Exploring an Unknown Dangerous Graph Using Tokens*

Stefan Dobrev † Paola Flocchini‡ Rastislav Královič § Nicola Santoro ¶

Abstract

Consider a team of (one or more) mobile agents operating in a graph G. Unaware of the graph topology and starting from the same node, the team must explore the graph. This problem, known as *graph exploration*, was initially formulated by Shannon in 1951, and has been extensively studied since under a variety of conditions. Most of the existing investigations have assumed that the network is *safe* for the agents, and the vast majority of the solutions presented in the literature succeed in their task only under this assumption.

Recently, the exploration problem has been examined also when the network is *unsafe*. The danger examined is the presence in the network of a *black hole*, a node that disposes of any incoming agent without leaving any observable trace of this destruction. The goal is for at least one agent to survive and to have all the surviving agents to construct a map of the network, indicating the edges leading to the black hole. This variant of the problem is also known as *black hole search*. This problem has been investigated for the most part assuming powerful inter-agent communication mechanisms: *whiteboards* at all nodes. Indeed, in this model, the black hole search problem can be solved with an optimal team size and performing a polynomial number of moves.

In this paper, we consider the less powerful *enhanced token* model: each agent has available a token that can be carried, placed on a node or on a link, and can be removed from it. All tokens are identical and no other form of marking or communication is available. We constructively prove that the black hole search problem can be solved also in this model; furthermore, this can be done using a team of agents of optimal size and performing a polynomial number of moves. Our algorithm works even if the agents are *asynchronous* and if both the agents and the nodes are *anonymous*.

1 Introduction

1.1 The Problem

The problem of exploring an unknown graph using a team of one or more mobile agents (or robots) is a classical fundamental problem that has been extensively studied since its initial formulation in 1951 by Shannon [36]. It requires the agents, starting from the same node, to visit within finite time all the sites of a graph whose topology is unknown to them. Different instances of the problem exist depending on whether or not the agents are required to eventually *stop* the exploration; and, if so, whether or not

*Some of these results have been presented at the *5th IFIP Int. Conference on Theoretical Computer Science* [17].

†Institute of Mathematics, Slovak Academy of Sciences, Dubravska 9, P.O.Box 56, 840 00 Bratislava, Slovak Republic. stefan@ifi.savba.sk

‡School of Electical Engineering and Computer Science, University of Ottawa, Ottawa, Ontario, K1N 6N5, Canada. flocchin@eecs.uottawa.ca. Partially supported by NSERC Discovery Grant and University Research Chair.

§Dept. of Computer Science, Comenius University, Mlynska dolina, 84248 Bratislava, Slovak Republic. kralovic@dcs.fmph.uniba.sk. Partially supported by grant VEGA 1/0671/11

¶School of Computer Science, Carleton University, Ottawa, K1S 5B6, Canada. santoro@scs.carleton.ca. Partially supported by NSERC Discovery Grant.

they must construct an accurate *map* of the network. Further differences exist depending on a variety of factors, including the (a)synchrony of the agents, the presence of distinct agent identifiers, the amount of memory, the coordination and communication tools available to the agents, etc. (e.g., see [1, 4, 5, 6, 14, 15, 26, 27, 29, 35]). The exploration with stop of anonymous graphs is usually done by marking the nodes in some way; various methods of marking nodes have been used by different authors ranging from the weak model of *tokens*, where a pebble can be dropped/picked up at a node, to the most powerful model of *whiteboards*, where some storage area accessible in mutual exclusion by the agents is available at the nodes.

The solutions proposed in the literature succeed in their task only assuming that the network is *safe* for the agents. This assumption unfortunately does not always hold in real systems and networks; for example, a node could contain a local program (virus) that harms the visiting agents; or the network could contain failed nodes that might damage incoming agents. In fact, protecting an agent from "host attacks" (i.e., harmful network sites) is a pressing security concern (e.g., see [7]).

Recently the exploration problem has been examined also when the network is unsafe (e.g., see [10, 13, 11, 16, 19, 18, 20, 25, 30, 31, 32, 33]). The danger considered is the presence in the network of a *black hole* (BH), a node that disposes of any incoming agent without leaving any observable trace of this destruction. Note that such a dangerous presence is not uncommon; in fact, any undetectable crash failure of a site in an asynchronous network transforms that site into a black hole. In spite of this severe danger, the goal is for a team of system agents starting from the same node, the homebase, to be able to explore the network and, within finite time, discover the location of the black hole BH. More precisely, at least one agent must survive, and any surviving agent must have constructed a map of the network indicating the edges leading to BH.

This version of the exploration problem is called *black hole search* (BHS). It is known that, for its solution, the number of nodes of the network must be known to the agents [19]; furthermore, if the graph is unknown, at least $\Delta + 1$ agents are needed, where Δ is the maximum node degree in the graph [18]. In the case of asynchronous agents in an unknown network, termination with an exact complete map in finite time is actually impossible; in fact, regardless of the protocol, a surviving agent upon termination can be wrong on $\Delta - deg(\text{BH})$ links, where $deg(x)$ denotes the degree of node x [18]. Hence, in the case of asynchronous agents, BHS requires termination by the surviving agents within finite time and creation of a map with just that level of (in)accuracy.

The problem of asynchronous agents exploring a dangerous graph has been investigated assuming powerful inter-agent communication mechanisms: *whiteboards* at all nodes. In the whiteboard model, each node has available a local storage area (the whiteboard) accessible in fair mutual exclusion to all incoming agents; upon gaining access, the agent can write messages on the whiteboard and can read all previously written messages. This mechanism can be used by the agents to communicate and mark nodes or/and edges, and has been employed e.g. in [14, 16, 18, 19, 20, 26, 27]. In the whiteboard model, the black hole search problem can be solved with an optimal team size and performing a polynomial number of moves (e.g., [16, 18, 19, 20]).

It is important to understand that whiteboards provide agents not only with a communication mechanism but also with an additional computational power, that of breaking symmetry. In fact, since access to a whiteboard is provided in mutual exclusion, and since the agents start from the same node, unique ids can be provided to the agents; additionally, the uniquely named agents can provide each node with a unique public name (by writing it on the whiteboard). In other words, in the whiteboard model, if agents start from the same node, both the agents and the network can be assumed to be *non-anonymous*, without any loss of generality.

The principal question targeted by our research is the impact of the communication model to the

solvability and complexity of the BHS problem: to what extent can be the whiteboard model weakened, and still allow the polynomial solvability of BHS? With this goal in mind, we examine the problem of performing black hole search in the less powerful *token* model, which is instead commonly employed in the exploration of safe graphs. In the classical (or "pure") token model, each agent has available a token that can be carried, can be placed in the center on a node, or removed from it. All tokens are identical (i.e., indistinguishable) and no other form of marking or communication is available. In the *enhanced token* model considered here, tokens can be placed also on a node in correspondence to a port. Notice that the classical token model can be implemented with 1-bit whiteboards, while the enhanced variation is not as weak; in fact, it could be implemented by having a $O(d)$-bits whiteboard on a node with degree d.

Several immediate computational and complexity questions naturally arise. In particular, are the weaker communication and marking capabilities provided by enhanced tokens sufficient to solve the problem? If so, how can the problem be solved? In this paper we provide definite answers to these questions.

1.2 Our Results

In this paper we present an algorithm that works in the enhanced token model and solves the BHS problem without a map with the optimal number of agents and with a polynomial number of moves. Our algorithm works even if the agents are *asynchronous*, and if both the agents and the nodes are *anonymous*. More precisely, we consider an unknown, arbitrary, anonymous network and a team of exploring agents starting their identical algorithm from the same node (*homebase*). The agents are anonymous, they move from node to neighboring node asynchronously (i.e., it takes a finite but unpredictable time to traverse a link). Links satisfy the FIFO property; that is, agents moving from a node to a neighbor will arrive in the same order they departed.

Each agent has available an indistinguishable token (or pebble) that can be placed on, or removed from, a node; on a node, the token can be placed either in the center or on an incident link. In our algorithm there are never two tokens placed on the same location (node center or port), nor an agent ever carries more than one token.

Using only this tool for marking nodes and communicating information, we show that with $\Delta + 1$ agents the exploration can be successfully completed (recall that $\Delta + 1$ agents are needed, even with whiteboards [18]) . In fact, we present an algorithm that will allow at least one agent to survive and, within finite time, the surviving agents will know the location of the black hole with the allowed level of accuracy. The number of moves performed by the agents when executing the proposed protocol is shown to be polynomial. The proposed algorithm is unfortunately not simple.

These results are the first that address the problem of exploration of a dangerous unknown graph using tokens. Our results indicate that, perhaps contrary to expectation, the enhanced token model is computationally as powerful as the whiteboard one with regards to black hole search, albeit the complexities are different.

1.3 Related Work

The research on safe *exploration* of unknown graphs was started in 1951 by Shannon [36]. Most of the work since has been concentrated on exploration by a single agent (e.g., [4, 15, 28, 35]). Safe explorations by *multiple* agents were initially studied for a team of *finite-state automata* (e.g., [6]); more recently the investigations have focused on collaborative exploration by *Turing machines*. An exploration algorithm

for directed graphs that employs two agents was given in [5], whereas algorithms for exploration by more agents were given by Frederickson et al. for arbitrary graphs [29], by Averbakh and Berman for weighted trees [1], and more recently by Fraigniaud *et al.* for trees [26] and arbitrary graphs [28]. To explore arbitrary anonymous graphs, various methods of marking nodes have been used by different authors. Bender *et al.* [4] proposed the method of dropping a token on a node to mark it and showed that any strongly connected directed graph can be explored using just one token, if the size N of the graph is known, and using $\Theta(\log \log N)$ tokens, otherwise. Dudek *et al.* [23] used a set of distinct markers to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [5] was to employ two cooperating agents, one of which would stand on a node, thus marking it, while the other explores new edges. In Fraigniaud *et al.* [26, 27], marking is achieved by accessing whiteboards located at nodes, and their strategy explores directed graphs and trees.

Studies concerning the exploration of *unsafe* graphs are quite recent and they concern both synchronous and asynchronous environments. In *synchronous* environments, the investigations have produced optimal solutions for trees [13]. In the case of general networks the decision problem corresponding to the one of finding the optimal strategy is shown to be NP-hard [12, 31] and approximation algorithms are given in [12] and subsequently improved in [32]. More recent results include [8, 9].

Mostly the research has focused on *asynchronous* environments. The BHS problem has been studied when the network is an anonymous ring, characterizing the limits and determining optimal solutions [3, 19]. When the network is an arbitrary graph the problem has been investigated in [18], and several tight bounds have been established, depending on the level of topological knowledge available to the agents. For example, when the network is arbitrary, the topology unknown and no form of consistent edge labelings are present, $\Delta + 1$ agents are necessary and $\Theta(N^2)$ moves are required in the worst case, where N is the size of the network. Improved bounds on the number of moves have later been obtained in the case the agents have a complete map of the network (but not the location of BH) [20]. In the case of specific graphs, including many important interconnection networks, the number of moves can be reduced to linear [16]. A variant of dangerous node behavior is studied in [34], where the authors introduce *gray holes* (i.e. black holes with Byzantine behavior, which do not always destroy a passing agent) and consider the periodic ring exploration problem.

In these investigations for asynchronous environments, the nodes of the network have available a *whiteboard*, i.e., a local storage area that the agents can use to communicate information. Access to the whiteboard is gained in mutual exclusion and the capacity of the whiteboard is always assumed to be at least of $\Omega(\log N)$ bits.

Following the preliminary announcements of our results [17], a number of investigations have been carried out to better understand the computational power of *tokens* in the exploration of dangerous graphs [21, 22, 24, 37]; all these results however assume that the networks topology is *known* to the agents. The same token model considered in this paper (where a token can be placed on a node in correspondence of ports) has been investigated in [21, 22, 37] in the case of some known specific topologies (the ring and some interconnection networks). Finally, the classical token model where the pebble can only be placed in the center of a node has been studied in [24] for arbitrary but known topologies; it has been shown that, when the topology is known, whiteboards and pebbles have the same computational power.

The contributions of this paper are still the only results available for the more difficult case when the networks topology is arbitrary and unknown.

2 The Model

The network $G = (V, E)$ is a simple undirected graph with node-connectivity two or higher. Note that 2-connectivity is a necessary condition for being able to solve BHS even in the whiteboard model [18]. Let $N = |V|$ and $M = |E|$ be the number of nodes and of edges of G, respectively, $d(x)$ 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 are *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). Without loss of generality, we assume that the labels at $x \in V$ are the consecutive integers #1, #2, ..., #$d(x)$.

Operating in G is a team of $\Delta + 1$ anonymous (i.e., identical) agents. The agents have no knowledge of the network except for the number of nodes N and the maximum degree Δ. Each agent can move from node to a neighboring node in G, has computing capabilities and limited amount of memory ($O(M \log N)$ bits suffice for our algorithm).

Each agent has a token that can be placed on a node and removed from it; tokens are identical and their placement can be used to mark nodes and ports/links. More precisely, a node can be marked by a token in different modalities: in the center, or in correspondence of one of the incident ports.

The agents obey the same set of behavioral rules (the "algorithm") and initially, they are all located at the same node H, called *homebase*.

The agents can be seen as automata, where one computational step of an agent A in a node v is defined as follows. Based on the state (local memory) of A and on the presence of tokens at v and incident links (examined atomically):

1. change the state (local memory of A);
2. remove (or place) at most one token from v or an incident link; and
3. start waiting (for a token to disappear) or leave v via one of the incident links.

The computational steps are atomic and mutually exclusive, i.e. no more than one agent computes in the same node at the same time. More precisely, the configuration of the system at any particular time consists of the locations of the tokens (in a node , at a link, or held by an agent), internal states of the agents, and locations of the agents (on a link or waiting in some node). The computation is governed by a scheduler that, given a configuration, selects an agent to perform a computational step. The selected agent is either one residing on a link, in which case it is taken to the end-node , and performs its step, or one waiting in a node where the waiting condition is satisfied. Note that the scheduler may not be fair, i.e. the algorithm must ensure no livelock can happen even if some agents are infinitely faster than others. The links satisfy FIFO property, i.e. the agents entering a link $e = (u, v)$ at u will arrive at v and execute the computational steps in the same order they entered e. The agents are *asynchronous* in the sense that traversing a link takes an unpredictable (but finite) amount of time; similarly, when waiting (for a token to disappear), the waking up happens after an unpredictable but finite amount of time.

The network contains a *black hole* (BH) that destroys any incoming agent without leaving any trace of that destruction.

The goal of a *black hole search* algorithm $\mathcal{P}$ is to identify the location of BH; that is, within finite time, at least one agent must terminate, and all the surviving agents must construct a map of the entire graph where the homebase, the current position of the agent, and the location of the black hole, are indicated.

Note that termination with an exact map in finite time is actually impossible. In fact, since an agent is destroyed upon arriving to BH, no surviving agent can discover the port numbers of the black hole.

Hence, the map will have to miss such an information. More importantly, the agents are asynchronous and do not know the actual degree $d(\text{BH})$ of the black hole (just that it is at most Δ). Hence, if an agent has a local map that contains $N-1$ vertices and at most Δ unexplored edges, it cannot distinguish between the case when all unexplored ports lead to the black hole, and the case when some of them are connected to each other; this ambiguity can not be resolved in finite time nor without the agents being destroyed. In other words, if we require termination within finite time, an agent might incorrectly label some links as incident to BH; however the agents need to be wrong only on at most $\Delta - d(\text{BH})$ links. Hence, we require from a solution algorithm $\mathcal{P}$ termination by the surviving agents within finite time and creation of a map with just that level of accuracy.

The complexity measures of a solution protocol are: the number of agents used, called *size* of the team, and the total number of moves performed by the agents during the execution, called *cost*.

3 The Solution

3.1 Overview

In our algorithm, each agent constructs its own local map (quasi-)independently from other agents until it enters the black hole BH or explores at least $N-1$ vertices and $M-\Delta$ edges.

In the beginning, the local map of each agent contains only the homebase H. During the computation, the communication ports in the graph are classified by each agent as follows:

- *unexplored* port/edge – not in the local map: the port is not marked by a token
- *dangerous* port – not in the local map; the port is marked by a token
- *safe* edge – in the local map; connecting two already explored vertices
- *quasi-safe* edge – presumed to be in the local map; however, the assumption is not confirmed, and the edge may actually lead to an unexplored node

Throughout the execution, whenever an agent leaves via a port that might lead to BH, it leaves its token there, marking the port as *dangerous*. The algorithm requires that no agent enters a *dangerous* port, ensuring in this way that at most Δ agents enter the black hole. We will thus say that a *dangerous* port *blocks* the (other) agents.

The modalities of token placings are used to distinguish various parts of the algorithm: tokens are placed on links only to mark a dangerous step. All other actions including the determination of the safety of newly explored edges, coordination of agents etc. are performed only by placing tokens in vertices (actually, there are only three vertices where a token may be possibly placed, one of them being the homebase).

Initially, all ports incident to H are *unexplored*. The local map of an agent is constructed by adding edges in a sequential manner according to Algorithm 1.
The search for an unexplored port is straightforward: any traversal of the explored part, using only the edges identified as *safe* in the local map, will do.

In the execution of EXPLORE(p), the agent explores the edge incident to port p, determines whether it leads to a new node or to an already discovered one[1], and updates the local map. Due to the complex interaction of anonymity with asynchrony, in some cases the agent might be unsure of whether an edge leads to a new node or to an already visited one. However, the error is one-sided: if the agent adds a new

[1] p might lead to BH as well, in which case the agent disappears there and does not continue the algorithm.

Algorithm 1 General algorithm of an agent

```
1: INITIALIZE
2: loop
3:     traverse the local map and look for an unexplored port p
4:     if unexplored port p found then
5:         EXPLORE(p)
6:         continue the main loop
7:     else
8:         if local map contains N − 1 vertices and there are at most Δ unexplored, or dangerous,
   edges then
9:             TERMINATE
10:        else
11:            SUSPEND    // wait at the homebase
12:        end if
13:    end if
14: end loop
```

node v to its local map, v was indeed a new node , and the corresponding edge is marked *safe*. Hence, the safe edges always comprise a spanning tree of the local map constructed so far, and are used for moving inside the explored subgraph. The *quasi-safe* edges, on the other hand, may lead to a node not yet in the map.

Eventually, no unexplored port is found. If $N-1$ nodes have been visited, the remaining node is BH and the algorithm can terminate. Otherwise, in this case the access to the unexplored part of the graph is blocked by *dangerous* ports. As we will see, either one of those ports does not lead to BH and the token will eventually be removed from it (making it *unexplored*), or some of the *quasi-safe* edges in fact lead to unexplored vertices (this situation is handled in the proof of Lemma 16). In order to avoid livelock, the agent that failed to find an *unexplored* port suspends itself using procedure SUSPEND until such a progress has been made. The basic idea of SUSPEND is to go to H, set a flag there (by using a token) indicating that an agent is waiting for wake-up, verify that no progress has been made before the flag has been set up, and then wait to be woken-up. Complementarily, whenever an agent removes its token from an edge, it goes to the homebase and wakes up the agents waiting there (using procedure WAKE-UP). There are several technical issues to be dealt with (discussed in the detailed description), e.g. several agents might be executing SUSPEND and WAKE-UP simultaneously, the flag can only be implemented using tokens, as well as the interference with the rest of the algorithm.

3.2 Detailed Description

In this section, we give the full description of the algorithm. The nodes on the other ends of the links #1 and #2 from H are called **storerooms** (SR), denoted by SR1 and SR2, respectively; they play a special role in the algorithm (as we will see, they will be employed to allow communication among the agents when they are temporary suspended looking for a new port to explore).

The following three rules clarify some terms used in the description:

R1 "cautious step in node v over link l" $\equiv$

Put a token on link l, traverse the link, return to v, take the token, perform WAKE-UP, return

to v and traverse l.

R2 **"put token in an empty node v (homebase or storeroom)"** $\equiv$
- If v is H then wait for all known safe links incident to H to become unmarked, then put the token.
- If v is a storeroom then put token, go to H and return.[2]

R3 **"put token on link l at node v"** $\equiv$

Wait for the center of v to become empty, then put the token on link l.

Initialize

Each agent starts the algorithm by exploring (using cautious step) SR1 and SR2 from H (in this order). Since the graph is simple, at least one of them does not lead to BH; if both these links are *dangerous*, the agent will simply wait until one of the blocking tokens disappears (Algorithm 2). Eventually, each agent will know about one or two safe storerooms. The *primary storeroom* for an agent is defined as the storeroom, known by the agent to be safe, with the lower numbered link leading to it.

Algorithm 2 INITIALIZE

```
1: while both ports #1 and #2 are marked do
2:     wait;
3: end while
4: let x be the smallest unmarked port;
5: perform a cautious step over x;
6: return to H and set x as primary storeroom;
```

Note that if BH is one of the storerooms, all surviving agents will choose the other storeroom as their primary storeroom. However, if none of the storerooms is BH, there might be agents with different primary storerooms (some might find SR1 safe and choose it, some might find it temporarily dangerous and select SR2). This disagreement among agents may lead to problems in verifying the local map: an agent that does not know that SR1 is safe may mark as *quasi-safe* an edge that actually leads to a new node in its local map. To remedy this situation, the algorithm uses an update rule **R4** (described in detail later): if, at any time during the execution of the algorithm, an agent with primary storeroom SR2 enters the homebase, and sees the port #1 free, then it makes SR1 its primary storeroom, and restarts the algorithm. It is shown that with the correct information about storerooms, all *quasi-safe* edges are identified correctly.

Explore

The execution by agent A of procedure EXPLORE(p) is to enable A to traverse an unexplored edge $e = (u, v)$ (starting at port p in u) and add it (possibly with the node v) to the local map. Agent A starts by executing a cautious step over the edge e and, if it survives, it proceeds with determining whether or not e leads to a new (not in the local map) node.

Notice that recognizing if v is already in the local map would be an easy task if either the agents were able to recognize their own tokens, or they were able to recognize the homebase. In fact, if agents were able to recognize their tokens, then A could simply put its token at v and scan the explored subgraph: if it finds its token, v is already explored, otherwise it is a new node.

[2]This effectively "flushes" the edge to H, making some arguments about FIFO simpler.

If the agents were able to recognize the homebase, then A could determine whether v is a new node as follows. For each node w in the local map, A guesses that $v = w$ and verifies whether that is really true: Let α be a sequence of port labels specifying a safe path (determined by looking in the local map) from w to H. Starting from v, A follows the port labels specified by α; note that cautious walk needs to be used, as v might be different from w, and starting from v α might lead to BH. If A finishes in H, then $v = w$, otherwise A makes another guess. If all guesses fail, v is a new node.

Unfortunately, in our model, the agents cannot recognize their tokens nor the homebase. Still, the basic structure is to guess for all already explored nodes w whether $v = w$ and to verify the guess, although the verification is much more involved.

To verify the guess that $v = w$, let us consider the edge (u, w) added to the spanning tree consisting of *safe* edges. If the formed cycle C does not contain H, add the path from H to C. If the resulting (possibly not simple) cycle does not contain SR x as a neighbor of H, add the edge between H and SR x. Resulting is a (possibly not simple) cycle that starts from SR x, continues to u, to w, to H, and back to SR x (see Figure 1). Let β_w (or simply β if w is clear from context) be the sequence of port labels following this cycle, noting both the outgoing and incoming ports (clearly, $|\beta| \leq 2N - 2$).

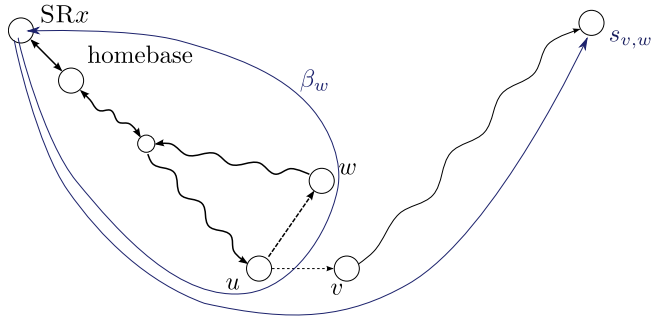

Figure 1: β_w is the sequence of port labels such that if $v = w$ then traversing β from SR x forms a cycle of length at most $2N - 2$. If $v \neq w$, traversing β_w from SR x may lead to a node $s_{v,w} \neq$ SR x.

Consider now the sequence of ports β_w starting from SR x. If $v = w$, the sequence correctly describes the cycle back to SR x. If $v \neq w$, either a discrepancy occurs (i.e. the incoming port number in some node is different from that specified by β_w), or the path ends in some node $s_{v,w}$. Note the following immediate but important property:

Claim 1 *Consider an edge (u, v), and a node $w \neq v$. If the traversal of the sequence β_w from* SR x *does not show any discrepancy, it ends in a node $s_{v,w}$, such that $s_{v,w} \neq$* SR x.

During the exploration, the agent A first checks whether there is some discrepancy traversing β (in which case it is clear that $v \neq w$). If no discrepancy is found, A ends the traversal at some node $s_{v,w}$; it then verifies whether $s_{v,w} =$ SR x (which implies $v = w$) using procedure VERIFY described later.

In the traversal, the steps along β that go past v must be done in a cautious manner, never entering *dangerous* ports, since it may be the case that $v \neq w$ and β leads to BH. The cautious walk is complicated by the fact that a port to be taken (let its label be λ) from a node w' might be marked as *dangerous*. In such a situation, if $v = w$ the token will be removed, and A can continue its cautious walk through λ. On the other hand, the agent cannot afford to wait in w' until the token is removed, because this edge might indeed lead to BH.

To resolve this situation, A goes backwards for $|\beta|$ steps reaching a safe node through safe links; this node might indeed be w' (this happens if the guess $v = w$ is correct), or it could be a different node w''.

Algorithm 3 EXPLORE: Exploring from node u the edge with label l_1 leading to node v.

```
1: do a cautious step over link l_1, let l_2 := label of the link upon which you arrived;
2: for all w in local map, different from u and from the neighbors of u in local map do
3:     go to primary storeroom, and compute the sequence β = β_w;
4:     for |β| steps do
5:         while next port γ in β is dangerous and looped less than K times do
6:             go back |β| steps;
7:             wait until there is no token on the edge along which you arrived;
8:             go forwards |β| steps;
9:         end while
10:        if port γ is still dangerous then
11:            backtrack your steps to v and continue the outermost for cycle for the next w;
12:        end if
13:        do a cautious step;
14:        if what you see is not compatible with the local map assuming v = w then
15:            backtrack your steps to v and continue the outermost for cycle for the next w;
16:        end if
17:    end for
18:    if VERIFY then
19:        add edge to w to the local map as quasi-safe;
20:        exit from EXPLORE;
21:    end if
22: end for
23: add to the local map node v and the edge (u, v) labeled l_1 at u and l_2 at v; mark the edge as safe;
```

Agent A waits here until there is no token on the port labelled λ. Although not sure about the identity of the node, the agent knows that λ must lead to a safe node (A is now revisiting nodes it has visited earlier), thus the token will be eventually removed from there.

After ensuring the removal of the token, agent A returns to w'. It can happen that the port λ is still *dangerous*. However, if $v = w$ then this must be a newly placed token. For the correctness of the procedure, and also for the verification procedure introduced next, the following property is crucial:

Property 1 *Let $K = (4N+1)M(N-1)(\Delta+1)$. The event that some token is removed from some link occurs less than K times. Moreover, let us consider a* SR *x. The event that a token is put in* SR *x occurs less than K times.*

This property will be proved later, in Section 4.

From Property 1 it follows that if the agent repeats at least K times the described traversal to w'' and back, and the port λ is still *dangerous*, then $w' \neq w''$, and hence $v \neq w$. The procedure EXPLORE is described in full detail in Algorithm 3.

Notice that, during the actual exploration, tokens are placed in correspondence to *links only*. Thus, a token found on a link is a clear sign of danger. As we will soon discover, both in the verification process (described below) and in the suspension process (described later), tokens are instead placed in (and removed from) the homebase and the storerooms. More precisely, an agent puts its token in a storeroom as part of the verification process, and it expects the token to stay there until it collects it (the token may actually be taken by another agent). In the suspension/wakeup process, on the other hand, the agent puts its token in the homebase, and waits until another agent moves the token to the storeroom as a wakeup signal. Since these two processes may interfere (e.g. a token in a storeroom may be either a mark for some agent performing VERIFY or a wakeup sign for another agent), and the agents do not recognize their own tokens, neither they agree on the number of safe storerooms, much care must be given to keep the possible interactions under control.

Verification

The test of a candidate node w in the procedure EXPLORE may end either by finding a discrepancy, or by arriving to the node $s_{v,w}$, denoted in the following simply as s. In the latter case, if $v = w$ then $s =$ SR x, and if $v \neq w$ then $s \neq$ SR x. The procedure VERIFY (see Algorithm 4) is used to test whether $s =$ SR x.

The idea of VERIFY is for the agent to put a token in the primary storeroom, and check whether s contains a token. An agent A performing VERIFY first makes sure that it is not interfering with any other agent; it does so by waiting until both H and the storerooms it knows to be safe are empty. It then puts its token in the primary storeroom and walks along β to s, and checks whether there is a token in s. (Note that it does not need to use cautious steps, as the used edges have already been traversed and are known to be safe.) The idea is that, if $v = w$, then s is the storeroom and contains the token, while, if $v \neq w$, then s should be empty as it is not the correct storeroom. Notice that a straightforward check on whether there is a token in s can fail for two reasons.

1. It may happen that s is the homebase H (the only other node that can contain a token). As mentioned above, H is also used by procedure SUSPEND and WAKE UP, which are employed when an agent has not found a suitable port to explore and is waiting for one to become available. If some other agent has started to perform a SUSPEND (which requires putting a token in H) while A traveled to s, A is deceived since it finds a token in s, but this is not the token it left in storeroom.

2. It may happen that s is indeed a storeroom but some other agent took the token from the storeroom in the meanwhile (when finishing SUSPEND); so A is again deceived because it does not find its own token.

However, as we show later, from Property 1 it follows that each of these two cases occurs less than K times. Thus, if A saw a token in s at least K times, then s must be the storeroom; conversely, if A saw no token in s at least K times, then s is not the storeroom.

The procedure VERIFY is described in full in Algorithm 4. Note that on line 20 the agent arrives to the storeroom from H.

Algorithm 4 VERIFY: Check if the newly explored node v is indeed new (not in the local map).

Rule **R4** is not applied during traversals of β
1: $PosCount$:= $NegCount$:= 0;
2: backtrack your steps to H;
3: **loop**
4: wait until it becomes empty, and go to the primary storeroom;
5: **if** the *storeroom is empty* **then**
6: put token and exit loop;
7: **else**
8: wait until the storeroom becomes empty and go to H;
9: **end if**
10: **end loop**
11: **while** $PosCount < K$ and $NegCount < K$ **do**
12: if known, go to the other storeroom and wait until it becomes empty;
13: go to H, wait until it becomes empty and all known safe ports are free;
14: go to the primary storeroom, and follow β to s;
15: **if** *there is a token* **then**
16: $PosCount$:= $PosCount + 1$;
17: **else**
18: $NegCount$:= $NegCount + 1$;
19: **end if**
20: follow the reverse of β to the primary storeroom and if empty update the knowledge of storerooms using **R4**;
21: **end while**
22: take token
23: **if** $PosCount \geq K$ **then**;
24: return TRUE:
25: **else**
26: return FALSE:
27: **end if**

One last complication comes from the fact that, at the beginning of each iteration of the while cycle, A has to make sure that H and the safe storerooms are empty. The problem is that agents cannot always agree on which storerooms are safe. In fact, even if BH is not a storeroom, there are three types of agents: those who think that only SR1 is safe, those who think that only SR2 is safe, and those who know that both storerooms are safe. However, if an agent does not know that both storerooms are safe, it cannot ensure that both of them are empty. In this case, it may happen that the result of VERIFY is

wrong: v is actually a new node, but VERIFY returns FALSE. On the other hand, it will be shown that if VERIFY returns TRUE then v is indeed a new node, and if the agent has correct information about the storerooms, VERIFY never errs. This is the reason why, when A decides that $v = w$, it marks the edge (u, v) as *quasi-safe* and never uses it for traversals, so the spanning tree defined by the *safe* edges is always available for traversal.

As we prove later, the only way for an agent A to find an empty storeroom on line 20 is if A does not know about (safe) SR1. This means that, after seeing an empty storeroom, A can update its knowledge about the storerooms and reset the algorithm according to rule **R4** below.

R4 $\equiv$ When an agent first realizes that both storerooms are safe, it performs the following actions:

1. If you have no token and your old primary storeroom is SR2, go to SR2 and, if there is a token there, get it.
2. Go to H, and if you have no token execute GRAB-TOKEN.
3. If you came to H to perform WAKE-UP but have not done so, do it now.
4. Update the knowledge about the storerooms and restart Algorithm 1 from line 2 with the knowledge of both storerooms.

Note that, since the links marked as *safe* are always correct, when applying rule **R4**, the agent could actually reuse the part of the local map consisting of *safe* links when restarting the algorithm; however, for simplicity of presentation and discussion, we assume that the whole algorithm is restarted.

Unless explicitly stated (line 20 of VERIFY, procedure GRAB-TOKEN, traversing β in VERIFY), this rule is activated anytime an agent A, who remembers that some storeroom was marked as dangerous, arrives to the homebase and sees that the link to that storeroom is unmarked.

Grab-Token

The procedure GRAB-TOKEN is used by an agent A to pick up a token that it has previously put at H or a storeroom. It might happen that some other agent B has meanwhile picked A's token instead of its own. However, in such a case, B's token must be in H or in a storeroom, and A will take it (or the token of yet another agent). Note that, on line 4, the agent may actually follow a link marked as dangerous. Also note that, on line 5, the agent already has the token, so in fact only the last two items of **R4** are executed here.

Algorithm 5 GRAB-TOKEN – start in H, do not apply **R4** during the execution

1: if there is a token in H, get it and exit GRAB-TOKEN
2: go to primary storeroom, if there is a token there, get it and exit GRAB-TOKEN
3: go to H and if there is a token there, get it and exit GRAB-TOKEN
4: go to the other storeroom and get token
5: if your knowledge about safe storerooms increased, execute **R4**

Suspend & Wake-Up

Recall that an agent A performs SUSPEND when it finds that the unexplored links it found were all marked *dangerous*. In this case, A knows that at least one of these links does not lead to BH; thus, eventually some agent finishes a *cautious step* and performs a WAKE-UP.

The basic idea is to put the token in H to signal "I want to be woken-up", and check whether progress has been made before the token was put down (this is done to prevent deadlock, since an agent performing

WAKE-UP after removing its token from a *dangerous* edge might have arrived to H before the token was put there); if not, then the agent waits until the token disappears. An agent performing WAKE-UP simply moves a token from H (if there is any) to its primary storeroom.

Algorithm 6 SUSPEND

```
 1: go to H, wait until it is empty and put a token there;
 2: scan your primary storeroom and return to H;
 3: if H is empty then
 4:     GRAB-TOKEN
 5: else if storeroom is empty then
 6:     traverse the local map;
 7:     if traversal revealed progress then
 8:         GRAB-TOKEN
 9:     else
10:         wait until either H becomes empty or a token from link #1 disappears;
    (in the latter case execute R4 now)
11:         GRAB-TOKEN
12:     end if
13: else // found a token in storeroom, and there is a token in H
14:     get token;
15:     go to the storeroom that contained a token, and wait until it becomes empty;
16: end if
```

Problems may arise because several agents might be executing SUSPEND, WAKE-UP and VERIFY simultaneously, and because the agents do not necessarily agree on the primary storeroom. Dealing with this fact constitutes the most technical part of the algorithm. The idea is to wait until any activity going on (detected by a non-empty H or storeroom) looks to have finished and then restart SUSPEND. Still, there are many possible cases of how the agents can steal each other's tokens and/or misinterpret what is going on. The reasons behind the design of SUSPEND and WAKE-UP will become fully apparent only when reading the formal proofs in the next section.

The idea of WAKE-UP is to wake-up an agent suspended at H by moving its token to a storeroom. In order to make GRAB-TOKEN work, the waking-up agent first places its token in the storeroom and then removes the token from H. If H is empty or there is a token in the storeroom, WAKE-UP does nothing, because either there is nobody suspended, or it has been already waken-up and just has to pick up its token. When an agent suspended at H sees that its token has disappeared, it will search around and find its token (using GRAB-TOKEN).

4 Correctness and Complexity

Let us call an agent *informed* if its knowledge about which storerooms are safe is correct. If BH is one of the storerooms, all agents (that have finished initialization) are informed; otherwise, an informed agent is one that knows that both storerooms are safe. The notion of an informed agent is for discussion purposes: the agents themselves may not know whether they are informed or not.

The overall structure of the correctness proof is the following: we first prove that, during the whole algorithm, at most Δ agents enter BH, and all agents that are alive make progress by eventually exploring

Algorithm 7 WAKE-UP

```
1: go to H; if empty, abort;
2: go to primary SR;
3: if there is a token in SR then
4:     abort;
5: else
6:     put token;
7:     go to H;
8:     GRAB-TOKEN
9: end if
```

a new edge. Second, we prove that all informed agents maintain a correct local map, i.e. the local map of an informed agent is at any time isomorphic to some subgraph of the network (including port labels).

The above arguments are formally carried out through a sequence of Claims and Lemmas, which will lead to the main Theorem:

Theorem 1 (Main Theorem) *At most Δ agents die. The remaining ones terminate with a correct map.*

Let us start with some basic observations. Since a token is put in a node only in SUSPEND, WAKE-UP or VERIFY, we get:

Claim 2 *A token is in the node v only if v is* H *or a storeroom.*

The most technical part of the algorithm is the implementation of the communication between agents by means of tokens. We are specifically interested in agents who have put their token in H or in a storeroom, and are now without a token; we will call them *empty-handed* to distinguish them from agents who do not have a token because they are performing a cautious step. From the definition of *cautious step*, from Claim 2, and by construction we get:

Claim 3 *There are as many empty-handed agents as tokens in* H *and storerooms. Moreover, a token from a node is taken only by an empty-handed agent.*

By construction, an agent may take a token from a storeroom only when performing GRAB-TOKEN, VERIFY, or **R4**. Hence, we have

Claim 4 *If an agent takes a token from a storeroom, it has just arrived there from* H.

An agent performing procedure GRAB-TOKEN visits H and possibly some storerooms a constant number of times in search for a token. The algorithm assumes that a token is found, so for the correctness of the algorithm it is important to prove that is is indeed so. Moreover, we prove that an agent never enters BH during GRAB-TOKEN:

Lemma 1 *An agent always successfully gets a token in procedure* GRAB-TOKEN.

Proof: Consider, for the sake of contradiction, an agent A performing GRAB-TOKEN that does not find a token. Let t_0 be the time when A leaves its primary storeroom (call it SR x) empty on line 2, and starts travelling to H. Let t_1 be the time when A executes line 3, i.e. A arrives to H, sees it empty, and

(atomically) leaves to SR y. Finally, let t_2 be the time when A arrives to SR y and either dies there (if it is BH) or finds it empty.

For any time t, $t_0 \leq t \leq t_2$, let us say that the vertices still to be visited by A are *ahead of* A, and those that A will not visit are *behind* A. Since SR x is empty at t_0, from Claim 3 (and the fact that there are no tokens in BH) it follows that there is at least one token in a node ahead of A at t_0. However, when A arrives to SR y at t_2, there is no token ahead of A.

We say that an agent B *jumps over* A, if there are times t' and t'', $t_0 < t' < t'' < t_2$, such that B puts a token at t' behind A, thus becoming empty-handed, and the next time B takes a token is at t'' from ahead of A. First we show that no agent B jumps over A. By contradiction, let B jump over A. Then $t' > t_1$: indeed if $t' < t_1$ then B puts the token at t' in SR x which is the only node behind A at t'. However, due to **R2**, B would start traversing the edge to H at t'. Due to FIFO, A arrives to H first at t_1, and immediately leaves on the link to SR y; then at some time after t_1, B arrives to H, and due to FIFO and Claim 4 B cannot take a token from SR before time t_2. So indeed $t' > t_1$, and B puts a token in H or SR x at t' when A is already traversing to SR y. Due to FIFO and Claim 4, B cannot take a token from SR y (the only place ahead of A at t') before time t_2. So no agent B jumps over A.

Let us now consider the situation at t_0, when A starts going to H. There are $c \in \{1, 2\}$ empty-handed agents at t_0, and there is always at least one token ahead of A: at t_0 there are c tokens ahead of A (note that SR x is empty at t_0), at most $c - 1$ may be taken by the possibly other empty-handed agent, and there are no jumps. Hence, A will find a token. □

We now introduce the concept of token ownership: An agent A holding a token T *owns* that token. This ownership continues after the agent puts the token T in a node until one of the following occurs:

- *Legal swap:* An agent B takes T from H. Let T' be B's token at that moment, located at SR x. Then T becomes B's token and T' becomes A's token. We say that A's token has been *kicked out* from H to SR x.
- *Steal:* When an agent B takes T from a storeroom and becomes its new owner, we say that A's token has been *stolen* by B; B's token T' now becomes A's token.

A simple observation that is useful later in the proof is the following.

Lemma 2 *If there is only one safe storeroom, no token is ever stolen.*

Proof: Let us suppose, for the sake of contradiction, that the first time the statement of the lemma is violated is at t_1 when A's token is taken from the storeroom by an empty-handed agent B. Due to Claim 3, B's token is in H at t_1. Let t_0 be the last time before t_1 when B put a token (i.e. at t_0, B put a token, then remained empty-handed until t_1 when it took A's token from the storeroom). Since there are no steals from SR before t_1, B put its token in H, and the token remained there until t_1 (if B's token resided in the storeroom sometimes between t_0 and t_1, the only way A's token could be in the storeroom at t_1 is if B's token was either being stolen, or removed by B, none of which could happen). However, since B put a token in H at t_0, it must have been performing SUSPEND at that time (the only place where a token is put in H), and by the construction of Algorithm 6, B would take its token from H before t_1. □

The following Lemmas will be used in subsequent parts of the proof of the mail theorem.

Lemma 3 *A token from* SR1 *is never stolen. Moreover, let A be an agent knowing that* SR1 *is safe that puts a token in* H. *Then A's token will not be kicked out to* SR2.

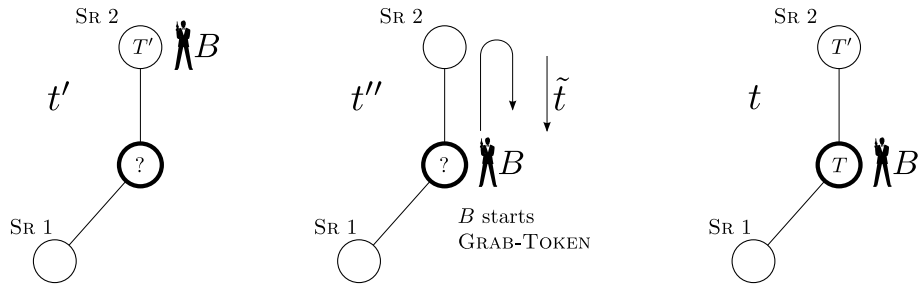

Figure 2: Illustration of the proof of the first part of Lemma 3.

Proof: Consider, for the sake of contradiction, the first time t when the conditions of the lemma do not hold. We distinguish two cases: either there was a steal from Sr1 at t, and/or a token of some agent A knowing about safe Sr1 was kicked to Sr2 at t.

Assume first that A's token T has been kicked out to Sr2, and let agent B be the one that did it (see Figure 2). Let t be the time when B grabbed A's token from H. From the definition of kicking-out we get that B's token T' was in Sr2 at t. Let $t' < t$ be the last time before t when B put its token in a node (i.e. B put T' at t', visited some vertices, and its next action involving tokens was to take T from H at t). Then, at time t', B did not know that Sr1 was safe. Indeed, if B had known that Sr1 was safe at t', it would never put its token in Sr2. Moreover, it cannot be that T' was kicked-out to Sr2 or stolen from Sr1 to Sr2, as it would violate the assumption of the lemma (B knows about Sr1), and we supposed that t is the time of the first such violation. However, we claimed that B's token is in Sr2 at t: A contradiction.

On the other hand, at time t, B knows that Sr1 is safe: Since A knows about Sr1, the link from H to Sr1 was free when A put T in H (from rule **R2**), and because of rule **R3** it always remained free until T disappeared from H. Therefore, when B took A's token from H at time t, the link to Sr1 was free and (at least at that time) B learned that Sr1 is safe.

This means that at some time t'', $t' < t'' \leq t$, B learned about Sr1. According to rule **R4**, at t'', B first goes to its old primary storeroom (i.e. Sr2) to look for a token. However, since the next action of B involving tokens after t' is to take T from H, B does not find a token in Sr2, returns to H, and starts Grab-Token. Due to Lemma 1, B finds a token during Grab-Token, and due to the assumption it must happen in H at t. Let $\tilde{t}$ be last time before t when B starts travelling from Sr2 to H (to find T there at t). Since Sr2 is empty at $\tilde{t}$, and due to Claim 3, T' is in Sr1 or H. Since T' is in Sr2 at t there must be some agent (different from B) who put it there after $\tilde{t}$. However, from the definition of ownership, it follows that T' is not B's token at t.

Assume now that the first action to violate this lemma is a steal from Sr1. Let agent A's token be stolen from Sr1 by an agent B at time t, and let t' be the last time before t when B put a token. So the scenario is the following: at t', B put a token somewhere thus becoming empty-handed, and never took a token until t. At t, B's token is in Sr2 or H, and B takes from Sr1 the token that belongs to the empty-handed agent A. Obviously, since there are no steals from Sr1 before t, B put at t' its token in Sr2 or H. Moreover, for the same reason, B's token has never been in Sr1 between t' and t: if it were so, the only way for A's token to be in Sr1 at t is if B took a token from Sr1 between t' and t, or there was a steal from Sr1 before t, none of which could happen.

First let us argue what part of the algorithm B performs at t. A token from Sr1 can be taken only in

VERIFY or GRAB-TOKEN; however, if B was performing VERIFY at t, it would had put its token to SR1 at t' (VERIFY requires to take the token from the same place where it was put). Hence, B is performing GRAB-TOKEN at t. We distinguish two cases:

(Case 1) B *knows about* SR1 *at* t'. In this case B puts a token in H at t': it must be SR2 or H, and from the construction it follows that no agent that knows about safe SR1 puts a token in SR2. However, since there are no kick-outs to SR2 for informed agents before t, B finds a token in H while performing the GRAB-TOKEN before t, and grabs it – a contradiction since B does not grab a token between t' and t.

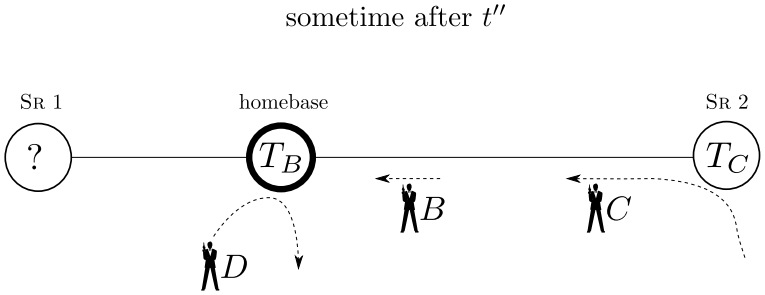

Figure 3: Illustration of the proof of the second part of Lemma 3.

(Case 2) B *does not know about* SR1 *at* t'. In this case B may or may not realize that SR1 is safe between t' and t; if it does, it invokes the rule **R4**, but since we suppose that B does not pick a token between t' and t, in any case the situation before t is such that B, at some time t'' leaves SR2 (as part of the final GRAB-TOKEN before t) which is empty, goes to H, finds it empty, continues to SR1, and takes A's token from there at t. Since B's token is always in SR2 or H, and B finds H empty upon arrival, there must be an agent C that[3] puts a token in SR2 after B left but before it reached H (see Figure 3), and some agent D that takes that token from H before B gets there. However, as soon as C puts the token in SR2, it starts going to H due to rule **R2**, and cannot be in H before B. Hence, when B arrives to H, its token is not there, and is not in SR2 either, since C was the last agent to put its token there, and C is still travelling (the only way the ownership of C's token can be transformed to B is if C takes B's token). □

Lemma 4 *A token put in* SR x *by an informed agent* A *executing* VERIFY *can be removed from* x *only by* A*.*

Proof: An agent can be informed because either there is only one safe storeroom, or it knows that both storerooms are safe. Consider first the case of only one safe SR x. Let t be the first time when some agent B takes from the storeroom a token put there by some agent A performing VERIFY at $t_0 < t$. Hence, at t, A is still empty-handed. Clearly, B's token is in H at t, and A and B are the only empty-handed agents. Moreover, since all agents are informed, and at t the lemma is violated for the first time, B had to put its token in H when about to become empty-handed before t. From that follows that B cannot be performing VERIFY, since during VERIFY an agent puts a token only in a storeroom. Therefore, B executes GRAB-TOKEN at t. However, in such case, B's execution of GRAB-TOKEN will find the token in H and grab that one, not going to the storeroom.

[3]There could possibly be more than one agent satisfying these conditions (if the token is taken in between); in this case, C is the last such agent.

If both storerooms are safe and A knows about that, it is using SR1 and by the first part of the statement of Lemma 3, its token will never be stolen from there. □

For the correctness of the algorithm is important to prove the following:

Lemma 5 *An informed agent A always finds a token in its primary storeroom on line 20 of Algorithm 4.*

Proof: In performing VERIFY, an agent puts a token in its primary SR on line 6. The statement of the lemma is then a simple consequence of Lemma 4. □

Note that

Lemma 6 *The event that a token is removed from some link happens less than $4NM(N-1)(\Delta+1)$ times.*

Proof: A token is put on and removed from a link only during *cautious step*. Cautious steps are performed only in EXPLORE on line 1 (which is executed only once) and on line 13, which is executed less then than $(2N-1)(N-1)$ times (at most $2N-2$ iterations of the inner loop, for at most $N-1$ candidate vertices). EXPLORE is called by each of the $\Delta+1$ agents at most M times. Finally, each agent might reset the algorithm once, applying rule **R4**. □

We can now prove Property 1, originally stated in Section 3.2.

Lemma 7 (= Property 1) *Let $K := (4N+1)M(N-1)(\Delta+1)$. The event that some token is removed from some link occurs less than K times. Moreover, let us consider a* SR *x. The event that a token is put in* SR *x occurs less than K times.*

Proof: An agent performing VERIFY can put its token only to its primary SR on line 6 of Algorithm 4, i.e. once for each call. There are $\Delta+1$ agents, each of them calls at most M times EXPLORE with a given primary storeroom, and each of the calls can result in at most $N-1$ calls to VERIFY. Hence, it might have happened at most $(\Delta+1)M(N-1)$ times that the token to SR x was put by an agent performing VERIFY.

The only other place when a token is put in a SR is in WAKE-UP. However, WAKE-UP is called only after an agent finishes a *cautious step*, and so by Lemma 6 it may happen less than $4NM(N-1)(\Delta+1)$ times. □

We now aim at proving that at most Δ agents disappear in BH. In order to do so we need to show first that an agent can enter a BH only during a cautious step, i.e. that edges marked *safe* in the local map of an agent correspond to safe edges in the network.

Lemma 8 *When an agent A adds a node v to its local map as a new node , then the local map indeed did not contain v.*

Proof: A node v is added as new only if the test $w = v$ in EXPLORE failed for every candidate w. We show that if the test fails then indeed $w \neq v$. The test for a given w can fail only in three cases:

(*Case 1.*) By having the port γ still *dangerous* after executing the loop on lines $5 \ldots 9$ of Algorithm 3 for K times. However, if $w = v$, then between each iteration of that loop the port γ is cleared which is a contradiction with Lemma 7. Hence, $w \neq v$.

(*Case 2.*) By noticing (in line 14) difference between what the map tells what should be seen if $v = w$ and what really is visible. Clearly, in such case $v \neq w$.

(*Case 3.*) By having VERIFY return false. VERIFY returns false if the agent A has not found the token in the node s (which is equal to its correct SR x if $v = w$) for at least K times. Note that A always leaves SR x with its token there. Consider, for the sake of contradiction, that $v = w$, and A visited $p =$SR x at least K times, always seeing s empty. Moreover, each time it returned back to SR x, it contained a token. Hence, during each loop, the token must have been removed from SR x, and (at least one) token put back in SR x. However, due to Lemma 7, this happens less than K times. □

Using the previous lemma, we can argue that an agent disappears in a BH only during a cautious step:

Lemma 9 *If A enters* BH*, the link e upon which it arrived is marked by its token.*

Proof: The statement of the lemma clearly holds if A used cautious step to cross e; thus, we only have to prove that whenever A does not use cautious step, it is crossing an edge not going to BH. The edges traversed in EXPLORE and VERIFY without cautious step are the edges of β that have already been proven safe by traversing them cautiously on l. 13 of EXPLORE. The traversal for finding new unexplored edges uses only *safe* edges, which according to Lemma 8 are mapped correctly and do not lead to BH. By Lemma 1, an agent never dies in GRAB-TOKEN. □

Since no agent dies in GRAB-TOKEN, no agent enters a link marked by a token elsewhere, and the degree of the BH is at most Δ, we get:

Corollary 1 *At most Δ agents die.*

From Lemma 8 it follows that the links marked as *safe* are always correctly identified. Now we prove that the only way a *quasi-safe* link may not be correct is that the agent is not informed (i.e. the token put in the, unknown to the agent, second SR interfered with its verification process).

Lemma 10 *Each informed agent has a correct map.*

Proof: It follows from Lemma 8 that if an agent A adds a new node v to its map, then indeed v has not been in A's local map before (from the proof it also follows that the port numbers are indeed the correct ones). So it remains to be proven that if an informed agent A with primary SR being SR x adds to its map an edge (u, w) between two visited vertices, then there is an edge (u, w) in the graph. Adding an edge (u, w) requires that the hypothesis $v = w$ tested in EXPLORE and VERIFY returns TRUE, i.e. the agent saw a token in s at least K times. To conclude we show that if VERIFY returns TRUE then indeed $w = v$.

Consider for the sake of contradiction an informed agent A with primary SR x, and a situation where VERIFY returned TRUE but $v \neq w$. Due to Claim 1, if $v \neq w$, then $s \neq$ SR x. Hence, if $v \neq w$ then s is either H or SR y, since that are the only possible places where a token may be (and A saw a token in s many times). However, s cannot be SR y: if SR y is safe, A knows it (since A is informed), and checks SR y to be empty on line 12 in every iteration. Due to Lemma 7, it is not possible that a token appears in SR y in every iteration.

Therefore, s is H. Let t_0 be the time when A put its token in SR x. Due to Lemma 4 the token never disappears during the iterations of A's loop. Let us count how many times A can see a token in H. Consider an iteration of A in which A checks H on line 13 at time t_1. Let $t_3 > t_1$ be the time when A arrives to H and sees a token there. Let t_2 be the last time before t_3 when an agent put a token in H, and let this agent be C. Clearly, $t_2 > t_1$. First, let us count how many times C may be some agent that was not informed at t_2 (see Figure 4).

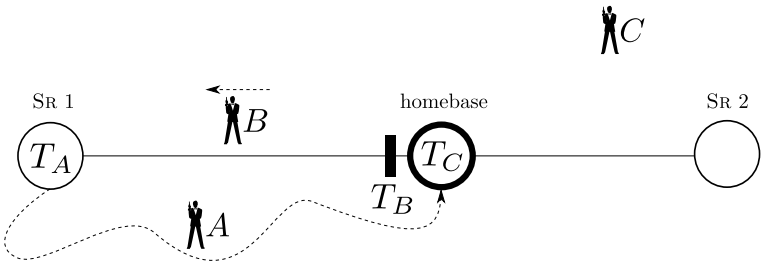

Figure 4: Illustration of Lemma 10: A starts its iteration from SR1, comes to s which is, by chance, H, and finds a token put there by some uninformed agent C.

The homebase was free at t_1, as was the port leading from H to SR1. (Note that $x = 1$ if there are supposed to be uninformed agents.) C put a token at t_2 and did not became informed, so there must be an agent B who started traversing the link to SR1 using cautious step between t_1 and t_2. However, in the next iteration, A will also traverse the link from H to SR1, and due to FIFO, B will have known by that time that SR1 is safe and never put a token on the link again. Hence, every iteration in which A sees a token of some uninformed agent in H can be charged a fresh agent B. This yields that there are at most Δ iterations in which A sees token of an uninformed agent.

Now let us count the number of iterations, except the last (that may only add 1 to the count), in which A sees a token of some informed agent C in H (i.e. C that is informed at t_2). Obviously, C is performing line 1 of SUSPEND at t_2 (that is the only place where a token is put in H), and starts traversing to SR1. We show that the token put in H by C remains there until C collects it. From that it follows that during one iteration of A, C goes to SR1, finds a token there, returns to H, and collects the token (since A waits for H to be free at the beginning of each iteration, and due to Lemma 4 its token remains in SR1). Moreover, C then goes to SR1, and waits for the token to be removed (line 15 of SUSPEND), so it cannot responsible for any other iteration where A sees a token in H.

Let $X \neq B$ be the first agent to take the token from H at some time after t_2. Note that after t_2, both SR1 and H contain A's and C's token, respectively, so when X is about to take the token from H, its own token is in SR2. However, this means that C's token gets kicked to SR2, a contradiction with Lemma 3. Putting it together, if s is H, there are at most $2\Delta + 1 < K$ iterations where A can see a token in s. □

When showing that livelock is not possible, the following lemmas are handy.

Lemma 11 *An agent spends $O(\Delta MN^3)$ moves in one call to* VERIFY.

Proof: The loop on line 11 of VERIFY has at most $2K = O(\Delta MN^2)$ iterations of $O(N)$ steps each, and it either terminates or the algorithm is restarted in **R4** after constant number of steps.

It remains to be shown that the loop on line 3 of VERIFY does not cost more. In one iteration of that loop A leaves empty storeroom, waits until H is free, and finds a token in storeroom, i.e. the number of iterations is bound by the number of times a token can be put in a storeroom, and the lemma follows from Lemma 7. □

Lemma 12 *The overall number of times an agent A executes* SUSPEND *is $O(\Delta^2 MN^2)$.*

Proof: Let us call an execution of SUSPEND by some agent A *active* if it got past line 1 of Algorithm 6 (i.e. the agent has already put its token). Each execution ends on one of the lines 4,8,11,15. Obviously,

for an execution to end on line 8, there must be a progress, which may happen at most $2M$ times. Similarly, if an agent A exits on line 15, there must be a time when A saw the respective SR full, and it was empty before exiting SUSPEND. Due to Lemma 7 this may happen at most K times per storeroom; this means that there are at most $O(\Delta MN^2)$ executions of SUSPEND ending on line 15.

In the rest of the proof we shall count the number of executions that end either on line 4 or line 11. In these cases, A either finds H empty at some point of time (maybe after waiting for a token from H to disappear), or realizes that a token from link #1 disappeared. However, the second condition may happen at most Δ times, so we may suppose that A finds an empty H. When A started the execution, it put a token in H on line 1 at some time t_1. Let t_2 be the first time after t_1 when the token from H was removed by some agent B. We argue about the part of the algorithm B was performing[4] at t_2. Due to Lemma 5 an agent performing VERIFY never takes a token from H, unless as part of rule **R4**. Overall, there are only Δ times when some agent may grab a token as part of **R4**. Using Lemma 7, there are at most K executions of WAKE-UP, each of them can remove at most one token from H. Summing it up so far, there are $O(\Delta MN^2)$ executions where A's token is taken by some agent not executing SUSPEND.

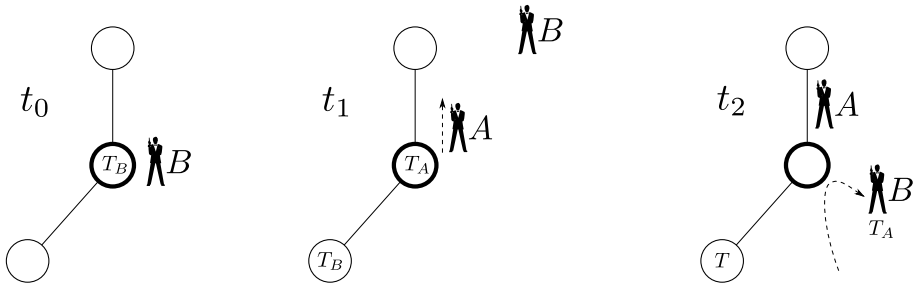

Figure 5: Illustration of Lemma 12: B puts its token in H when executing SUSPEND at t_0. At t_1, A puts its token in H while B's token is somewhere else. At t_2, B kicks A's token from H.

Next, note that between t_1 and t_2 no agent starts SUSPEND (no execution gets past the waiting on line 1), so if B is executing SUSPEND at t_2, it must have been performing the same execution of SUSPEND also at t_1; in particular, B is empty-handed in the interval between t_1 and t_2. Let this execution of B started at $t_0 < t_1$ by B putting its token in H on line 1 (see Figure 5); obviously, B did no become informed between t_0 and t_2 (otherwise it would be executing **R4** inbetween), so it either was informed at t_0 or is not informed at t_2.

Let us examine in which part of Algorithm 6 B's execution ends. Since it is B who takes A's token after t_1, B's execution does not pass the test on line 3 after t_1. Similarly, B will not find an empty H on line 10 after t_1. For a particular agent B, there is at most one execution ending by executing **R4**, at most $2M$ executions ending on line 8, and, as argued above, at most $O(\Delta MN^2)$ executions ending on line 15. Altogether, there are at most $O(\Delta^2 MN^2)$ executions of A where the token was removed by some agent B executing SUSPEND such that B was not executing GRAB-TOKEN from line 4 or 11 at t_1.

For the two remaining possibilities for B to take A's token: at t_1, B was already executing the GRAB-TOKEN from either line 4 or 11, we distinguish three cases:

(Case 1) *B is informed at t_0.* We prove that it is not possible that B is performing GRAB-TOKEN on line 4 or 11 at t_1. Let us suppose, for the sake of contradiction, that it is so; let t', $t_0 < t' < t_1$ be the time when the execution of GRAB-TOKEN started. B saw empty H at t', and started traversing to its primary

[4]We consider the executions of **R4**, VERIFY, SUSPEND, and WAKE-UP; the executions of GRAB-TOKEN are counted in the respective calling procedures.

storeroom. However, due to Lemma 3, B's token must have been kicked to its primary SR between t_0 and t', and by Lemmas 2, 3 is never stolen from there. So B would find its token there and take it before t_2. So B is not performing the final GRAB-TOKEN at t_1.

(Case 2) *B is not informed at t_0, and executes the* GRAB-TOKEN *on line 4 at t_1.* It means that B put a token in H at t_0, went to SR2, found a token there, returned to H, found it empty on line 3, went to SR2 as part of GRAB-TOKEN, found it empty, and finally returned to H to remove A's token at t_2. Due to Lemma 7, this may happen $O(\Delta MN^2)$ times per agent, and $O(\Delta^2 MN^2)$ times overall.

(Case 3) *B is not informed at t_0, and executes the* GRAB-TOKEN *on line 11 at t_1.* Then, as argued above, B is not informed at t_2, either, and from **R2, R3** it follows that A is not informed at t_1 (if A was informed at t_1, the rules would force that the link to SR1 is free at t_2). Hence, at some point of time

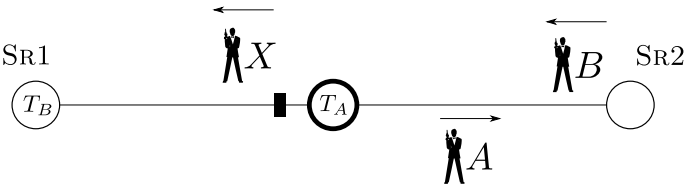

Figure 6: Critical situation for agent A w.r.t. agent X.

between t_1 and t_2, the situation is as seen on Figure 6: Both A, and some agent B are uninformed, there is an agent X that was performing a cautious step to SR1 at t_1, A is traversing the link to SR2 on line 2 of SUSPEND, A's token is in H (since it is taken by B at t_2), B is just leaving an empty SR2 while performing the GRAB-TOKEN on line 11 of SUSPEND. We call such situation *critical* for agent A w.r.t. agent X. We aim to show that for a fixed agent A, two critical situations w.r.t. the same agent X can not occur unless one of the following happens: a token is put in SR2, an uninformed agent gets informed, some agent calls WAKE-UP, or some agent increases its local map.

Let us suppose there is a critical situation at t' as in Figure 6, and that the above conditions hold. Then A and B are the only empty-handed agents at t', and the token from SR1 will not be removed while the conditions hold (no informed agent takes the token, and no uninformed agent goes to SR1 without being informed). For the sake of contradiction, let us further suppose that there is another critical situation of A w.r.t. X at time $t'' > t'$. Hence, at some time $t'_1 > t_2$, A put a token in H, and started to traverse to SR2, while some agent B' had already been performing the GRAB-TOKEN on line 11. That means that, at some time $t'_0 > t_2$, B' put a token in H. However, after t' no token has been put in SR2, and SR1 remains full. Hence, B''s token is still in H at t'_1 – a contradiction since at t'_1 it is not possible for A to put a token in H.

Any agent can make a cautious step from H to SR1 at most once; due to Lemma 7, there are at most $O(\Delta MN^2)$ times a token is put in SR2; each agent can get informed only once; due to Lemma 7, there are at most $O(\Delta MN^2)$ calls to WAKE-UP, and there are at most $O(\Delta M)$ times an agent increases its local map. Putting it together, there are at most $O(\Delta MN^2)$ critical situations for agent A w.r.t. agent X, and hence there are $O(\Delta^2 MN^2)$ critical situation of agent A in total. □

Lemma 13 *An agent performs $O(\Delta MN^5)$ steps in one call to* EXPLORE.

Proof: A call to EXPLORE consists of $O(N)$ iterations of the loop on line 2, each iteration involving $O(N)$ iterations of the for loop on line 4 having $O(\Delta MN^3)$ steps each, and one call of VERIFY. Summing up and applying Lemma 11 we get that the number of steps spent in EXPLORE is $O(\Delta MN^5)$. □

Now we are ready to prove that no livelock can occur:

Lemma 14 *An agent spends $O(\Delta M^2 N^5)$ steps executing Algorithm 1.*

Proof: Procedure INITIALIZE takes constant number of steps. Each iteration of the loop on line 2 takes $O(N)$ steps in the traversal on line 3. In at most $O(M)$ iterations, the loop ends by calling EXPLORE, which following Lemma 13 has $O(\Delta M N^5)$ steps. The remaining iterations end with a call to SUSPEND. Due to Lemma 12 there are at most $O(\Delta^2 M N^2)$ such iterations, each having at most $O(N)$ steps. Using the fact that $\Delta \leq N$ we get the result. □

We have shown so far that no livelock occurs, and that after $O(\Delta^2 M^2 N^5)$ steps the algorithm stops with at most Δ agents dead, and the remaining ones either terminated or deadlocked. The next lemmas are needed to show that no deadlock can occur, i.e. every agent is always able to continue its algorithm until termination. An agent may wait for a token to disappear either on a link, in H or in a storeroom.

Lemma 15 *Any port not leading to* BH *is eventually free.*

Proof: A token is put on a link only during a cautious step. If the link does not lead to a BH, any agent performing the cautions step eventually returns and removes the token. Moreover, due to Lemma 7, it may not be the case that the token is put on the link infinitely often. □

From that immediately follows:

Corollary 2 *An agent performing* **R2** *eventually finishes.*

Lemma 16 *Eventually, there is no token in* H.

Proof: Since the overall number of steps is finite, it is not possible that a token is put in H infinitely many times. Hence, the only way the lemma may not hold is if the token is placed in H at some time t_0, and never it disappears.

The only way a token can be put in H is in SUSPEND. Consider for the sake of contradiction that an agent A puts a token in H at time t_0 and that token never disappears. That means A went to its primary SR x and found it empty at time t_1, then returned and went to rescan. Since A cannot die during the scan (due to Lemma 8), and A never waits during the scan, either, A returns to H, and, since the token never disappears, starts waiting there.

First, we show that after t_1, SR x remains empty. To show this, note that at t_1, SR x is empty. Moreover, no agent performing VERIFY gets beyond line 4 of Algorithm 4 after t_0. Hence, the only way an agent B performing VERIFY could put a token in SR x after t_1 is that, at t_0, B was already traversing the link to SR x on line 4; however, B would then arrive to SR x before A, i.e. before t_1. The only other possibility that some agent B puts a token to SR x after t_1 is that B is performing WAKE-UP. However, B would then continue to H, and would remove the token from there – a contradiction. Hence, after t_1, SR x remains empty forever.

If A was not informed at t_0 (i.e. $x = 2$, and SR1 is safe, but link #1 is blocked in H), due to Lemma 15, the port #1 will eventually be free. In such a case, agent A, which is waiting in H, is eventually awaken (due to Lemma 14). Agent A would then start performing **R4**, going to SR2, it would find it empty (due to the arguments above), it would return to H, and take the token – a contradiction.

Hence, A was informed at t_0. Due to Lemma 10, A has a correct local map; that means that when A returns from the rescan on line 6 of SUSPEND, some of the links A saw blocked do not lead to H (due to the 2-connectivity). Eventually, the cautious step on one of them is finished, and the respective agent B

comes (without waiting) to H performing a WAKE-UP. B is not empty-handed, so regardless of whether it was already informed, or got informed upon arrival to H, it goes to SR x, and puts a token there – a contradiction. □

Lemma 17 *Eventually, there in no token in any storeroom.*

Proof: As in the previous lemma, the only way the token is in the SR infinitely many times is if it never disappears after some time t_0. Consider, for the sake of contradiction, an agent A that put a token in SR x at t_0, and the token is never removed. The ownership of the token may change in time, however, due to Lemma 14, eventually, there is an empty-handed agent B that owns the token in SR x, and B is either dead, or waits indefinitely somewhere. However, due to Lemma 9, empty-handed agents don't die, so B is waiting somewhere. An empty-handed agent may wait only in H or a storeroom, and due to Lemma 16, B must be waiting in a storeroom. An empty-handed agent waits in a SR only on line 12 of Algorithm 4. However, that means that B is informed, $x = 1$, and B is waiting for a token to disappear from SR2. Since the token from SR2 never disappears, we may apply similar arguments, and arrive to a conclusion that its owner cannot be waiting forever. □

Corollary 3 *An agent performing* **R3** *eventually finishes.*

Now we can prove the main theorem:

Proof of **Theorem 1***:* Due to Corollary 1, at most Δ agents die. Due to Lemma 14, all surviving agents either terminate or end waiting. Due to Lemmas 16, 17, and their corollaries, all surviving agents terminate. Due to Lemma 10, the local map of an informed agent is correct, hence to conclude the proof it is sufficient to note that uninformed agent cannot terminate. Indeed, an agent A is uninformed if both storerooms are safe, but A does not know it. Clearly, there are at most $N - 2$ vertices in A's local map (at least one storeroom, and BH are missing), so A cannot terminate. □

Corollary 4 *The black hole can be located using* $O(\Delta^2 M^2 N^5)$ *moves.*

5 Concluding Remarks and Open Questions

In this paper, we have shown that black hole search of an unknown graph can be done with the optimal number of asynchronous agents in a polynomial number of moves not only in the (extensively studied) powerful whiteboard model but also is the substantially weaker enhanced token model. The results hold even if both agents and the nodes are *anonymous*.

These results are the first that address the problem of exploration of a dangerous unknown graph using tokens. Our results indicate that, perhaps contrary to expectation, the enhanced token model is computationally as powerful as the whiteboard one with regards to black hole search, albeit the complexities are different.

The algorithm proving these results is unfortunately rather complex; furthermore, its complexity, although polynomial, is very high. Immediate open questions are whether these results can be obtained with simpler techniques, and whether the cost can be substantially reduced.

Still open is to understand whether the same computational results can be obtained in the classical token model; i.e., when a token can only be placed on a node. It has been recently shown that this is the case if the agents have a map of the network [24]. Investigations of the problems when the agents are without a map (the model considered here), indicate that a computational difference between pure

tokens and enhanced tokens indeed exists if only a constant number of pure tokens can be put on a node [2]. Further negative results for agents with pure tokens and without a map would also show the impact that knowledge of the map has on the complexity of exploration of dangerous graphs.

Acknowledgments
The authors would like to thank the anonymous referees; their helpful comments and suggestions have led to greatly clarify the arguments and simplify the presentation.

References

[1] 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.

[2] B. Balamohan, S. Dobrev, P. Flocchini, N. Santoro. Asynchronous black hole search in an unknown anonymous graph with $O(1)$ pebbles. Proceedings of *19th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), 2012.

[3] B. Balamohan, P. Flocchini, A. Miri, N. Santoro. Improving the optimal bounds for black hole search in rings. Proceedings of *18th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), pages 198-209, 2011.

[4] M. A. Bender, A. Fernández, D. Ron, A. Sahai, and S. P. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation*, 176(1):1–21, 2002.

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

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

[7] J. Cao and S. K. Das (Eds). *Mobile Agents in Networking and Distributed Computing*, Wiley, 2010.

[8] J. Chalopin, S. Das, A. Labourel, E. Markou. Black hole search with finite automata scattered in a synchronous torus. Proceedings of *25th International Symposium on Distributed Computing* (DISC), pages 432-446, 2011.

[9] J. Chalopin, S. Das, A. Labourel, E. Markou. Tight bounds for scattered black hole search in a ring. Proceedings of *18th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), pages 186-197, 2011.

[10] C. Cooper, R. Klasing, and T. Radzik. Searching for black-hole faults in a network using multiple agents. Proceedings of *10th International Conference on Principles of Distributed Systems* (OPODIS), pages 320-332, 2006.

[11] J. Czyzowicz, S. Dobrev, R. Královic, S. Miklík, and D. Pardubská. Black hole search in directed graphs. Proceedings of *16th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), pages 182–194, 2009.

[12] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71 (2-3): 229-242, 2006

[13] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability & Computing* 16: 595-619, 2007.

[14] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science* 385(1-3): 34-48, 2007.

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

[16] S. Dobrev, P. Flocchini, R. Královic, G. Prencipe, P. Ruzicka, and N. Santoro. Optimal search for a black hole in common interconnection networks. *Networks*, 47(2):61–71, 2006.

[17] S. Dobrev, P. Flocchini, R. Královic, and N. Santoro. Exploring an unknown graph to locate a black hole using tokens. Proceedings of *5th IFIP International Conference on Theoretical Computer Science* (TCS), pages 131–150, 2006.

[18] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocol. *Distributed Computing* 19 (1): 1-19, 2006.

[19] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica* 48: 67–90, 2007.

[20] S. Dobrev, P. Flocchini, and N. Santoro. Improved bounds for optimal black hole search in a network with a map. Proceedings of *10th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), pages 111–122, 2004.

[21] S. Dobrev, R. Královic, N. Santoro, and W. Shi. Black hole search in asynchronous rings using tokens. Proceedings of *6th International Conference on Algorithms and Complexity* (CIAC), pages 139-150, 2006.

[22] S. Dobrev, N. Santoro, and W. Shi. Using scattered mobile agents to locate a black hole in an unoriented ring with tokens. *International Journal on Foundations of Computer Science* 19(6):1355 1372, 2008.

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

[24] P. Flocchini, D. Ilcinkas, and N. Santoro. Ping Pong in dangerous graphs: optimal black hole search with pebbles. *Algorithmica*, 62(3-4):1006–1033, 2012.

[25] P. Flocchini, M. Kellett, P. Mason, N. Santoro. Map construction and exploration by mobile agents scattered in a dangerous network. Proceedings of *24th IEEE International Parallel and Distributed Processing Symposium* (IPDPS), pages 1-10, 2009

[26] P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48 (3): 166-177, 2006.

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

[28] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345 (2-3): 331-344, 2005.

[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] P. Glaus. Locating a black hole without the knowledge of incoming links. Proceedings of *5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks* (ALGOSENSOR), pages 128–138, 2009.

[31] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theoretical Computer Science* 384 (2-3): 201-221, 2007.

[32] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Approximation bounds for black hole search problems. *Networks*, 52(4): 216-226, 2008.

[33] A. Kosowski, A. Navarra, and C. M. Pinotti. Synchronous black hole search in directed graphs. *Theoretical Computer Science*, 412(41):5752–5759, 2011.

[34] R. Královic, S. Miklík. Periodic data retrieval problem in rings containing a malicious host. Proceedings of *17th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), pages 157-167, 2010.

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

[36] C. E. Shannon. Presentation of a maze-solving machine. Proceedings of *8th Conference of the Josiah Macy Jr. Foundation (Cybernetics)*, pages 173–180, 1951.

[37] W. Shi. Black hole search with tokens in interconnected networks. Proceedings of *11th International Symposium on Stabilization, Safety, and Security of Distributed Systems* (SSS), pages 670-682, 2009.