at least $\Delta + 1$ dangerous ports.*

3. *There are Δ dangerous ports, at most one* open node *(explored node with an unexplored port) and at most $n - 2$ explored nodes.*

4. *There are some dangerous ports, no open node and at most $n - 2$ explored nodes.*

Proof. For each case we prove that there is no witness graph G_{wit} and location of the BH in G_{wit} such that all dangerous ports lead to BH.

1. Since G_{wit} does not contain multiple edges, there can be at most one link from any node to the BH. Therefore, not both dangerous ports of u can lead to the BH; hence, at least one dangerous port does not lead to the BH.

2. Since G_{wit} has degree at most Δ, if there are $\Delta + 1$ or more dangerous port at least one of them does not lead to the BH.

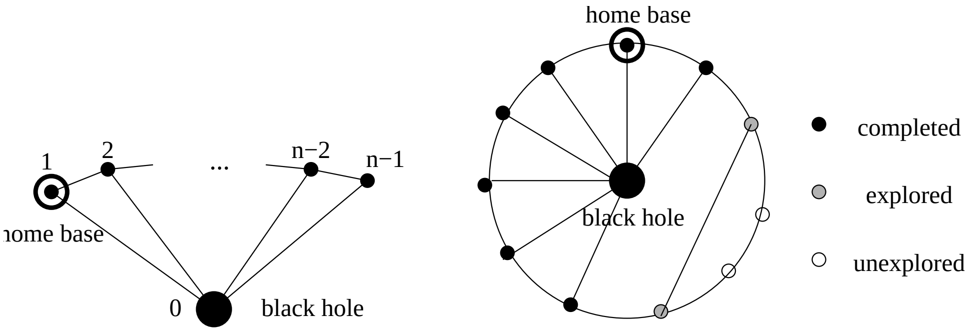


Figure 2: Theorem 3.4: The lower bound graphs for $\Delta \geq n-4$ (left) and $\Delta \leq n-4$ (right).

3. Suppose, by contradiction, that all dangerous ports lead to the BH. By definition of dangerous port, and since there are Δ dangerous ports, it follows that there is no link connecting the BH and the unexplored part of G_{wit}. Therefore, the unexplored part of G_{wit} must be directly connected to the explored part. By hypothesis, however, there is at most one open node that can connect the unexplored and the explored part of G_{wit}; this contradicts the 2-connectivity of G_{wit}.

4. Using an argument similar to the one adopted in Point 3., it follows that the unexplored part can be connected only to the BH, which again contradicts the 2-connectivity of G_{wit}.

□

If any of the conditions stated in Observation 3.2 holds, the adversary can not find a witness graph with all dangerous ports leading to the BH. However, avoiding those conditions is not sufficient:

Observation 3.3. *Even if none of the conditions of Observation 3.2 holds in a stable configuration, it might be impossible for the adversary to connect all dangerous ports to the* BH.

Proof. Consider the following G^+_{ex}: a) G_{ex} contains $n-2$ nodes; b) $\Delta = 3$; c) there are three explored nodes v_1, v_2, v_3 with one dangerous and zero unexplored ports each; and d) there is one open node v_4 with two unexplored links (see Figure 1). None of the conditions of Observation 3.2 is satisfied; however, it is still not possible to have all dangerous ports leading to the BH: in fact, in such a case, v_4 has also to be connected to the BH, violating $\Delta = 3$. □

3.1.2 Lower bound on the number of agents

The following theorem gives us the first lower bound, binding the size of the team to the maximal degree of the network:

Theorem 3.4. *There is an n node graph G with maximum degree $3 \leq \Delta \leq n-4$ such that, without topological information, any algorithm for locating the black hole in arbitrary networks needs at least $\Delta + 1$ agents in G. In addition, if $n-4 < \Delta < n$ then any such algorithm needs at least Δ agents.*

Proof. We first prove the second part for $\Delta = n-1$. Consider the graph shown in Figure 2(a), with the BH at node 0, and the *home base* at node 1. The first agent departing the *home base* is sent via the link leading to 0 and it is blocked on this link. Therefore, there must be an agent moving to node 2. Again, the first agent departing node 2 via unexplored link is sent to 0; the same is applied for nodes $3, 4, \ldots, \Delta - 1$, resulting in $\Delta - 1$ agents being blocked before the algorithm can reach node $n-1$ and terminate. If $\Delta = n-2$, the adversary chooses the same graph, except that node 2 is not connected to 0. If $\Delta = n-3$, also 3 is not connected to 0. It is easy to see that the previous approach can be applied also in these cases.

The case $\Delta \leq n-4$ is played on a spoked ring graph G' consisting of an $n-1$ node ring with each node connected to the central node – the BH. The *home base* is one of the ring nodes. The adversary applies the following rule: The first agent leaving a node via an unexplored link is sent via the link leading to the BH. We call a node v *completed* if an agent has left v towards the BH. At any moment, the subgraph induced by the completed nodes is a path containing the *home base*. Moreover, there are at most two explored but non-completed nodes - the non-BH neighbours of the completed path.

Consider now the situation during the execution of an algorithm A on G' where there are exactly Δ completed nodes. The graph G of maximum degree Δ on which A needs at least $\Delta+1$ agents is obtained from G' by removing all edges leading to the BH from non-complete vertices and adding an edge connecting the two neighbours of the completed path – see Figure 2(b). While the number of completed nodes is at most Δ, A sees only a path of degree-3 nodes and therefore behaves the same on G and G'. Since $\Delta \leq n-4$, at the moment when there are Δ completed nodes there are still at least two unexplored nodes and at least one more agent is needed. □

The case $\Delta = 2$ corresponds to the ring and has been studied in [20] (two agents are necessary and sufficient).

3.1.3 Lower bound on the cost using optimal number of agents

In this subsection we establish a lower bound on the cost of BLACK HOLE SEARCH using a team of optimal (i.e., $\Delta + 1$) size. Let $\mathcal{G}(\Delta, n)$ denote the set of 2-connected graphs of n nodes of maximal degree at most $3 \leq \Delta \leq n-4$. We show that for every agent-optimal (using $\Delta + 1$ agents) algorithm $\mathcal{A}$ that correctly locates the BH in all graphs in $\mathcal{G}(\Delta, n)$, there exists a $G \in \mathcal{G}(\Delta, n)$ such that $\mathcal{A}$ spends $\Omega(n^2)$ moves on G in the worst case.

Let us call *basic cell* the graph obtained from a mesh $(\Delta+2) \times 2$ by collapsing the outermost pairs of nodes. The game is played on an $\lfloor n/(2\Delta+2) \rfloor$-node ring having each node replaced by a basic cell (see Figure 3). If n is not divisible by $2\Delta + 2$, an appropriate number of nodes is connected to the basic cell opposite to the starting node to yield an n node graph G.

Note that G is of degree 3, not Δ. Still, $\mathcal{A}$ must work correctly on it, and we will show that it incurs $\Omega(n^2)$ cost. Interestingly, this means that the mere possibility of the graph having a node of degree Δ pushes the lower bound up to $\Omega(n^2)$.

Overview. At any moment of an execution the explored part of the network can be divided into three parts (see Figure 4):

$\mathcal{E}$: The middle, fully explored part, consisting of fully explored basic cells.

$\mathcal{O}$: Open border, containing a single partially explored basic cell.

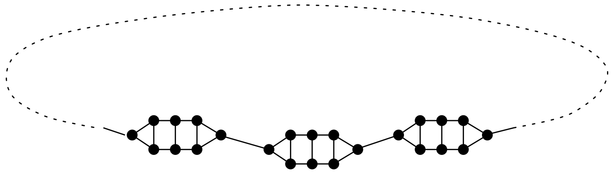


Figure 3: The graph G for $\Delta = 3$.

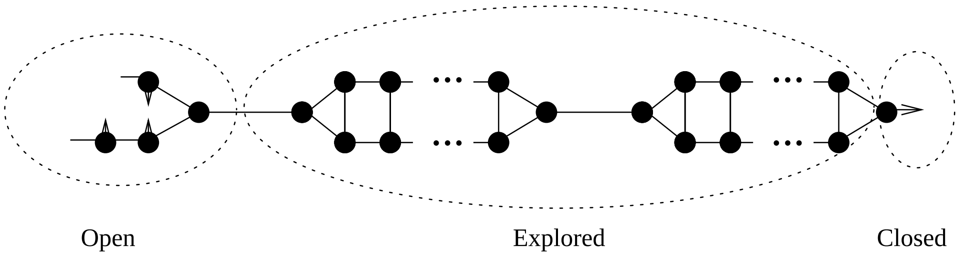


Figure 4: An example of explored part of the network. The arrows correspond to agents in transit.

$\mathcal{C}$: Closed border, consisting of a single blocked *bridge link* leading to the next (yet unexplored) basic cell.

The idea is to force the agents to repeatedly migrate between the opposite ends of the explored subgraph, crossing the ever increasing middle part. This migration is forced by the adversary "closing" the open border part by blocking the horizontal link to the next basic cell, and simultaneously "opening" the closed border part. Eventually, all unexplored ports in the newly closed part will be explored and there will be no exploration work left there. If the agents do not move to the open part (i.e. they start waiting for the blocked link to become unblocked), the adversary chooses a witness graph in which all dangerous ports lead to the black hole. Once all agents (except the agent blocked at the horizontal link) have migrated, the closed part is opened, the open part is closed and the process is repeated until (almost) the whole network is explored.

Detailed description. To reduce the number of cases we must consider, we give all algorithms solving the Bhs problem the following additional power: Whenever an agent arrives to a node with a dangerous port, all remaining unexplored ports are *revealed* (explored) at no cost. Clearly, this modification only further strengthens our lower bound.

Between two stable configurations, an agent either remains blocked, or moves over explored links until it departs via an unexplored port (becoming blocked), or starts waiting. These choices are called "meta-actions" by the algorithm. We describe the adversary's actions as responses to such meta-actions of the algorithm. In particular, we specify how the adversary acts (1) at the beginning of the execution, until a closed and open border parts are formed; (2) in an open border part, until it is closed; and (3) in a closed border part, until it is opened. The actions of the adversary are described as reaction to arrival of an agent to the particular part. Note that, although there could be several available agents (e.g., at the beginning of the execution), the adversary can handle them sequentially by allowing only one of them to move, while blocking

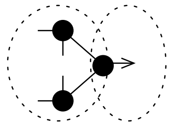


Figure 5: The initial configuration.

all others. The adversary reacts to awakening of an agent in the same way as if the agent has just arrived.

Initialization. The adversary chooses a graph with the *home base* node at the right border of a basic cell. The initialization is achieved by sending the first agent to the right and blocking it on that link. In addition, the two left links are revealed to the algorithm without any cost (this is done to simplify presentation, and it clearly only strengthens our lower bound). This terminates initialization, with the right border part being closed and the left border part being open (see Figure 5).

Open part. Whenever an agent arrives to an open part (either from the closed segment, or being awakened) and departs from a node v via an unexplored port, the adversary directs it to the vertical link incident to v and blocks it. This ensures that, in general, an open part looks like in Figure 4; the only freedom the algorithm has is to choose whether to explore from the top line or from the bottom one.

When the number of agents that are creating dangerous links in the open part reaches Δ, the adversary switches the open and closed parts as follows.

The open part is closed by blocking the link leading to the next basic cell. Moreover, all agents that have been blocked in this part are unblocked, and all remaining unexplored links in this basic cell are revealed at no cost to the algorithm. The resulting fully explored basic cell is then added to the middle (explored) part.

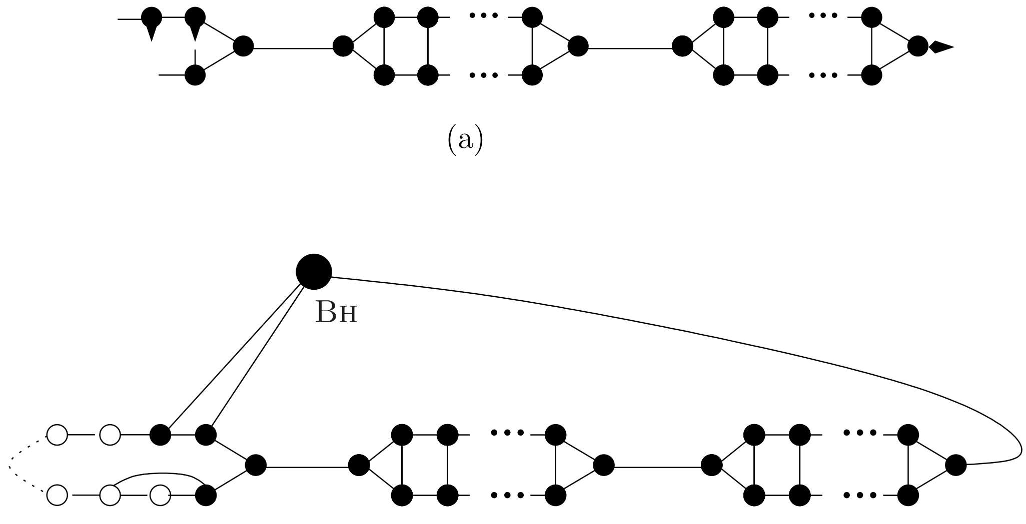
The closed part is opened by unblocking the bridge link and revealing the two incident links of the neighboring unexplored basic cell.

Closed part. The adversary keeps the bridge link blocked. Since there are no unexplored ports in this part, all agents will either start waiting or leave for the open part.

Lemma 3.5. *If an open part does not switch to closed, and the number of explored nodes is at most $n - 3$, then the algorithm does not locate the black hole.*

Proof. We show that, in such a case, the adversary can construct a witness graph in which all dangerous ports lead to the Bh.

First note that an open part has at most two open nodes – the last explored node in each horizontal line. Since, by hypothesis, the open part does not switch to closed, then there are at most $\Delta - 1$ dangerous ports there. However, there is at most one dangerous port in the closed part. Since there are no dangerous ports in the middle segment, at most Δ agents are blocked in dangerous ports. The witness graph is constructed from G_{ex} (the currently explored graph – the subgraph of G the algorithm has seen so far) by adding a new node u (the Bh) connected to all dangerous ports; in addition a loop of nodes is connected to G_{ex} at the two open nodes of the



(a)

(b)

Figure 6: (a) A stable configuration with three agents on dangerous links (shown by arrows) and others waiting (not shown). Note that there are two open nodes, with one and two unexplored links, respectively. (b) The witness graph the adversary responds with to the configuration in (a). The explored nodes are black, the newly added nodes are white. Note the additional edge added to maintain the proper node degrees in the witness graph.

open part, in order to obtain a consistent graph of n nodes. It may be the case that one or both of the open nodes have two incident unexplored links (and therefore not all unexplored links are covered by the newly added loop). If both open nodes have such spare unexplored port, an edge is added connecting those ports. If there is only one spare unexplored port, an edge is added leading to a node of the newly added ring (see Figure 6.(b)). Note that since there at at least three unexplored nodes at this moment, the newly added loop has at least two nodes and this can be done without creating a multiple edge.

In the particular case when there is only one dangerous port, the BH is also connected to this loop, to ensure 2-connectivity. (Note that since $\Delta \geq 3$, the degree of the BH would not exceed Δ.)

Therefore, in G_{wit} all dangerous ports lead to the BH. Since, by hypothesis, the open part does not switch to closed, it follows that no other agents ever reach the open part. This means the algorithm can no longer progress nor locate the black hole among the remaining unexplored nodes. □

We remind that $\mathcal{G}(\Delta, n)$ denotes the set of 2-connected graphs of n nodes of maximal degree at most $3 \leq \Delta \leq n-4$.
Based on the definitions introduced, we can finally state the following:

Theorem 3.6. *Let $\mathcal{A}$ be an algorithm that, with topological ignorance, correctly locates the* BH *in all graphs in $\mathcal{G}(\Delta, n)$ using $\Delta + 1$ agents, where $n \geq 7$ and $3 \leq \Delta \leq n-4$. Then there exists a $G \in \mathcal{G}(\Delta, n)$ such that $\mathcal{A}$ incurs $\Omega(n^2)$ moves on G in the worst case.*

Proof. During each change (or, flip) of open/closed parts, $\Delta - 1$ agents cross the middle explored part and this part grows by one basic cell. Since $\Delta + 3$ moves are needed to cross a basic cell (and the bridge connecting to the next cell), the number of moves between the i-th and $i+1$-th flip is at least $(\Delta - 1)(i - 1)(\Delta + 3)$. From Lemma 3.5, it follows that the number of flips is at least $\lfloor n/(2\Delta + 2)\rfloor - 1$; hence, summing until there is only the last basic cell left yields the result of the theorem:

$$\sum_{i=1}^{\lfloor n/(2\Delta+2)\rfloor - 1} (\Delta - 1)(i - 1)(\Delta + 3) = n^2/8 + o(n^2)$$

□

Note that, since a graph of degree 3 was used, this lower bound does not carry for $\Delta = 2$. However, only ring networks satisfy both $\Delta = 2$ and 2-connectivity, and for those matching upper and lower bounds of $\Theta(n \log n)$ are already known [20].

3.2 Optimal Protocol

In this section we show that the lower bounds established in the previous section are tight.

Let us refine the classification of the explored nodes as follows: (1) an explored node v whose ports are either explored or dangerous is called *expanded*, (2) otherwise v is called *visited*.

The idea of the algorithm, called Ignorance, is to have the agents cooperatively visit the graph by expanding all nodes until the black hole is localized. 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 our 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.

As we want at most two agents to expand each node, we associate an *asking* counter $c_a(v)$ with each explored node v: two minus the number of agents currently expanding v. Once v becomes expanded, $c_a(v)$ is set to 0. Thus, finding a node to expand means finding a node with asking counter greater than zero.

There are three main problems that have to be solved for Algorithm Ignorance to work: to ensure that (1) at least one agent survives, (2) each node is eventually expanded and (3) the agents efficiently find a node with non-zero asking counter.
The first problem is taken care of by using Cautious Walk. The second two problems are addressed by maintaining two counters c_n (globally needed) and c_e (already expanded) at the home base. The idea is to have c_n reflect the sum of all the asking counters c_a's; c_e is used to decide when to terminate. If an agent a waiting at the home base sees a non-zero c_n, it knows that somewhere there is an explored node asking for an agent. The agent a then traverses a spanning tree T constructed during the exploration of G (when a node u is visited for the first time, the node from which the agent came is set to be the u's parent in T) until it finds a node v with $c_a(v) > 0$. The counters c_n and $c_a(v)$ are updated in such a way that at any time c_n is at most the sum of all c_a's, and if the sum of c_a's is more than 0 then eventually also $c_n > 0$. The last property is achieved as follows: Whenever a new node v is explored, the agent visiting it not only sets $c_a(v)$ to two, but also travels to the home base and increments c_n by two (and then returns to the node it was expanding).

At any time, an agent will be either expanding a node, searching for a node to expand, waiting at the *home base* for an assignment, or being destroyed by the black hole.

Theorem 3.7. *Algorithm* IGNORANCE *correctly locates the black hole in $O(n^2)$ moves using $\Delta+1$ agents.*

Proof. Correctness. Since Algorithm IGNORANCE uses a cautious walk, at most one agent will enter each link incident to the black hole.

Let us denote by $\#_s$ the number of agents in *searching* state and by $\#_u$ the number of agents that are currently travelling to the home base to update c_n. By construction, it follows that at any time $c_n + \#_s + 2\#_u = \sum c_a(v)$.

Since a searching agent always finds a node with $c_a(v) > 0$ in at most $O(n)$ moves, livelock is not possible and the only possible terminal state of the system is with no agents moving. As the number of agents is higher than the degree of the BH, at least one agent must be waiting at the home base. However, that means $c_n = \sum c_a(v) = 0$, i.e. all nodes have been expanded.

Cost. A vertex v is charged (1) the cost of exploring its incident links - $O(\deg(v))$, (2) $O(n)$ for updating c_n when v is found for the first time and (3) the searching cost $2|T| \in O(n)$ of the (at most two) agents that decrement $c_a(v)$ and expand v. Summing up over all vertices results in $O(n^2)$ overall cost. □

We have presented Algorithm IGNORANCE that locates the BH using $\Delta+1$ agents and at a cost of $O(n^2)$ moves. By Theorems 3.4 and 3.6 this is optimal for $\Delta \leq n-4$. For the case of $n-4 < \Delta < n$, an optimal algorithm using Δ agents can be constructed based on Algorithm IGNORANCE, with a case-analysis final stage when only few nodes remain unexplored. Its description and analysis can be found in the Appendix.

4 Searching with Sense of Direction

In this section we show that, even in a situation of topological ignorance, if there is a sense of direction known to the agents, *two agents* suffice to locate the black hole, regardless of the topology of G.

Sense of direction is a property of labeled graphs capturing global consistency of the port labels [24]; it has been extensively studied in the context of distributed computing and has been shown to have an impact in reducing the communication complexity of several problems (e.g., see [25]).

Informally, sense of direction is the ability (called coding) to decide, by looking at sequences of labels corresponding to different paths starting from the same node, whether or not they end up in the same node; plus the ability (called decoding) to translate from neighbor to neighbor the coded information about paths in the system.

For some graphs (e.g., the ones used for interconnection networks), a sense of direction is provided by very natural labellings. This is the case, for example, of the mesh consistently labeled with North, South, East, West (*compass labelling*), or of the hypercube where the edges are labeled by the corresponding dimensions (*dimensional labeling*). Interestingly, every graph can be endowed with sense of direction.

Formally, sense of direction is based on the notions of *coding* and *decoding* functions for a labelled graph (G, λ).

A coding function for λ is any function $\mathbf{c} : \mathcal{L}^+ \rightarrow \mathcal{N}$ where $\mathcal{N}$ is a finite set, such that walks originating from the same node are mapped to the same element of $\mathcal{N}$ if and only if they end in

Algorithm 1 IGNORANCE

There are $\Delta + 1$ agents, with $\Delta \leq n - 4$. At the *home base* there are two counters: c_n ("agents-needed") and c_e ("nodes-expanded"). Furthermore, at each node u there is a counter $c_a(u)$ ("asking"). Initially, $c_a(h) = c_n = d(h)$ (the degree of the home base), $c_e = 0$, $c_a(u) = 0$ for all $u \neq h$, h is marked as visited, all agents are waiting at h. T is a tree spanning the explored nodes; at the beginning it contains only h.

Waiting for Assignment - Agent a is waiting at h while $c_n = 0$.

1. If $c_n > 0$ then a sets $c_n \leftarrow c_n - 1$ and starts searching for a node to expand.

Searching - Agent a is searching for a node to expand.

1. a traverses T until a node u with $c_a(u) > 0$ is found. A "right-hand-on-the-wall" rule for traversing mazes is used (the ports at a node are cyclically ordered: after arriving by port p, an agent leaves using the next port in the tree), as it does not require additional memory at the agent and/or nodes.
2. a sets $c_a(u) \leftarrow c_a(u) - 1$.
3. a starts expanding u.

Node Expansion - Agent a is at u expanding it.

1. a leaves through an unexplored port p of u, making it dangerous. If the incident node v had already been visited, a returns to u to continue its expansion. Otherwise
   (1) a sets $c_a(v) \leftarrow 2$, updates T by adding v and (u, v), and returns to u (making p explored).
   (2) a goes to h where it sets $c_n \leftarrow c_n + 2$ (to register that another node has been visited and two agents must be assigned to its expansion);
   (3) It returns to u to continue its expansion. It is sufficient for a to remember the identifier of u and traverse T until u is found. If the nodes did not have unique identifiers at the beginning, the algorithm can be modified to assign to each node a unique identifier when it is first visited.
2. If no port of u is unexplored (a has finished its expansion of u): if u is already marked as expanded, a returns to h to wait; otherwise, a marks u as expanded, returns to h, sets $c_a(u) \leftarrow 0$ and $c_e \leftarrow c_e + 1$, and waits.

Termination - When $c_e = n - 1$, then T includes every node except the black hole; furthermore, at that time, every port p that is still dangerous leads to the black hole.

the same node. More precisely, $\forall x, y, z \in V, \forall \pi_1 \in P[x, y], \forall \pi_2 \in P[x, z]$,

$$\mathbf{c}(\Lambda_x(\pi_1)) = \mathbf{c}(\Lambda_x(\pi_2)) \Leftrightarrow y = z.$$

The value of $\mathbf{c}(\Lambda_x(\pi_1))$ is also denoted by $\beta_x(y)$ and called the *local name* of y at x according to $\mathbf{c}$. Any coding function $\mathbf{c}$, if it exists, is also called a *weak sense of direction* of (G, λ).

A decoding function for $\mathbf{c}$ is a function $\mathbf{d} : \mathcal{L} \times \mathcal{N} \to \mathcal{N}$ which allows, for any two neighbors x and y and every walk π from y to any z, to determine the coding of the walk $\pi' =< \{x, y\}, \pi >$ from x to z just from the coding of π and the label of the link $\{x, y\}$. More precisely, $\forall x \in V, \forall \pi \in P[x]$, with $\Lambda_x(\pi) = [\alpha_1, \alpha_2, , \ldots, \alpha_k]$,

$$\mathbf{d}(\alpha_1, \mathbf{c}([\alpha_2, \ldots, \alpha_k])) = \mathbf{c}([\alpha_1, \alpha_2, , \ldots, \alpha_k])$$

In other words, while in general to know the coding of the labels of a walk we need to know all the labels, with the decoding function it is sufficient to know only the first label and just the coding of the rest.

An edge-labeled graph (G, λ) has *sense of direction* if and only if there exists a coding function $\mathbf{c}$ for (G, λ) and a decoding function $\mathbf{d}$ for $\mathbf{c}$. We also say that the pair $(\mathbf{c}, \mathbf{d})$ is a sense of direction for (G, λ).

Examples. Consider an arbitrary graph $G = (V, E)$, $V = \{v_0, \ldots, v_{n-1}\}$, where the edge (v_i, v_j) is labeled at v_i by the label $j - i$. With this labelling there is a simple coding function: two paths from the same node terminate in the same node iff the sum of the corresponding labels is the same, that is

$$\forall x \in V, \forall \pi \in P[x], \text{ if } \Lambda_x(\pi) = [l_0, \ldots, l_k] \text{ then } \mathbf{c}(\Lambda_x(\pi)) = \sum_{i=0}^{k} l_i.$$

The decoding function is defined as follows:

$$\forall \{x, y\} \in E, \forall \pi \in P[y], \mathbf{d}(\lambda_x(\{x, y\}), \mathbf{c}(\Lambda_y(\pi))) = \lambda_x(\{x, y\}) + \mathbf{c}(\Lambda_y(\pi)).$$

It is easy to verify that $\lambda_x(\{x, y\}) + \mathbf{c}(\Lambda_y(\pi)) = \mathbf{c}(\lambda_x(\{x, y\}) \circ \Lambda_y(\pi))$, where $\circ$ denotes concatenation of strings of labels. This labelling (called *chordal*) is one of many endowed with sense of direction that can be constructed in every graph.

Another labelling, endowed with sense of direction, that can be constructed in every non-anonymous graph is the *neighboring* one defined as follows: $\forall \{x, z\}, \{y, w\} \in E$, $\lambda_x(\{x, z\}) = \lambda_y\{y, w\}$ iff $z = w$. In this case the coding function is:

$$\forall x \in V, \forall \pi \in P[x], \text{ if } \Lambda_x(\pi) = [l_0, \ldots, l_k] \text{ then } \mathbf{c}(\Lambda_x(\pi)) = l_k$$

and the corresponding decoding function is:

$$\forall \{x, y\} \in E, \forall \pi \in P[y], \mathbf{d}(\lambda_x(\{x, y\}), \mathbf{c}(\Lambda_y(\pi))) = \mathbf{c}(\Lambda_y(\pi)).$$

We now show that, even in a situation of topological ignorance, if there is a sense of direction known to the agents, *two agents* suffice to locate the black hole, regardless of the topology of G. We do so constructively, by designing a solution protocol, called SD, that requires only two agents. We further prove that the proposed solution is also cost-optimal.

The idea of the algorithm is similar to the one of Algorithm IGNORANCE: the graph is traversed, expanding the encountered nodes and constructing a spanning tree T of G rooted at h. Unlike the previous solution, only two agents a and b are employed.

The two agents, a and b, cooperate to expand every node (except the black hole), starting from the *home base*. Using only two agents, the crucial and difficult task is to prevent both of them entering the BH. In particular, let u be the node currently being expanded. To expand u, agent a successively explores the incident unexplored ports: a leaves u via an unexplored port p, comes to a node v (if v is not a black hole), adds the link (v, u) to the spanning tree T of the explored subgraph if v has not been visited before, and returns back to u. This process is repeated until a returns to u and finds that no port is unexplored; then a continues the traversal, searching for another node to expand. However, if one port p of u is dangerous (i.e, the other agent is currently exploring it), the node w reachable through p might be the BH, and must be avoided. Thus, from this moment on, until told otherwise, agent a avoids entering an unexplored port leading to w. If a is not able to find an unexpanded node, w is the BH and the algorithm terminates.

Sense of direction (coding and decoding functions) is used to identify the ports leading to the dangerous node. While still at u, the agent a computes $\mathbf{c}(\lambda_u(\{u, w\})) = \beta_u(w)$, where $\lambda_u(\{u, w\})$ is the label of port p. Let z be the first node a reaches after departing u to find another node to expand. If z has an unexplored port, a can determine whether or not it leads to the dangerous node w as follows: It computes $\mathbf{d}(\lambda_z(\{z, u\}), \beta_u(w)) = \beta_z(w)$; a port with label l leads to w if and only if $\mathbf{c}(l) = \beta_z(w)$. Similarly, at any node z' in the traversal, a computes $\beta_{z'}(w)$ and uses it to determine if an unexplored port of z' leads to the dangerous node w.

The information that w is dangerous can be modified only by the other agent b. If w was not the BH, when b completes the exploration of v, it becomes a *follower* and tries to reach a following it in its traversal. Once b reaches the node being expanded by a, it leaves a message for a notifying it of its presence (and, thus, that w is no longer dangerous), and joins in the expansion (leaving through an unexplored port, or starting traversal, if there are none left).

Thus, at any point in time, an agent is either expanding a node (*Node Expansion*), or searching for a node to expand (*Searching*), or following the other agent (*Following*), or destroyed by the black hole. In the first two cases, the agent might carry with it information, in the variable *danger*, about the dangerous node to be avoided. As before, we use a spanning tree T of all visited nodes; initially, T contains just h.

Theorem 4.1. *In an arbitrary network with sense of direction, the black hole can be located by* two *agents with cost $O(n^2)$.*

Proof. Correctness: We start by showing that at most one agent will enter the black hole. First, note that a *Following* agent never enters an unexplored port. In fact, when an agent becomes *Following* (Rule 1.2.1), it follows the trace of the other agent until it reaches a node that is being expanded by the other agent (Rule 3); in doing this, it only follows safe links until it reaches the node that the other agent is currently expanding. Second, if both agents are expanding the same node (Rule 1), since the graph is simple, at most one of them enters the black hole. Finally, the only possibility for the agents to expand different nodes u and v is when one (say b) is exploring the last unexplored port p of a node u (leading to a node w) (Rule 1.1), while a has already left u in search of the next node v to expand (Rule 1.3). In such a case a remembers the "name" (variable *danger* in Rule 2.1) of w and avoids entering any port leading to w, thanks to the properties of sense of direction.

Thus, at least one agent survives; it eventually explores $n-1$ nodes and terminates, correctly identifying the only remaining node (*danger*) as the black hole.

Cost: The total cost for expanding the nodes is $O(m)$ (each link is visited at most twice). The number of moves made by a following agent from the moment it became follower until it caught up with the other agent is bounded by $O(n)$ (the follower visits each node at most once).

Algorithm 2 Sd

1. **Expanding** - Let a be the agent expanding node u; let $\widetilde{w}$ be the current dangerous node (if any) and $danger = \beta_u(\widetilde{w})$ be the information known to a (if there is no dangerous node, $danger = nil$).

    1.1 If there is an unexplored port p of u not leading to the dangerous node: a leaves u through p making it dangerous, and returns to u (making p explored). If the incident node v had not been previously visited, a updates T by adding v and (u, v) to it.

    1.2 If all ports are explored and none is leading to the dangerous node:

    1.2.1 If there is a note from b indicating that b is searching, a removes the note, becomes *Following*.

    1.2.2 If there are no notes from b, a marks u as expanded and becomes *Searching*.

    1.3 If one port p, leading to node w, is dangerous and all the others are explored (i.e., b is currently exploring link (u, w) and the dangerous node is w), a will: set $danger = \beta_u(w) = \mathbf{c}(\lambda(u, w)) = \mathbf{c}(l)$, where l is the label of p; mark u as expanded, and becomes *Searching*.

2. **Searching** - Let a be the searching agent. Agent a traverses T searching for another node to expand, leaving a navigational instruction for b (a pebble marking the port over which a leaves the current node is sufficient). Let $\widetilde{w}$ be the current dangerous node (if any); thus, $danger = \beta_u(\widetilde{w})$. Upon arriving at v:

    2.1 a sets $danger = \beta_v(\widetilde{w}) = \mathbf{d}(\lambda(v, u), \beta_u(\widetilde{w}))$ updating the information about the dangerous node.

    2.2 If v has been already expanded, a continues its traversal of T searching for another node to expand, and leaves a navigational instruction for b.

    2.3 If v is unexpanded, a becomes *Expanding* to start the expansion of v.

3. **Following** - Let a arrive at v while following b. If v is not expanded, a becomes *Expanding* (i.e., it joins b in the expansion); otherwise a follows the navigational command left there by b.

4. **Termination** - When the searching agent finds that all nodes in T are expanded, the node identified by $danger$ is the black hole.

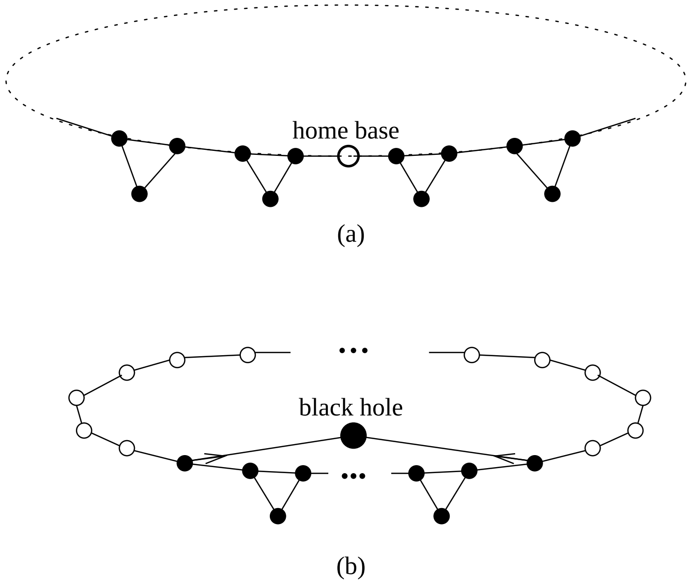


Figure 7: Theorem 4.2: (a)The lower bound graph. (b) Directing the eagerly exploring agents to the black hole.

An agent becomes follower only when the last port of a node is explored, hence the total cost following is $O(n^2)$. The cost of finding the next node to expand is $O(n)$ (traversal of the tree T). The total cost of finding is again $O(n^2)$, as for a given node at most one agent (the first one to not find an unexplored port leading to non-dangerous node) leaves it in a search for the next node to expand. □

We now show that the quadratic bound on the cost is tight in the worst case. More precisely, we show that any algorithm that works with all senses of direction (i.e. does not assume a specific sense of direction) incurs a worst case cost of $\Omega(n^2)$. We prove this by showing that there exists a type of sense of direction (the *neighboring SD* described earlier in this section) for which $\Omega(n^2)$ is the lower bound.

In a graph with neighboring sense of direction, the labels on all the edges leading to the same node are identical; i.e, $\forall x, y, z, w \in V$: $\lambda_x\{x, z\} = \lambda_y(y, w)$ iff $z = w$; we shall call such a label the identity of node z. The lower bound is based on the observation that, although SD reveals to an agent the identity of the node on the other side of an incident unexplored link, it gives no information about what is deeper inside the unexplored part.

Theorem 4.2. *The worst case cost for locating a black hole in arbitrary networks, independently of the type of sense of direction available, is* $\Omega(n^2)$.

Proof. To prove the lemma we show that with a specific sense of direction (neighboring SD) $\Omega(n^2)$ is a lower bound. Let $\mathcal{A}$ be a two agents algorithm that correctly locates the black hole in arbitrary networks with neighboring SD. We view the execution as a game between the adversary and the algorithm, played on the graph shown in Figure 7(a): a ring with each vertex except the *home base* replaced by a "triangle". The game proceeds in rounds. At the beginning of each round both agents are leaving the explored part in opposite directions over the "ring edges" (the edges coming from the original ring; the edges of the triangles are called *triangle* edges).

The beginning of the first round is reached by the adversary blocking both edges incident to the *home base* and waiting until both agents depart in the opposite direction (they must do so, otherwise the algorithm does not correctly handle the situation with a black hole incident to the *home base*).

During a round, the adversary performs two tests: (1) If the left blocked edge is unblocked, will the agent return to the middle node of the explored part before entering a triangle edge of the next unexplored triangle? (2) The same test on the right side. Note that these tests are only virtual, by examining the algorithm. A negative answer to both tests means that the agents proceed (without communicating) to explore parts of the graph about which they have no knowledge. In such a case, the adversary forces the algorithm to direct both agents to the black hole (see Figure 7(b)) contradicting the correctness of the algorithm.

Hence, at most one agent enters a triangle edge without returning to the middle of the explored part. The adversary unblocks the agent a that would return there. Then the whole next triangle is revealed and the next unexplored ring edge is blocked (the other agent remains blocked all the time on the ring edge on the opposite side). Eventually, the freed agent enters the blocked ring edge and the next round starts (otherwise the algorithm would not locate the black hole, as there will be more than one unexplored node remaining).

Note that, since during round p the explored subgraph G_e^p contains $p-1$ triangles, the freed agent performs $\Omega(p)$ moves. Because the arguments above apply for all rounds with at least one unexplored triangle (i.e. $p < n/4-1$), summing over all such rounds results in the $\Omega(n^2)$ lower bound. □

5 Searching with Complete Knowledge

In this section we consider the case of an arbitrary system where the agents have complete topological knowledge of (G, λ). In this case, not surprisingly, two agents suffice, even if there is no sense of direction. In fact, with complete topological knowledge, each agent can unambiguously determine the node being visited by the other agent, and thus avoid that node. In other words, we can employ the solution strategy for systems with sense of direction, even if there is no sense of direction. This yields, by Theorem 4.1, a two-agents solution with $O(n^2)$ cost.

We show how, using a different approach and making full use of the available knowledge, the two agents can locate the black hole with at most $O(n \log n)$ moves, which is optimal. The proposed algorithm, called CompleteKnowledge, does not use collaborative "expansion" of the nodes, employed by all previous protocols. Instead, it is based on the notion of individual "working sets" and of "safe" nodes, using an approach similar to the one for ring networks [20]. The *safe* nodes are those in the subgraph induced by the explored links.

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 *home base* h. The nodes to be explored are partitioned, and each agents explores its part ("working set"). In particular, the two agents a and b, partition the unexplored area into disjoint subgraphs G_a and G_b, such that for each connected component of G_a and G_b there is a link connecting it to G_{ex} (we show later that this is always possible). G_a and G_b are the working sets of a and b, respectively. Let us define T_a and T_b to 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

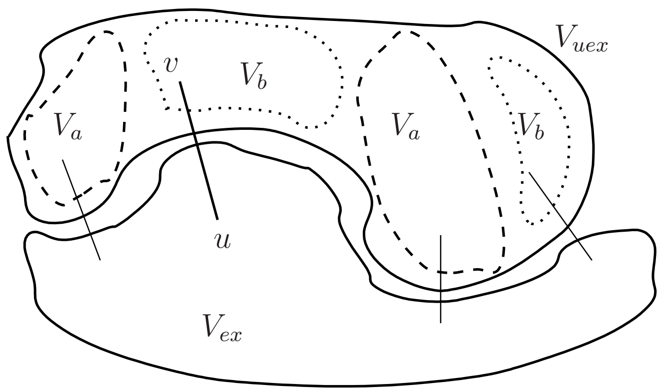


Figure 8: Splitting the unexplored subgraph G_{uex} into G_a and G_b.

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 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.

In the correctness proof of Algorithm CompleteKnowledge we need the following special case of the Györi-Lovasz Theorem [31, 37]:

Lemma 5.1. *Let $G = (V, E)$ be a 2-connected graph and let x and y be any two nodes in V. Let $s < |V|$. Then there exists a partition of V into V_1 and V_2 such that $|V_1| = s$, $x \in V_1$, $y \in V_2$ and the graphs induced in G by V_1 and V_2 are connected.*

The following lemma states that the Step 1 of *Computing the Working Sets* can always be performed:

Lemma 5.2. *Let $G = (V, E)$ be a 2-connected graph and let $\{V_{ex}, V_{uex}\}$ be a partition of V such that G_{ex} (the graph induced by V_{ex}) is connected. Let $(u, v) \in E$, where $u \in V_{ex}$ and $v \in V_{uex}$, and let $1 \leq s < |V_{uex}|$. Then V_{uex} can be partitioned into V_1 and V_2 such that $|V_1| = s$, $v \in V_1$ and every connected component of G_a and G_b (the graphs induced in G by V_1 and V_2) is directly reachable from G_{ex}.*

Proof. We prove that the sets V_a and V_b computed by Algorithm PartitionC satisfy the lemma.

Let G_{uex} be the graph induced by V_{uex} in G. Each connected component of G_{uex} can be viewed as a set of 2-connected components[3] connected by the bridge edges into a tree of 2-connected components.

[3] or a single node

Algorithm 3 COMPLETEKNOWLEDGE

Let G_{ex} be the explored part of the network and G_{uex} be the remaining unexplored part; initially, G_{ex} consists only of the *home base* h, and $G_{uex} = G \setminus G_{ex}$.

Computing the Working Sets - Let a be the agent computing the working sets, and let u be the node where this is done (initially $u = h$).

1. a partitions G_{uex} into disjoint subgraphs, G_a and G_b of (almost) equal size, such that for each connected component of G_a and G_b there is a link connecting it to G_{ex} (Algorithm PARTITIONC). If this is not the first assignment of working sets (i.e. u is the last safe node visited by b, and b has left u towards node v) then G_b is the subgraph containing v.
2. a leaves a note for b informing it of the new working set G_b, and leaves to explore its working set G_a.

Exploring the Working Set -

1. a uses the shortest possible route in G_{ex} to reach a node in T_a.
2. a explores G_a using *cautious walk* on T_a.
3. During the cautious walk, if a finds at a node w a note from b informing it of the new working sets G_a and G_b, a starts the exploration of the (new) G_a.
4. If a completes the exploration of its working set, it will search for b to recompute the working sets.

Searching for the other agent - Let a be searching for b.

1. Let u be the last node where the working sets have been computed. a uses the shortest path avoiding G_b to reach u.
2. Starting from u, a follows the trace of b (a shortest path in G_{ex} to reach T_b, and then the safe links of the T_b) until it reaches the last safe node w reached by b.
3. a computes the new working sets.

Termination - When computing the working set, if G_{uex} contains a single node, that node is the black hole.

Algorithm 4 PartitionC

1: Keep adding the connected components of G_{uex} to V_a (starting with the component containing v), until V_a contains at least s nodes.
2: **If** $|V_a| > s$ **Then**
3: Remove the last added connected component C from V_a.
4: **If** $v \in C$ **Then**
5: Add to V_a the 2-connected component of C containing v.
6: **Else**
7: Add to V_a any 2-connected component of C from which there is an edge leading to a node in V_{ex}.
8: **While** $|V_a| < s$ **Do**
9: Find a 2-connected component B of C which neighbors an already added 2-connected component of C and add B to V_a.
10: **If** $|V_a| > s$ **Then**
11: Remove the last added 2-connected component B from V_a.
12: Let $s' = s - |V_a|$.
13: Use Lemma 5.1 applied to B to choose the s' nodes to be added to V_a. (If $v \in B$, a is set to v, otherwise a is any node of B which has a neighbor in $V_a \cup V_{ex}$. $b \neq a$ is any other node of B with a neighbor in $V \setminus (V_a \cup B)$. Note that since G and B are 2-connected, such b must exist.)
14: Set $V_b = V_{uex} \setminus V_a$.

Since $s < |V_{uex}|$, by construction we get that Algorithm PartitionC eventually constructs V_a of size s. The reachability property for V_a follows by construction (line 9. for 2-connected components fully in V_a) and from Lemma 5.1 (for the last 2-connected component that is only partially in V_a). Since the fact that $v \in V_a$ also follows from construction (lines 1. and 5.), we have only to prove the reachability property for V_b.

By construction, we have that there is at most one connected component C of G_{uex} shared between V_a and V_b, all other connected components are either fully in V_a or fully in V_b. Therefore, it is sufficient to prove reachability only for $V_b \cap C$.

The component C can be seen as a rooted tree T of 2-connected components, with the root being the 2-connected component of C that was first added to V_a. From construction (line 5.) we have that $V_a \cap C$ is a connected subtree of C and at most one 2-connected component (B from line 9.) is shared between V_a and V_b. This means that $V_b \cap C$ is a union of connected components, with each connected component containing a 2-connected component that is a leaf in T. From the 2-connectivity of the original graph G it follows that each leaf 2-connected component has an edge leading to V_{ex}. Therefore, each such a connected component of $V_b \cap C$ satisfies the reachability property. There is one possible exception – if $V_b \cap B$ is not connected to another part of V_b. In such case the reachability property for $V_b \cap B$ follows directly from Lemma 5.1. $\square$

The existence of a link between V_{ex} and V_a implies that a can safely reach the part (subgraph G_a induced by V_a) it has to explore. The lemma is more general than strictly needed, as only the case $s = |V_{uex}|/2$ is used by the algorithm.

Theorem 5.3. *The black hole can be located by two agents with full topological knowledge in arbitrary networks of vertex connectivity* 2 *with cost* $O(n \log n)$*, and this is optimal.*

Proof. Correctness: Since G_a and G_b are disjoint, one agent always survives. This, together with the fact that agents never wait for each other, ensures progress. Lemma 5.2 used with $s = |V_{uex}|/2$ allows to halve in each round (iteration of the main loop) the size of the unexplored area. This means after $O(\log n)$ rounds the algorithm terminates.

Cost: Note that each action an agent (say a) performs within one round is either (1) a local computation, or (2) moving to another node of G_{ex}, or (3) traversing T_a or (4) following the traces of b in T_b). Clearly, the cost of each of these actions is at most $O(n)$. Since the number of such actions per round is constant, and there are only two agents, the total cost of one round is $O(n)$. As the number of rounds is $O(\log n)$, the total cost of Algorithm 3 is $O(n \log n)$.

The cost optimality follows from the $\Omega(n \log n)$ lower bound for ring networks by [20]. $\square$

Note that the analysis above uses very weak arguments to prove that the cost of one round is $O(n)$. Nevertheless, they can not be improved in general – the cost of communication (an agent finding the other agent, and leaving a message) in the ring network is $O(|G_{ex}|)$, which over all rounds sums to $O(n \log n)$.

6 Conclusions and Open Problems

We have studied conditions under which a team of autonomous asynchronous mobile agents can identify the location of the harmful host. We have shown that the size and the cost of an optimal solution depend on the a priori knowledge the agents have about the network, and on the consistency of the local port labellings. In particular, we have provided tight bounds when computing with topological ignorance, in presence of sense of direction, and with complete topological knowledge.

Let us stress that, in the case of topological ignorance, the lower bound on the cost depends on the smallest upper bound known to the agents on the maximal degree of the graph. In particular, if the agents know only that it is possible for the graph to have maximal degree Δ, then any team of $\Delta + 1$ might incur worst time cost of $O(n^2)$, regardless of the actual degree of the graph; in fact, in the proof we use a graph of maximum degree 3. In other words, at play is not only structural property (the maximal degree of the graph) but also and more predominantly the a priori information (the upper bound Δ) the agents have about that property.

Our results provide a characterization of the impact that factors such as a priori network knowledge and consistency of the local port labellings have on the complexity of the black hole location problem. This characterization, far from being complete, poses many new questions and opens many new research problems, some of them outlined below.

Topological ignorance and complete topological knowledge represent the two extreme levels of possible knowledge. Interesting natural questions arise about the intermediate levels of topological knowledge. In particular, can two agents locate the black hole with $O(n \log n)$ cost with less than complete knowledge?

We have shown that the presence of sense of direction allows two agents to overcome any ignorance about the topology. Because of its computational equivalence with sense of direction, it follows that the same result holds if the labelled graph has backward consistency[4] [26]. An

[4] Backward consistency requires the existence of a "backward coding function" $\mathbf{c}_b : \mathcal{L}^+ \to \mathcal{N}$ such that walks ending in the same node are mapped to the same element of $\mathcal{N}$ if and only if they start from the same node.

immediate question is whether there are other consistency properties capable of similar results.

In our model, agents communicate by means of whiteboards. An interesting research area is to investigate the black hole location problem using other models of inter-agent communication. Examples of such means range from the more powerful *messaging service* (providing communication between agents regardless of their locations, e.g., [18]) to the weaker *tokens* used mostly as markers (e.g., [9, 36]). Such an investigation would have many important objectives, e.g. to identify the weakest type of inter-agent communication mechanism that allows black hole location by two agents with a network map.

The graphs we have considered are simple; indeed our results do not carry over to multigraphs. The problem of locating the black hole in multigraphs is open.

All our protocols are deterministic. This leaves the door open to the obvious and natural question of determining the impact, if any, of randomization.

Finally, an immediate obvious generalization of our investigation is to consider the presence of more than one black hole. Some of the existing results generalize immediately. For example, it is clear that the graph must be at least $k+1$ connected to deal with the presence of k black holes. However the problem in its generality is open.

Acknowledgments

The authors would like to thank the anonymous referees, whose comments and suggestions have contributed to greatly improve the presentation, and Doug West for pointing out the source of Lemma 5.1.

This research has been partially supported by "Progetto ALINWEB: Algoritmica per Internet e per il Web", MIUR Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale, and by the Natural Sciences and Engineering Research Council of Canada.

References

[1] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29:1164–1188, 2000.

[2] S. Alpern, V. Baston, and S. Essegaier. Rendezvous search on a graph. *Journal of Applied Probability*, 36(1):223–231, 1999.

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

[4] I. Averbakh and O. Berman. A heuristic with worst-case analysis for minimax routing of two traveling salesmen on a tree. *Discrete Applied Mathematics*, 68:17–32, 1996.

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

[6] L. Barrière, P. Flocchini, P. Fraigniaud, and N.Santoro. Rendezvous and election of mobile agents: impact of sense of direction. *Theory of Computing Systems*. To appear.

[7] L. Barrière, P. Flocchini, P. Fraigniaud, and N.Santoro. Capture of an intruder by mobile agents. In *Proc. 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA'02)*, pages 324–332, 2002.

[8] L. Barrière, P. Flocchini, P. Fraigniaud, and N.Santoro. Can we elect if we cannot compare? In *Proc. 15th ACM Symposium on Parallel Algorithms and Architectures (SPAA'03)*, pages 200–209, 2003.

[9] M. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *Proc. 30th ACM Symposium on Theory of Computing (STOC'98)*, pages 269–287, 1998.

[10] M. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proc. 35th Symposium on Foundations of Computer Science (FOCS'94)*, pages 75–85, 1994.

[11] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM Journal on Computing*, 26:110–137, 1997.

[12] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. 19th Symposium on Foundations of Computer Science (FOCS'78)*, pages 132–142, 1978.

[13] D. M. Chess. Security issues in mobile code systems. In *Proc. Conference on Mobile Agent Security*, LNCS 1419, pages 1–14, 1998.

[14] X. Deng, T. Kameda, and C. H. Papadimitriou. How to learn an unknown environment i: The rectilinear case. *Journal of the ACM*, 45:215–245, 1998.

[15] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

[16] A. Dessmark, P. Fraigniaud, and A. Pelc. Deterministic rendezvous in graphs. In *Proc. 11th European Symposium on Algorithms (ESA'03)*, pages 184–195, 2003.

[17] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. In *Proc. 10th European Symposium on Algorithms (ESA'02)*, pages 374–386, 2002.

[18] D. Deugo. Mobile agents for electing a leader. In *Proc. 4th International Symposium on Autonomous Decentralized System*, pages 324–324, 1999.

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

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

[21] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Finding a black hole in an arbitrary network: optimal mobile agents protocols. In *Proc. of 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*, pages 153–162, 2002.

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

[23] O. Esparza, M. Soriano, J.J. Munoz, and J. Forne. Host revocation authority: A way of protecting mobile agents from malicious hosts. In *Proc. International Conference onWeb Engineering (ICWE'03)*, LNCS 2722, 2003.

[24] P. Flocchini, B. Mans, and N. Santoro. Sense of direction: definition, properties and classes. *Networks*, 32(3):165–180, 1998.

[25] P. Flocchini, B. Mans, and N. Santoro. Sense of direction in distributed computing. *Theoretical Computer Science*, (291):29–53, 2003.

[26] P. Flocchini, A. Roncato, and N. Santoro. Backward consistency and sense of direction in advanced distributed systems. *SIAM Journal on Computing*, 32(2):281–306, 2003.

[27] P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. In *Proc. 6th Latin American Theoretical Informatics Symp. (LATIN'04)*, pages 141–151, 2004.

[28] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. In *Proc. 29th Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 451–462, 2004.

[29] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7:178–193, 1978.

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

[31] E. Gyuri. On division of graphs to connected subgraphs. In *Proc. 5th Hungarian Combinatorial Colloquium*, 1976.

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

[33] F. Hohl. A framework to protect mobile agents by using reference states. In *Proc. of the 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, 2000.

[34] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[35] D. Kozen. Automata and planar graphs. In *Proc. Fundations Computatial Theory (FCT'79)*, pages 243–254, 1979.

[36] E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Mobile agent rendezvous in a ring. In *Proc. International Conference on Distibuted Computing Systems (ICDCS'03)*, pages 592–599, 2003.

[37] L. Lovàsz. A homology theory for spanning trees of a graph. *Acta Mathematica Academiae Scientiarum Hungaricae*, 30(3-4):241–251, 1977.

[38] H. Muller. Automata catching labyrinths with at most three components. *Elektronische Informationsverarbeitung und Kybernetic*, 15(1):3–9, 1979.

[39] S.K. Ng and K.W. Cheung. Protecting mobile agents against malicious hosts by intention spreading. In *Proc. 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 725–729, 1999.

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

[41] P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33:281–295, 1999.

[42] M.O. Rabin. Maze threading automata. Technical Report Seminar Talk, University of California at Berkeley, October 1967.

[43] H.A. Rollik. Automaten in planaren graphen. *Acta Informatica*, 13:287–298, 1980.

[44] N. Roo, S. Hareti, W. Shi, and S. Iyengar. Robot navigation in unknown terrains: Introductory survey of length,non-heuristic algorithms. Technical Report ORNL/TM12410, Oak Ridge National Lab., 1993.

[45] T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts. In *Proc. Conference on Mobile Agent Security*, LNCS 1419, pages 44–60, 1998.

[46] K. Schelderup and J. Ones. Mobile agent security - issues and directions. In *Proc. 6th International Conference on Intelligence and Services in Networks*, LNCS 1597, pages 155–167, 1999.

[47] CL. E. Shannon. Presentation of a maze-solving machine. In *Proc. 8th Conference of the Josiah Macy Jr. Found. (Cybernetics)*, pages 173–180, 1951.

[48] Jan Vitek and Giuseppe Castagna. Mobile computations and hostile hosts. In D. Tsichritzis, editor, *Mobile Objects*, pages 241–261. University of Geneva, 1999.

[49] X. Yu and M. Yung. Agent rendezvous: A dynamic symmetry-breaking problem. In *Proc. International Colloquium on Automata Languages and Programming (ICALP'96)*, pages 610–621, 1996.

Appendix

A Case $n-4 < \Delta \leq n-1$

By Theorem 3.4, we know that when $n-4 < \Delta \leq n-1$ there are cases when Δ agents are sufficient, and thus Algorithm Ignorance is not optimal. In this section we present an optimal protocol for the case $n-4 < \Delta \leq n-1$: Algorithm IgnoranceΔ. This new protocol is based on algorithm Ignorance: the main idea is to make sure that at most $\Delta - 1$ nodes are being expanded simultaneously. Since from each node there is at most one link to the black hole, this guarantees that at least one agent survives. However, if the black hole is of degree Δ, it may be necessary to expand Δ nodes simultaneously in order to complete exploration of the graph. In such a case, algorithm IgnoranceΔ carefully chooses in procedure `Choose()` the order in which to expand the remaining few nodes, so that when the algorithm terminates $n-1$ nodes are explored, and one agent survives. There might be however an ambiguity about three unexplored ports: two correspond to the same link, connecting two safe nodes, while the other leads to the black hole; in other words, two ports might be incorrectly suspected. We conjecture that this level of ambiguity is actually inevitable when using only Δ agents in this restricted range of values. In the following, by correct termination we mean termination with this level of ambiguity.

In order to describe algorithm IgnoranceΔ in more detail, we need to refine the classification of the nodes in the following way. An unexplored node that is reached for the first time by an agent turns into *visited*: such nodes have only one explored port and all the others are unexplored. When a port of a visited node becomes dangerous, the node turns into *partially expanded*. A node stays partially expanded until it has exactly one dangerous incident edge and all the remaining edges are explored: it is now *quasi expanded*. Finally, when a node has all its incident edges explored, it is called *fully expanded*. A node that is either unexplored or visited is called *unexpanded*.

Algorithm IgnoranceΔ makes sure that at most $\Delta - 1$ nodes are being expanded simultaneously by having, at the beginning, $\Delta - 1$ tokens at the *home base*; moreover, a node is being expanded if and only if it contains one token (except the *home base*, which may contain more tokens). The modifications to Algorithm Ignorance to ensure this property are reported in Algorithm 5.

Procedure `Choose()`. Line 14 of Algorithm IgnoranceΔ handles the case when $c_n > 0$, there are no more tokens left in h, there are at most $\Delta-1$ quasi expanded nodes and no partially expanded node: Procedure `Choose()` is called (reported in Figure 9). First note that, since when this procedure is called $\Delta-1$ agents are already employed in expanding nodes, it cannot happen that more than one agent concurrently execute this procedure: let c be the agent that executes `Choose()`. Furthermore, at the moment `Choose()` is called, the number of unexpanded nodes is at most 4, since $\Delta \geq n-3$ and $\Delta - 1$ nodes are quasi-expanded. Finally, at least one of the unexpanded nodes is visited, otherwise the explored part of the network would be connected to the unexplored part only through the Bh, and G would not be 2-connected.

This procedure uses case analysis to choose the next node to expand. In particular, the choice depends on the number of the remaining unexpanded nodes and on the state of the links incident to them. Once such a node has been located, say u, c expands it by calling routine `Expand(`u`)`: c first reaches u (with no token), and then starts expanding it as in the **Node Expansion** of Algorithm Ignorance. If, during the expansion, c finds in u a token, it exits procedure

Algorithm 5 IGNORANCEΔ

1: When an agent b detects that there are nodes to expand ($c_n > 0$ at the home base) and starts to search for a node to expand, it first tries to take a token. (If the home base is still being expanded, it claims its token and the agent can only take a different token.)

2: **If** there is an available token **Then**

3: b takes the token and travels until it finds a node v to expand (with $c_a(v) > 0$).

4: **If** $c_a(v) = 2$ **Then**

5: b places the token at v, decrements $c_a(v)$ and starts expanding v.

6: **Else**

7: %$c_a(v) = 1$, *i.e. v is already being expanded by another agent. Note that v already contains a token.*%

8: b decrements $c_a(v)$, brings *its* token back to h and returns to v to help the other agent in expanding v.

9: **If** when b returns to v from the *home base*, v is already at least quasi-expanded **Then**

10: b returns to h, increments c_n and starts again.

11: **Else**

12: b joins expanding v.

13: **Else**

14: %$c_n > 0$ *and there are no more tokens left.*% Agent b still tries to find a node to expand, but can only join in the expansion of a partially expanded node. If b is not able to find such a node (i.e., it traversed whole T and returned back to h) then it executes procedure `Choose()` to choose the next node to expand, and before leaving h to explore this node, it leaves a note at h.

15: When an agent finishes expanding a node (so that the node is fully expanded), it brings the token back to the home base. If it realizes that an agent is expanding a node chosen by `Choose()` (by finding the note left in Line 14), it removes the note and brings the just freed token to the node being expanded.

`Choose()` and continues the expansion as if it were executing Algorithm IGNORANCEΔ: that is, having a token, c is now performing a *legal* expansion of node u. Such a token can be found in u by c in the following scenario.

Scen When `Choose()` is called, not all of the $\Delta - 1$ quasi expanded nodes are connected to the BH. In this case, according to IGNORANCEΔ, one of the $\Delta - 1$ quasi expanded nodes will eventually turn into fully explored, and agent b that completed the expansion of this node will free its token, bringing it back to h. At this point, it realizes that an agent is expanding a node, say u, returned by `Choose()` (by finding the note left at h, Line 14 of IGNORANCEΔ); therefore, it brings to u the token it just freed. When c realizes the presence of this token in u, it exits `Choose()` (by routine `Expand(`u`)`) and keeps executing IGNORANCEΔ.

Theorem A.1. *Algorithm* IGNORANCEΔ *correctly locates the black hole using* Δ *agents, in* $O(n^2)$ *moves.*

Proof. Correctness. First of all notice that, if the black hole is not located by procedure Choose(), the execution of algorithm IGNORANCEΔ can be seen as a special case of the execution of algorithm IGNORANCE: In algorithm IGNORANCEΔ the procedure Choose() chooses the next node to explore, in algorithm IGNORANCE the next node is chosen arbitrarily. Therefore, if the BH is not located in Choose(), the correctness of algorithm IGNORANCEΔ follows from correctness of algorithm IGNORANCE. Hence, we only need to show that procedure Choose() correctly locates the black hole in order to prove correctness of algorithm IGNORANCEΔ.

We analyze procedure Choose() depending on the number of unexpanded nodes at the beginning of its main *while* loop.

1. *Single unexpanded node v:* The black hole must be in v and Choose() correctly identifies it and terminates at Line 27.
2. *Two unexpanded nodes u and v:* Since one of these nodes must be visited, the other one must contain the black hole. Again, Choose() handles this case correctly at Line 29.
3. *Three unexpanded nodes u, v and w.* There are two cases:
   (a) If two of these nodes (say u and v) are visited, the black hole must be in w. Note that at this moment, the unexplored ports in u and v can only lead either to each other or to the black hole. Since, by hypothesis, the black hole is already connected to $\Delta - 1$ quasi expanded nodes, it cannot be connected to both u and v. That means that the node with the minimal number of unexplored links cannot be connected to the black hole, and can be safely expanded in Line 9; at this point, the number of unexpanded nodes decreased by one, and previous Case b. applies.

   Of course, if such a node has no unexplored links left, it is not necessary to expand it at all (line 11).

   (b) The second case is when only one (e.g. v) of the three nodes has been visited. Then v is not connected to the BH and can be safely expanded in Line 4. In fact, if by contradiction v is connected to the black hole, then the BH cannot be connected to the last (remaining) unexpanded node, contradicting the 2-connectivity of G (i.e., removing v disconnects the graph). With the expansion of v, the number of unexpanded nodes decreased by one, and previous Case 2. applies.

Procedure `Choose()`

```
While the number of unexpanded nodes is at least 3 Do
    If there are three unexpanded nodes u, v and w Then
        If only one of them (say u) is visited Then
            Expand(u).
        Else
            Let u and v be the visited nodes. Clearly, the black hole is in w.
            If the total number of unexplored ports from u and v is at least 2 Then
                f := node among u, v with fewer unexplored links
                Expand(f).
            Else
                Terminate, the black hole is in w.
    Else
        %There are four unexpanded node u, v, w and t.%
        If only one one of them (say u is visited) Then
            Expand(u).
        Else If two of them (say u and v) are visited Then
            f := node among u and v with fewer unexplored incident edges, or any
            of them if they have the same number of incident unexplored edges;
            Expand(f).
        Else
            There are three (say u, v and w) visited nodes. Clearly, the BH is in t.
            If all of them have single incident unexplored link Then
                Terminate, the black hole is in t. Declare as dangerous all remaining
                unexplored links.
            Else
                f := node among u, v, and w with the minimal number of unex-
                plored incident links;
                Expand(f).
If there is a single unexpanded (and visited) node v left Then
    Terminate, the black hole is in v and all dangerous links lead to it. (Note that
    this may happen only if the degree of v is at most Δ − 1.)
Else
    There are two unexpanded nodes u and v. One of them (e.g. v) is visited.
    Terminate, the black hole is in u.
```

Figure 9: Procedure `Choose()` called in Line 14 of Algorithm IGNORANCEΔ.

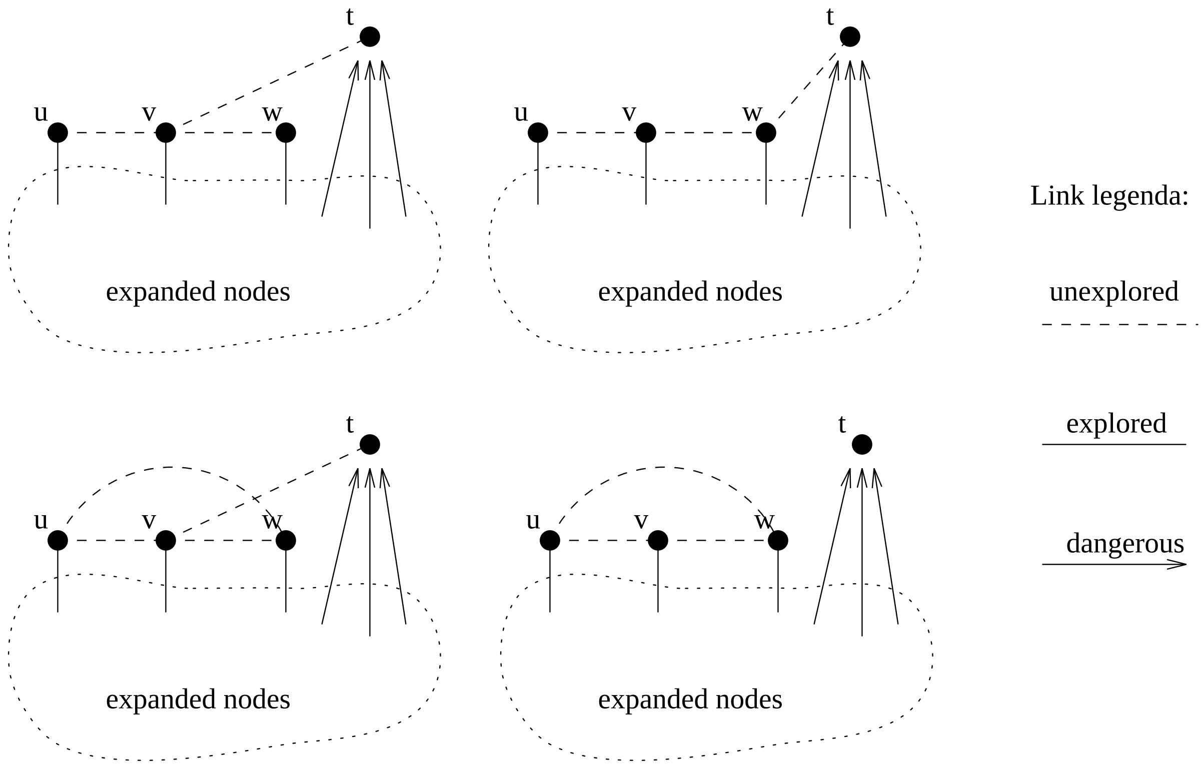


Figure 10: The only possible configurations (up to symmetries) of unexplored ports in the remaining unexpanded nodes.

4. *Four unexpanded nodes u, v, w and t.* Similarly to the previous case, if there is a single visited node among u, v, w and t then it is not connected to the black hole and can be safely expanded. At this point, only 3 unexpanded nodes are left, and previous Case 3. applies.

   If there are three visited nodes (say u, v and w) then the black hole is located at t. If each of u, v and w has single unexpanded port, then the algorithm declares as dangerous all the corresponding unexplored links (only on of them leads to the black hole, but the algorithm cannot determine which one without expending an agent), and correctly terminates. Otherwise the node with minimal number of unexplored links is not connected to the black hole and can be safely expanded in Line 24: If this minimal number is 0, that node is clearly not connected to the black hole. The remaining possible cases (up to permutation of u, v and w) are shown in Figure 10, clearly the node(s) with minimal number of unexplored links is(are) not connected to the black hole. After the expansion of Line 24, only 3 unexpanded nodes are left, and previous Case 3. applies.

   Finally, if there are two visited nodes (w.l.o.g. assume u and v) the one with minimal number of unexplored ports is not connected to the black hole and can be safely expanded at Line 18: This is certainly true if the black hole is connected to the non-visited node. Otherwise (the BH is connected to either u or v), since the graph is 2-connected, the non-visited node must be connected to both u and v. The claim now follows immediately, as the only other unexplored link can be the link connecting u and v. After the expansion of Line 18, only 3 unexpanded nodes are left, and previous Case 3. applies.

Number of moves: The only additional moves of algorithm IGNORANCEΔ with respect to algorithm IGNORANCE are incurred due to the token management. Since each expanded node and each call/exit of Choose() incurs cost at most $O(n)$, the overall overhead is bounded by $O(n^2)$. □