Exploring an Unknown Graph to Locate a Black Hole Using Tokens

Stefan Dobrev[1], Paola Flocchini[1], Rastislav Královič[1,2⋆], and Nicola Santoro[3]

[1] SITE, University of Ottawa, {sdobrev,flocchin}@site.uottawa.ca
[2] Dept. of Computer Science, Comenius University, kralovic@dcs.fmph.uniba.sk
[3] School of Computer Science, Carleton University, santoro@scs.carleton.ca

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. The existing investigations have all assumed that the network is *safe* for the agents, and 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 assuming powerful inter-agent communication mechanisms: *whiteboards* at all nodes. Indeed, in this model, the black hole search problem can be solved with a minimal team size and performing a polynomial number of moves.
In this paper, we consider a less powerful *token* model. We constructively prove that the black hole search problem can be solved also in this model; furthermore, this can be done using a minimal team 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 [19]. It requires the agents, starting from the same node, to visit within finite time all

⋆ Partially supported by grant VEGA 1/3106/06.

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 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, 2, 3, 4, 6, 7, 13, 14, 15, 18]). Notice that, except for trees, the exploration with stop of anonymous graphs is possible only if the agents are allowed to mark the nodes in some way; various methods of marking nodes have been used by different authors ranging from the weak model of *tokens* to the most powerful model of *whiteboards*.

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) has become a pressing security concern (e.g., see [17, 20]).

Recently the exploration problem has been examined also when the network is unsafe [5, 8, 9, 10, 11, 16]. 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 the team of agents to be able to explore the network and, within finite time, discover the location of the 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 the 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 [9]; furthermore, if the graph is unknown, at least $\Delta + 1$ agents are needed, where Δ is the maximum node degree in the graph [10]. 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 [10]. 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 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 em-

ployed e.g. in [6, 8, 9, 10, 11, 13, 14]. In the whiteboard models, the black hole search problem can be solved with a minimal team size and performing a polynomial number of moves (e.g., [8, 9, 10, 11]).

The problem of exploring a dangerous graph has never been investigated in the less powerful *token* model, which is instead commonly employed in the exploration of safe graphs. In the classical 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 our variation (*enhanced token model*) we allow tokens to 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 our variation is not as weak; in fact, it could be implemented by having a $\log d$-whiteboard on a node with degree d.

The principal question targeted by our research was 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 enhanced token model. 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? at what costs? In this paper we provide definite answer to these questions.

1.2 Our Results

In this paper we present an algorithm that works in the token model and solves the BHS problem with the minimal 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 (*home-base*). 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).

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. 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 rather complex.

This work is the first that addresses the problem of exploration of a dangerous unknown graph using tokens. Our results indicate that, perhaps contrary to expectation, our variation of the token model is computationally as powerful as the whiteboard one with regards to black hole search.

topology	communication	# of agents	# of moves
arbitrary, unknown	whiteboard	$\Delta + 1$	$\Theta(N^2)$
arbitrary, known	whiteboard	$\Delta + 1$	$\Theta(N \log N)$
arbitrary, unknown	**tokens**	$\Delta + 1$	$O(\Delta^2 M^2 N^7)$

Fig. 1. Existing and new results for the BHS problem.

1.3 Related Work

The research on safe *exploration* of unknown graphs was started in 1951 by Shannon [19]. Most of the work since has been concentrated on exploration by a single agent (e.g., [2, 7, 18]). Safe explorations by *multiple* agents were initially studied for a team of 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 [3], whereas algorithms for exploration by more agents were given by Frederickson et al. for arbitrary graphs [15], by Averbakh and Berman for weighted trees [1], and more recently by Fraigniaud *et al.* for trees [13]. To explore arbitrary anonymous graphs, various methods of marking nodes have been used by different authors. Bender *et al.* [2] 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 of the graph is known and using $\Theta(\log \log N)$ tokens, otherwise. Dudek *et al.* [12] used a set of distinct markers to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [3] 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.* [13, 14], marking is achieved by accessing whiteboards located at nodes, and their strategy explores directed graphs and trees.

The explorations of unsafe graphs are quite recent and have focused mostly on asynchronous environments. The BHS problem has been studied when the network is an anonymous ring, characterizing the limits and determining optimal solutions [9]. When the network is an arbitrary graph the problem has been investigated in [10], 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. 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 the BH) [11]. In the case of specific graphs, including many important interconnection networks, the number of moves can be reduced to linear [8].

In all these investigations, 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.

In the synchronous environments, the investigations have produced optimal solutions for trees [5]; approximation results have been obtained for arbitrary graphs in [5, 16].

2 The Model

The network $G = (V, E)$ is a simple undirected graph with node-connectivity two or higher; 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 $\#1, \#2, \ldots, \#d(x)$.

Operating in G is a team of $\Delta + 1$ anonymous agents. The agents know the number of nodes of the network, can move from node to a neighboring node in G, have computing capabilities and limited amount of memory ($O(M \log N)$ bits suffice for our algorithm). We also assume that agents know the degree Δ of the BH.

Each agent has a token that can be placed on 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 *home-base* (home-base).

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):
- change the state (local memory of A)
- remove (or place) at most one token from v or an incident link and
- 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. 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 waiting (for a token to disappear) and traversing a link can take 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 home-base, the current position of the agent, and the location of the black hole, are indicated.

Note that termination with an exact map in finite time is actually impossible. In fact, since an agent is destroyed upon arriving to the BH, no surviving agent can discover the port numbers of the black hole. Hence, the map will have to miss such an information. More importantly, the agents are asynchronous and do not know the actual degree $d(\text{BH})$ of the black hole (just that it is at most Δ). Hence, if an agent has a local map that contains $N-1$ vertices and at most Δ unexplored edges, it cannot distinguish between the case when all unexplored ports lead to the black hole, and the case when some of them are connected to each other; this ambiguity can not be resolved in finite time nor without the agents being destroyed. In other words, if we require termination within finite time, an agent might incorrectly label some links as incident to the BH; however the 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 BH or explores at least $N-1$ vertices and $M-\Delta$ edges.

In the beginning, the local map of each agent contains only the home-base. 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 - in the local map; connecting two already explored vertices, but could be wrong

Throughout the execution, whenever an agent leaves via a port that might lead to the 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.

Initially, all ports incident to the home-base are *unexplored*. The local map of an agent is constructed by adding edges in a sequential manner according to Algorithm 1: The searching for an unexplored port is straightforward: any

Algorithm 1 General algorithm of an agent

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

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[2], and updates the local map. Due to 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 agent is able to recognize this uncertainty, and will add this edge to the local map as *quasi-safe* instead of *safe*.

Eventually, no unexplored port is found. If $N-1$ nodes has been visited, the remaining node is the BH and the algorithm can terminate. Otherwise, the access to the unexplored part of the graph is blocked by *dangerous* ports. Since G is two-connected, at least one of those ports does not lead to the BH and the token will eventually be removed from it, making it *unexplored*. In order to avoid live-lock, 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 the home-base, 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-

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

up. Complementarily, whenever an agent removes its token from an edge, it goes to the home-base 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 following three rules clarify some terms used in the description:

R1 "cautious step" in a vertex v over a 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 the home-base" $\equiv$ wait for all known safe links incident to home-base to become unmarked, then put the token

R3 "put token on a link" (in vertex v) $\equiv$ wait for v to become empty, then put the token

The nodes on the other ends of the links #1 and #2 from home-base are called **storerooms** (SR) and they play 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).

Each agent starts the algorithm by exploring (using cautious step) SR1 and SR2 from the home-base (in this order). Since the graph is simple, at least one of them is safe; if both these links are *dangerous*, the agent will simply wait until one of the blocking tokens disappears. Eventually, each agent will know about one or two safe storerooms. The primary storeroom for an agent is defined as the storeroom known to be safe with the lower numbered link leading to it.

Note that if the BH is located in one of the storerooms, all surviving agents will choose the other SR as their primary SR. However, if none of the SR's contains the BH, there might be agents with different primary SR's (some might find SR1 safe and choose it, some might find it temporarily dangerous and select SR2).

As this might lead to problems, the algorithm tries to remedy this situation by "updating" the primary store room of agents that had originally selected SR2 and later discover that SR1 has become safe in the meanwhile. The update rule is called **R4** and will be described later.

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 vertex v) to the local map. Agent A starts executing a cautious step over the edge e and, if survives, it proceeds with determining whether or not e leads to a new (not in the local map) vertex.

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 home-base. 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. If the agents were able to recognize the home-base, 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 the home-base. Starting from v, A follows[3] the port labels specified by α. If A finishes in the home-base, then $v = w$, otherwise A makes another guess. If all guesses fail, v is a new node. However, in our model the agents can not recognize their tokens nor the home-base. 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.

Let β_w (we will use β when w is clear from the context) be a sequence of port labels starting with the label of the port from u to v and then following a path (using only edges marked as *safe* in the agent's map) from w through the primary SR and ending in u. Clearly, if $v = w$ then β specifies a simple cycle in the graph (and therefore $|\beta| \leq n$, even if actually $v \neq w$).

Agent A verifies whether $v = w$ by following the labels specified by a cyclic repeating of β (we will call it β^*) for up to N^2 edges or until A finds a difference between what it sees in the current node and what it should see (according to its map) if $v = w$. The number of steps is chosen large enough so that following β^* creates a cycle even if $v \neq w$ (as we will see later, using only β is not enough). This means (as will be proven later) that if no discrepancy has been found for N^2 steps, u and v indeed lie on a cycle C passing through the correct SR, with the labels specified by β^*. Unfortunately, it is still possible that, although no discrepancy is found, $v \neq w$: this could happen if $|C|$ is a multiple of $|\beta|$. In this case the agent verifies whether $v = w$ or not in the procedure VERIFY, which will be described later.

The N^2 steps along β^* must be done in cautious manner, not entering *dangerous* ports, since it may be the case that $v \neq w$ and β^* leads to the 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 *dangerous*. If this happens, the agent cannot afford to wait in w' until the token is removed, because this edge might indeed lead to the BH. Instead, it wants to ensure that, if $v = w$ then the token will be removed allowing A to continue its cautious walk through λ. To do so, 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''. 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

[3] cautious walk needs to be used, as v might be different from w, and α from v might lead to the BH

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. Since (as we will see later) during the whole execution of the algorithm a token is placed on a given port less then $2\Delta MN^3$ times, if after $2\Delta MN^3$ cleaning tries λ is still blocked, then $v \neq w$.

The Algorithm 2 describes the procedure EXPLORE in full detail.

Algorithm 2 Exploring an edge with label l_1 by EXPLORE

```
do a cautious step over link l_1, let l_2 := label of the link upon which you arrived
for all w in local map do
    compute the sequence β
    for N^2 steps do
        while next port γ in β* is dangerous and this loop has been executed less
            than 2ΔMN^3 times do
            go back |β| steps
            wait until there is no token on the edge along which you arrived
            go forwards |β| steps
        end while
        if port γ is still dangerous then
            backtrack your steps to v and continue the outermost for cycle for the
                next w
        end if
        do a cautious step
        if what you see in the vertex you arrived to is not compatible with the local
            map assuming v = w then
            backtrack your steps to v and continue the outermost for cycle for
                then next w
        end if
    end for
    if VERIFY then  // after traversing N^2 edges there was no discrepancy, so
        I am in a cycle. Is it a short one?
        add edge to w to the local map as quasi-safe
        exit from EXPLORE
    end if
end for
add to the local map the new vertex and edge; the added edge is marked as safe
```

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 home-base and the storerooms. In other words, the home-base and the storerooms are employed to accomplish different tasks and this requires much care to avoid ambiguity and interference between different activities.

Verification

The test of a candidate vertex w in the procedure EXPLORE may end, after traversing the sequence β^* for N^2 steps, in a situation where the agent knows that either β or its multiple forms a safe cycle connecting u and v. The procedure VERIFY is used to verify whether the cycle consists of just one repetition of β (in which case $v = w$).

Algorithm 3 VERIFY – let p be the SR, if the hypothesis about w is true

```
1: PosCount = NegCount = 0
2: loop
3:     go to home-base, wait until it becomes empty, and go to the primary SR
4:     if the SR is empty then
5:         put token and exit loop
6:     else
7:         wait until the SR becomes empty
8:     end if
9: end loop
10: while PosCount < 2ΔMN^3 + ΔMN and NegCount < 2ΔMN^3 + ΔMN do
11:     if known, go to the other SR and wait until it becomes empty
12:     go to the home-base, wait until it becomes empty
13:     go to p
14:     if there is a token then
15:         PosCount = PosCount + 1
16:     else
17:         NegCount = NegCount + 1
18:     end if
19:     go to the primary SR and if empty update the knowledge of storerooms using
            rule R4 and restart algorithm
20: end while
21: take token
22: if PosCount ≥ 2ΔMN^3 + ΔMN then
23:     return TRUE
24: else
25:     return FALSE
26: end if
```

The idea of VERIFY is to use a token in the primary SR for breaking symmetry on the β^*-cycle. An agent A performing a VERIFY first makes sure that it is not interfering with any other agent by waiting until both the home-base and the SR's it knows to be safe are empty. It then puts its token in the primary SR and walks[4] along the β^*-cycle for $|\beta|$ steps to a vertex w' and checks whether there is a token in w'. The idea is that if $v = w$ then w' is the SR and contains the token, if $v \neq w$ then w' should be empty as it is not the correct SR.

[4] Note that it is not needed to use cautious steps, as the cycle identified by β^* has already been traversed and is known to be safe

Notice that a straightforward check on whether there is a token in w' can fail for two reasons. (1) It may happen that w' is not a SR but, say, the home-base. As mentioned above, the home-base 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 home-base) while A traveled to w', A is deceived since it finds a token in w', but this is not the token it left in SR! (2) It may happen that w' is indeed a SR but some other agent took the token from the SR in the meanwhile (when finishing SUSPEND); so A is again deceived because it does not find its own token.

Luckily, as will be shown later, each of these two cases occurs less than K times, where $K = 2\Delta MN^3 + \Delta MN$. Hence, if A saw a token in w' at least K times, then w' must be the SR; conversely, if A saw no token in w' at least K times, then w' is not the SR.

One last complication comes from the fact that, at the beginning of each iteration of the while cycle, A has to make sure that the home-base and the SR's are empty. The problem is that agents cannot always agree on one primary SR. In fact, (if the BH is not in a SR) there are three types of agents: some think that only SR1 is safe, other thing that only SR2 is safe, while the third group knows that both SR's are safe. However, if an agent does not know that both SR's are safe, it cannot make sure that both of them are empty. In this case it may happen that the result of VERIFY is wrong. 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. Note that if EXPLORE declares v to be a new vertex it never errs, 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 SR on line 19 is if A does not know about (safe) SR 1. This means that after seeing an empty SR, A can update its knowledge about the storerooms and reset the algorithm according to rule R4.

R4 $\equiv$ When an agent first realizes that both SR's are safe, it performs the following actions:
- If you have no token and your old primary SR is SR2, execute GRAB-TOKEN starting from SR2, else execute GRAB-TOKEN from the home-base.
- Update the knowledge about SR's.
- If you came to the home-base to perform WAKE-UP but have not done so, do it now
- Restart the whole algorithm

Grab-Token

The procedure GRAB-TOKEN is used by an agent A to pick up a token that it has previously put at the home-base or a SR. It might happen that some other agent B has meanwhile picked the A's token instead of its own. However, in such case B's token must be somewhere around (in the home-base or in a

SR) and A will take it (or the token of yet another agent).

Algorithm 4 GRAB-TOKEN – starts in home-base

1: if there is a token in home-base, get it and exit GRAB-TOKEN
2: go to primary SR, if there is a token there, get it and exit GRAB-TOKEN
3: go to the home-base and if there is a token there, get it and exit GRAB-TOKEN
4: go to the other SR and get token

Suspend & Wake-Up

Recall that an agent A performs SUSPEND when further exploration progress is blocked by *dangerous* links, but A knows that eventually at least one of those link will become unblocked. The basic idea is to put the token in the home-base to signal "I want to be waken-up", check whether a progress has been made before the token was put down (to prevent deadlock, as an agent performing WAKE-UP after removing its token from a dangerous edge might have arrived to the home-base before the token was put there) and, if not, then wait until the token disappears. An agent performing WAKE-UP simply moves a token from the home-base (if there is any) to its primary SR.

The problems arise because several agents might be executing SUSPEND, WAKE-UP and VERIFY simultaneously, and because the agents do not necessarily agree on the correct SR. Dealing with that constitutes the most technical part of the algorithm. The basic idea is to wait until any activity going on (detected by non-empty home-base or SR) looks to have finished and then restart SUSPEND. Still, there are many possible cases 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 home-base by moving its token to a SR. In order to make GRAB-TOKEN work, the waking-up agent first places its token in the SR and then removes the token from the home-base. If the home-base is empty or the SR is full, 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 home-base 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 the BH is located in one of the storerooms, all agents (that have finished initialization) are informed; otherwise an informed agent knows that

Algorithm 5 SUSPEND

```
 1: go to home-base, wait until it is empty and put a token there
 2: scan all known SR's and return to home-base
 3: if SR's were empty then
 4:     traverse the local map
 5: else
 6:     if there is a token in home-base then
 7:         get token
 8:         go to the SR that contained a token, wait until it becomes empty and
                restart SUSPEND
 9:     else // my token has been moved
10:         GRAB-TOKEN
11:         restart SUSPEND
12:     end if
13: end if
```

upon return from traversal

```
1: if traversal revealed progress then
2:     GRAB-TOKEN
3: else
4:     wait until home-base becomes empty
5:     GRAB-TOKEN
6: end if
```

Algorithm 6 WAKE-UP

```
1: go to home-base and if empty, abort
2: go to "correct" SR
3: if SR full then
4:     abort
5: else
6:     put token
7:     go to home-base
8:     GRAB-TOKEN
9: end if
```

both storerooms are safe. However, the notion of an informed agent is for the purpose of the proof only. The agents themselves may not know whether they are informed or not.

The overall structure of the correctness proof, which is quite complicated, is the following: we first prove that during the whole algorithm, at most Δ agents enter the BH, and all agents that are alive make progress by eventually exploring 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 least one agent successfully terminates with a correct map.*

Due to the lack of space, we present only the key lemmas, we omit some proofs and we only informally sketch some reasonings.

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

Claim. 1.A token is in the vertex v only if v is a home-base or a SR.

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 the home-base or in the SR 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 1, and by construction we get:

Claim. 2. There are as many empty-handed agents as tokens in the home-base and storerooms.

An agent performing procedure GRAB-TOKEN visits the home-base and possibly some SR's a constant number of times in a search for a token. For the correctness of the algorithm it is important to prove that a token is always found.

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

Proof. Consider, for the sake of contradiction, an agent A executing GRAB-TOKEN that has not found a token. Let t_0 be the time when A sees that its primary SR x is empty and starts to travel back to home-base. Let $t_1 > t_0$ be the time when A arrives to the home-base, finds it empty again, and starts to travel to SR y. By Claim 2, at time t_0 there must be at least one token T in home-base or SR y. However, since A does not find T, T must have disappeared after t_0 before A gets there. The only way for T to disappear is if it is taken by some empty-handed agent B. However, since B is empty-handed, there must be another token T' in some vertex (home-base or SR) at the time when B grabs T. The idea is to argue about T and T' and show that A would find one of them. In particular, we first prove that at some point in time after t_0 both home-base and SR y are full, and then prove that from this fact it follows that A finds a token.

Let us focus on the time t' when B put T' and thus became empty-handed. We distinguish three cases. First, consider $t' > t_1$. B could not have removed T from the home-base before time t_1, therefore at time t_1 (and t' as well, as it is B that removes it) T must be in SR y. Since A started traveling from the home-base to SR y at time $t_1 < t'$ and due to the FIFO property, B cannot get to SR y before A and so A finds T in SR y – contradiction.

Next, let $t' < t_0$. This means that at time t_0 both A and B are empty-handed, and moreover, SR x is empty. Hence, due to Claim **??**, at time t_0 both home-base and SR y are full.

Third, let $t_0 < t' \leq t_1$. There are two possibilities: (1) B (at time t') put T' in SR x. Since $t_0 < t'$, due to FIFO property B cannot take T from the home-base before A does – contradiction. (2) B (at time t') put T' somewhere else (home-base or SR y). In such a case, at time t' both home-base and SR y are full, containing T and T': By assumption, B is the agent that takes T, therefore T did not move between t_0 and t'.

Hence, it must be the case that the home-base and SR y are full at some time t between t_0 and t_1. Since we suppose that A does not find a token, it must be that both tokens in home-base and SR y disappear at some time after t. However, at time t_0, SR x is empty, so at that time at most one agent other than A is empty-handed. Any agent that becomes empty-handed by putting a token in SR x after t_0 cannot, due to FIFO, prevent A from grabbing a token. This means that only one of the tokens in home-base and SR y can disappear after time t and before A arrives there, i.e. A will find a token – contradiction.

Lemma 2. *A token is removed from a given link less then* $2\Delta MN^3$ *times.*

Proof. A token is put and removed on a link only during *cautious step*. Cautious steps are performed only on line 13 in EXPLORE, which is executed less than N^3 times (at most N^2 iterations of the inner loop, for at most $N-1$ candidate vertices). EXPLORE is called by each of the Δ agents at most M times. Finally, each agent might reset the algorithm once, applying rule **R4**

We now aim at proving that at most Δ agents disappear in the 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. To do so, we use the following technical lemmas, whose proofs are omitted due to the lack of space.

Lemma 3. *Consider a situation when both* SR*'s are full and agents A and B are the only empty-handed agents. Then, before A or B grabs a token from the home-base or some* SR*, no agent other than A or B grabs a token from a* SR*.*

Lemma 4. *No token placed in* SR1 *will be stolen. Moreover, let A be an agent knowing that* SR1 *is safe that puts a token in the home-base. Then A's token will not be kicked out to* SR2*.*

Lemma 5. *A token put in a* SR x *by an informed agent A executing* VERIFY *can be removed from x only by A.*

We can now prove the following:

Lemma 6. *When an agent A adds a vertex v to its local map as a new vertex, then the local map indeed did not contain v.*

Proof. A vertex 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:
- By having the port γ still dangerous after executing the loop on lines 5..9 for $2\Delta MN^3$ times. However, if $w = v$, then between each iteration of that loop the port γ is cleared which is a contradiction with Lemma 2. Hence, $w \neq v$.
- 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 $v \neq w$.
- By having VERIFY return false. VERIFY returns false if the agent A has not found the token in the vertex p (which is equal to its correct SR x if $v = w$) for at least $2\Delta MN^3$ times. Note that A always leaves SR x with its token there. We distinguish two cases:
(i) If x=SR1 the lemma follows from the second part of Lemma 4: no other agent steals A's token from SR1, so if $v = w$ then A always sees a token in p and, subsequently, VERIFY never returns FALSE.
(ii) Let x=SR2. Which agent could remove A's token from SR2? From Lemmas 4 and 5 we know that A's token was not removed from SR2 by an agent B executing VERIFY from SR1, because in that case B's token remains in SR1. It cannot be the case that A's token was removed by an agent B executing VERIFY from SR2, because that agent would have first placed its token in SR2. Therefore, A's token was removed by an agent B executing a GRAB-TOKEN as a part of SUSPEND or WAKE-UP. However, for each removal of a token from a SR by an agent B executing SUSPEND there must have been a wake-up of some other agent that kicked out a token from the home-base to a SR (otherwise B would have picked up its token in the home-base). The only exception are the cases when an agent becomes informed and first takes a token from its old primary SR, which can happen at most Δ times. The lemma follows from the fact that there are less then $2\Delta MN^3$ wake-ups.

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

Lemma 7. *If A enters* BH*, the link e upon which it arrived is marked by its token.*

Since no agent enters a link marked by a token and the degree of the BH is at most Δ, we get:

Theorem 2. *At most Δ agents die.*

The next lemmas are needed to show that no deadlock can occur, i.e. every agent is always able to continue its algorithm after some finite time. First, we prove that no deadlock occurs when an agent is waiting for a disappearance of a token:

Lemma 8. *A token from the home-base eventually disappears.*

Proof. The only way a token can be put in the home-base is in SUSPEND. Consider for the sake of contradiction that an agent A puts a token in the home-base at time t_0 and that token never disappears. That means A went to its primary SR x and found in empty at time t_1, then returned and went to rescan.

We claim that if the token from the home-base does not disappear, then no token appears in the SR x after time t_1. An agent B executing VERIFY cannot place a token in SR x after t_1 – if B checked the home-base (line 3. of VERIFY) before t_0, then A would have found its token in SR x, if it checked it after t_0, it would wait in the home-base until it becomes empty. The only other possibility is that B is executing WAKE-UPand placed its token in SR x after t_1. In such case B would find A's token in the home-base (when executing GRAB-TOKEN) and take it. Contradiction.

Because A did not take its token after returning from rescan, it has seen no progress and did not terminate. This means (by Theorem 2 and from the two-connectivity of G) that it cannot be the case that all blocked links lead to the BH. Therefore one of them will eventually be freed and some agent B will execute WAKE-UP.

If $x = 1$ (i.e. A's primary SR is SR1), B will execute WAKE-UP using SR1 (either because SR1 was its primary SR, or because of rule **R4** – the link to the SR1 is free due to rules **R2** and **R3**) and since SR1 is empty after time t_1, it will indeed remove A's token from the home-base. Contradiction.

If $x = 2$, there are two cases. If B's primary SR is SR2, the same argument as above applies. Otherwise SR1 does not contain the BH and the link leading to it will eventually become free and due to rule **R3** remain so. That means A will eventually notice that SR1 is safe and apply rule **R4**, executing GRAB-TOKEN starting from SR2. Since SR2 remains empty after t_1, A will pick its token from the home-base. Contradiction.

In a similar fashion, we can show the following lemma, which is, due to space constraints, presented without proof:

Lemma 9. *A token from a* SR *eventually disappears.*

From the construction and Lemmas 8 and 9 we get:

Theorem 3. *An agent never deadlocks.*

The next two lemmas are crucial for bounding the number of moves. Due to space restrictions we present them without proofs.

Lemma 10. *An agent spends $O(\Delta MN^4)$ moves in one call to* VERIFY.

Lemma 11. *An agent spends $O(\Delta MN^7)$ steps executing one iteration of the outer loop of Algorithm 1.*

The last property we need for the proof of Theorem 1 is:

Lemma 12. *Each informed agent has a correct map.*

Proof. It follows from Lemma 6 that if an agent A adds a new vertex v to its map, then indeed v has not been in A's local map before. So it remains to be proven that if an informed agent A adds an edge (u, w) between two visited vertices to its map, 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.

We first prove that after successfully finishing N^2 iterations of the loop on line 4 in EXPLORE the sequence β^* defines a (not necessarily simple) cycle connecting v and u, whose length is a multiple of $|\beta|$. Let $\beta = (\beta_1, \beta_2, \ldots, \beta_k)$ where each β_i specifies two port numbers: a consistent traversal must arrive via port p_1 and leave via port p_2. Since $k \leq N$, by traversing β^* for N^2 steps it must happen that the agent visits a particular vertex q twice with the same position in the sequence β; say β_i. Clearly, from now on the agent walks in cycle. Let q be the first such vertex. However, since β_i specifies also the arriving port number, it means that the agent has both times arrived to q using the same port, i.e. it already started in the cycle.

To conclude, we prove that if VERIFY returns TRUE for some informed agent it must be that the cycle formed by β^* has length $|\beta|$ and hence $v = w$. If VERIFY returns TRUE it means that A saw a token in p at least $2\Delta MN^3$ times and between every two successive visits of p there was a time when home-base was free and, if there are two storerooms, also a time when SR2 was free. If p was not SR1, it must be that either p is home-base or p is SR1 and each of the $2\Delta MN^3$ times some agent put its token at p (which was removed before the next visit of A in p).

We conclude the proof by showing that a token is put in p less then $2\Delta MN^3 + \Delta MN$ times. There are two possible situations when an agent B could put its token to p: either B performs a VERIFY in SR2 (there are at most ΔMN such cases: B must be a non-informed agent and it puts its token once per each call of VERIFY before getting informed), or B performs a SUSPEND-WAKE-UP pair. However, in the latter case there must be a cautious step that triggers this WAKE-UP which, according to Lemma 2, accounts for another $2\Delta MN^3$ possibilities.

By Lemmas 1-12, the main theorem (Theorem 1) follows.

Let us now consider the number of moves. By Lemmas 10,11 plus the fact that each of the Δ agents performs at most M iterations of the loop in Algorithm 1, we have

Theorem 4. *The* BH *can be located using* $O(\Delta^2 M^2 N^7)$ *moves.*

References

1. I. Averbakh and O. Berman. A heuristic with worst-case analysis for minimax routing of two traveling salesmen on a tree. *Discr. Appl. Math.*, 68:17–32, 1996.

2. 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 Symp. on Theory of Computing (STOC'98)*, 269–287, 1998.
3. M. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proc. 35th Symp. on Foundations of Computer Science (FOCS'94)*, 75–85, 1994.
4. M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Symposium on Foundations of Computer Science (FOCS'78)*, 132–142, 1978.
5. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in tree networks. In *Proc. 8th International Conference on Principles of Distributed Systems (OPODIS 2004)*, 35–45, 2004.
6. S. Das, P. Flocchini, A. Nayak, and N. Santoro. Exploration and labelling of an unknown graph by multiple agents. In *Proc. 12th Int. Coll. on Structural Information and Communication Complexity (SIROCCO'05)*, 99-114, 2005.
7. X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.
8. S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, and N. Santoro. Optimal search for a black hole in common interconnection networks. *Networks*, 47(2):61–71, 2006.
9. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*. To appear.
10. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocols. *Distributed Computing*. To appear.
11. S. Dobrev, P. Flocchini, and N. Santoro. Improved bounds for optimal black hole search in a network with a map. In *Proc. of 10th Int. Coll. on Structural Information and Communication Complexity (SIROCCO'04)*, 111–122, 2004.
12. G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *Transactions on Robotics and Automation*, 7(6):859–865, 1991.
13. P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. In *6th Latin American Theoretical Informatics Symp. (LATIN'04)*, 141–151, 2004.
14. P. Fraigniaud and D. Ilcinkas. Digraph exploration with little memory. In *21st Symp. on Theoretical Aspects of Computer Science (STACS'04)*, 246–257, 2004.
15. G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM J. on Computing*, 7:178–193, 1978.
16. R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary graphs. In *Proc. 12th Coll. on Structural Information and Communication complexity (SIROCCO'05)*, 200–215, 2005.
17. R. Oppliger. Security issues related to mobile code and agent-based systems. *Computer Communications*, 22(12):1165 – 1170, 1999.
18. P. Panaite and A. Pelc. Exploring unknown undirected graphs. *J. Algorithms*, 33:281–295, 1999.
19. CL. E. Shannon. Presentation of a maze-solving machine. In *8th Conf. of the Josiah Macy Jr. Found. (Cybernetics)*, 173–180, 1951.
20. Jan Vitek and Giuseppe Castagna. Mobile computations and hostile hosts. In D. Tsichritzis, editor, *Mobile Objects*, 241–261, 1999.