Map Construction and Exploration by Mobile Agents Scattered in a Dangerous Network

Paola Flocchini, University of Ottawa
Matthew Kellett, Defence R&D Canada – Ottawa
Peter Mason, Defence R&D Canada – Ottawa
Nicola Santoro, Carleton University

Abstract

We consider the map construction problem in a simple, connected graph by a set of mobile computation entities or agents that start from scattered locations throughout the graph. The problem is further complicated by dangerous elements, nodes and links, in the graph that eliminate agents traversing or arriving at them. The agents working in the graph communicate using a limited amount of storage at each node and work asynchronously. We present a deterministic algorithm that solves the exploration and map construction problems. The end result is also a rooted spanning tree and the election of a leader. The total cost of the algorithm is $O(n_s\ m)$ total number of moves, where m is the number of links in the network and n_s is the number of safe nodes, improving the existing $O(m^2)$ bound.

1 Introduction

In networked environments supporting mobile entities, teams of such entities can be employed to perform a variety of system tasks, such as searching for a resource or a mobile user, detecting deadlocks, upkeep of routing tables, etc. Underlying many of these tasks is the primitive process of *exploration* of the network, in which all network sites and links are traversed by at least one member of the team. Closely related to this problem is that of *map construction* requiring the agents to produce the current map of the network; clearly to produce the map, the network must be first explored. Indeed exploration of an unknown network and construction of its map by mobile entities—usually called agents or robots—has been the subject of intense research efforts, and an extensive literature exists on the subject.

Different instances of the problem exist depending on a variety of factors, including the synchrony or asynchrony of the agents' actions and movements, whether the agents and/or the network sites have distinct identifiers or are anonymous, the amount of memory with which an agent is endowed, whether the agents are co-located (i.e., the team starts from a single site) or scattered (i.e., the agents start from different sites), the coordination and

communication mechanisms available to the agents, etc. (e.g., see [1–4, 9, 10, 20–22, 25, 26]). Notice that, except for trees, the exploration 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 tokens to white boards.

Most of the investigations on the exploration problem have assumed that the environment is *safe* for the exploring agents; this assumption unfortunately does not always hold in reality. The exploration problem has thus been examined also when some components of the network, nodes and links, are *unsafe* [5–8, 11–19, 23, 24]. The danger is considerable: any agent arriving on these harmful nodes, called *black holes*, and/or traversing such harmful links, called *black links*, is immediately eliminated and this destruction leaves no discernable trace. Note that such a harmful presences are not uncommon; for example, any undetectable crash failure of a site or a link in an asynchronous network transforms that site into a black hole and the link into a black link.

Since the location of these harmful components is not known a priori, the determination of the safe and unsafe parts of the network becomes a crucial task. Indeed, by solving the map construction problem in this environment, subsequent protocols can proceed without danger to the agents executing them. The problem, called *Dangerous Graph Exploration* (DGE) is for a team of agents to explore the network and, within finite time, generate a map of the safe part with an indication at each safe site of the dangerous ports (black links or leading to a black hole). Solving this problem is a dangerous (and possibly impossible) task for the agents. The goal is for at least one agent from the team to survive, with all surviving agents terminating with the map.

The DGE problem has been studied extensively assuming that the only harmful component in the network is a single black hole [7, 8, 11–19, 23, 24]. This version of the problem is called *black hole search* and its investigations have been restricted to the case when the agents are *co-located*, i.e. the team is injected in the system from a single node; the only exceptions are [6] that considers multiple black holes in synchronous networks of known topology, and [13, 19] that consider the more difficult case of *scattered* agents (i.e., when each agent is initially in an arbitrary location in the network) in the case of ring networks.

We are interested in the more general version of the DGE problem, that is in arbitrary networks of unknown topology with an arbitrary number of black holes and black links, and with the agents are asynchronous and initially scattered through the network. This general setting has been recently investigated in [5] where conditions for solvability of the related problems of *election* and *rendezvous* were established in the case of fully *anonymous* systems (i.e., where both agents and nodes are anonymous). This characterization was established in the common *white board* model: each node provides incoming agents with a limited memory—the white board—whose access is provided in fair mutual exclusion, and that the agents can use to communicate. To our knowledge, no other studies exist on this setting.

In this paper we continue the investigation of this general setting of the DGE problem (arbitrary networks of unknown topology with an arbitrary number of black holes and black links, with asynchronous scattered agents) in the white board model, and we consider the *non-anonymous case*, that is when the agents and/or the network sites have distinct ids.

For the problem to be solvable, all the safe nodes (i.e., not black holes) must clearly be connected; furthermore, the number of agents must be greater than unsafe links (i.e., black links or leading to a black node) incident on the safe nodes. It is important to stress that asynchrony imposes severe limitations to termination and the accuracy of any exploration and map construction protocol even when there is a single black hole and the agents are co-located. In fact, assuming connectivity and enough agents, it might be possible to guarantee that all unsafe links will be marked as such in the map when an agent terminates; however, unless both the number of safe nodes and the number of unsafe links are known precisely, it is impossible to guarantee also that all safe links will be marked as such in the map.

Under these provisos, we present a protocol that solves the *map construction* problem, as well as the *spanning tree construction*, *election*, and *rendezvous* problems. The proposed protocol does so at a worst case cost of at most $O(n_s\, m)$ total number of moves, where where m is the number of links in the network and n_s is the number of safe nodes. This improves the existing $O(m^2)$ bound of the protocol of [5] under the same circumstances.

2 Model and problem

Let $G = (V, E)$ be a simple connected graph of $n = |V|$ nodes and $m = |E|$ edges. Let $E(u)$ be the edges incident to node $u \in V$, $d(u) = |E(u)|$ be the degree of u, and Δ be the maximum degree in G.

At each node $u \in V$, there is a distinct label (called port number) associated to each of its incident links (or ports); let $\lambda_u(u, v)$ denote the label associated at v to the link $(u, v) \in E$, and λ_v denote the overall injective mapping at v. The set $\lambda = \{\lambda_u | u \in V\}$ of those mappings is called a *labelling* and we shall denote by (G, λ) the resulting edge-labelled graph. The nodes of G can be *anonymous* (i.e., without unique names).

Operating in (G, λ) is a set A of k agents. The agents are *distinct* (each has an id), *autonomous* (each has private memory and performs its own computations), and *mobile* (each can move from a node to one of its neighbours). All agents have the same behaviour, that is they follow the same algorithm or protocol, and are *asynchronous* in that their actions take a finite but unpredictable amount of time.

The agents are initially *scattered* in G, their initial location described by the mapping $h : A \Rightarrow N$; in the following, $h(a) \in V$ will be called the *home base* of $a \in A$. The agents do not know G nor its size; they also do not know the number k of agents nor their location. The agents start at different and unpredictable times.

Each node has available a limited amount of storage, called *white board*. Agents communicate by reading from and writing on the white boards; access to a white board is gained fairly in mutual exclusion. The mutual exclusion property and the distinct ids of the agents allow the agents to operate as if the edges were FIFO and the nodes had unique identifiers; hence, in the following we will assume that this is indeed the case, without any loss of generality.

The network in which the agents operate is dangerous due to the presence of harmful nodes and edges, called *black holes* and *black links*. A *black hole* is a network site that

destroys any incoming agent without leaving an observable trace; a *black link* is an edge, connecting two safe nodes, that causes a similar destruction to any agent traversing it. The location of the black holes and black links is unknown to the agents. Let $V_B \subseteq V$ and $E_B \subseteq E$ denote the set of black nodes and of black links, respectively. Let the *frontier* of G be the set $F_B = \{[u,v] \in E \setminus E_B : u \in V \setminus V_B \wedge v \in V_B\}$ of edges incident on the black holes; let the *safe* portion of G be the labelled graph $(G_S = (V_S, E_S), \lambda_S)$, where $V(S) = V \setminus V_B$, $E_S = E \setminus (E_B \cup F_B)$ and λ_S is the restriction of λ to E_S.

The *map construction* (`DGE`) problem is for the team of agents to explore the network and, within finite time, to construct a map of all the safe nodes V_S and where all links of the frontier F_B and the black links E_B are indicated. The map is unambiguous if every safe edge is marked as such. Since the location of the black holes and black links is a priori unknown to the agents, some agents will be destroyed in this process. We say that the problem is solved if at least one agent survives, and all surviving agents within finite time terminate having such a map.

Some obvious limitations follow from the problem definition. In particular, for the problem to be solvable, clearly the safe portion of the network G_S must be connected. Furthermore, since at least one agent must survive, the number of agents k must be greater than $f = |F_B| + 2|E_B|$. Thus, in the following, we assume that G_S is connected, that $k \geq f + 1$. We will also assume, to detect termination, that the number $n_S = |V \setminus V_B|$ of safe nodes is known.

3 Algorithm

Our approach to the map construction problem also solves the spanning tree construction problem and the election problem, all with scattered agents working in a dangerous network. The agents start out on scattered home bases that become the roots of the trees that are generated by the *exploration* process. These trees are merged during the *verification* process. The result is a complete map of G_S showing the frontier, F_B, and the black links, E_B, (map construction), a tree that spans all the nodes of G_S (spanning tree construction), and a tree root that can be used to elect a leader agent for subsequent protocols (election).

On each home base r, the first agent that begins executing the algorithm starts by establishing a root marker, *rmarker*, with the same id as the home base which becomes the root of a new tree T_r. The root marker is initialized with a map that includes the home base and its links, and a number of counters. The root marker's counters include the number of links in need of exploration, $rmarker.c_e$, which is initially set to the degree of the home base, $d(r)$; the number of links in need of verification, $rmarker.c_v$, which is initially set to 0; and the number of safe nodes that have had their links fully explored, $rmarker.c_f$. We use these counters as a convenience since all can be calculated from information stored in the map. Once the initialization is completed, the agent begins by looking for exploration work.

Any subsequent agent b starting on r, upon discovery that the algorithm has already started (such information will be found on the white board), will try to access the root marker. It is possible that the root marker has been moved in the meanwhile (as explained

Algorithm 1 MAIN LOOP

Agent a starts from the root r of its tree T_r.

1: **if** white board is blank **then**
2: initialize root marker's counters and map and set the id of the verifying agent to null
3: **end if**
4: access root marker ▷ Mutually exclusive access via the white board
5: if the root marker has moved then the agent follows the parent pointers
6: until it arrives at the new root and the root marker becomes accessible
7: **while** $r.rmarker.c_f < n_s$ **do** ▷ While there are still safe nodes with unexplored links
8: **if** no *verifying agent* exists and $r.rmarker.c_v > 0$ **then** ▷ First agent becomes the verifying agent
9: become *verifying agent* and record my id in root marker
10: **end if**
11: **if** $r.rmarker.c_v > 0$ and I am the *verifying agent* **then** ▷ Links to verify
12: $r.rmarker.c_v \leftarrow r.rmarker.c_v - 1$
13: choose link that will be verified and mark it on root marker's map
14: take copy of root marker's map and id ($a.rmarkerid \leftarrow r.rmarker.id$)
15: VERIFY LINK
16: **else if** $r.rmarker.c_e > 0$ and I am not the *verifying agent* **then** ▷ Links to explore
17: $r.rmarker.c_e \leftarrow r.rmarker.c_e - 1$
18: choose link that will be explored and mark it on root marker's map
19: take copy of root marker's map and id ($a.rmarkerid \leftarrow r.rmarker.id$)
20: release root marker
21: EXPLORE LINK
22: **else if** a is *verifying agent* **then** ▷ No verification work
23: cease to be the *verifying agent* and remove my id from root marker
24: release root marker
25: **else** ▷ No exploration work
26: release root marker
27: wait ▷ Agent becomes inactive
28: **end if**
29: access root marker
30: **end while**

later); should this be the case, the agent b follows the directions (left on the white boards) until it reaches and access a root marker; it will then begin by looking for exploration work.

The *exploration* process works as follows. Starting from r, an active agent checks the map to see if there is a link or links in need of exploration. If there are, it claims one on the root marker's map and travels there by the shortest path using the map. It then explores the link using *cautious walk*. Cautious walk is way of labelling ports that protects subsequent agents from being eliminated by a black hole or black link. A port is initially unexplored; it has no label. When an agent leaves a node via an unexplored port, the port is marked *dangerous*; when an agent arrives at a node, that port is marked *explored*. Agents are not allowed to enter ports marked dangerous. Since black holes and black links eliminate agents, the first agent entering a port that leads to a black hole or black link will mark it dangerous and prevent all the other agents from being eliminated by entering the same port. If the agent is eliminated, it is considered to have successfully completed its exploration of that link.

Assuming that the agent is not eliminated when it traverses the link and it arrives safely on node v, there are a number of possibilities. It is possible that no other agent has ever visited the node. In this case, the agent marks the node as visited and adds it to the agent's tree by creating a parent pointer pointing to the port by which the agent arrived. The agent then returns to r where it reports the $d(v) - 1$ links that now need to be explored, updates the root marker's map to show the new node, marks its links for exploration, and activates any agents waiting at the root for work (we discuss waiting below). It is also possible that another agent from the same tree or a different tree has already visited v. In this case, the agent returns to r where it reports that there is a link in need of verification, updates the root marker's map to show the location of that link noting the node incident to it but not adding that node to the map (needed for merging later), and activates any agents waiting at the root for work.

Verification is the process of determining whether or not a link is internal or external to the agent's tree, and works as follows. In the root marker of a tree T_r there is a list (possibly empty) of links, incident on nodes in T_r, that need to be verified; this will be called the verification work at T_r. The first agent in T_r to take on verification work becomes the *verifying agent* for T_r and its id is recorded in the root marker. Only one agent in a tree can take on this role at one time, and the verifying agent gives up the role, becoming an exploring agent again, if there is no verification work available. If there is verification work, the verifying agent a chooses one of the links, say (u, v) in need of verification. The agent starts by checking in the map if both u and v are nodes of T_r. If so, the agent marks the link as internal and the verification of that link is done. Otherwise, there are two possibilities: the link is internal but the map has yet to be updated to show that, or the link is external. To determine which case it is, agent a travels to the root of the tree to which v belongs. The agent first takes a copy of its own root marker's map and id; it then moves to u by the shortest path using the map, traverses (u, v), and starts following the parent pointers until it reaches and access the root marker of the tree $T_{r'}$ containing v (as we will show, the agent will always successfully do so). If $r.rmarker.id = r'.rmarker.id$, agent a marks the link as

Algorithm 2 EXPLORATION

Agent a has chosen to explore a link incident to node u. The link $[u, v]$ leads from u to the incident node v and $[v, u]$ leads from v back to u. Let $d(v)$ be v's degree.

1: **procedure** EXPLORE LINK
2: traverse from r to u by the shortest known path
3: walk from u through $[u, v]$ to v ▷ First step of cautious walk

If the agent is not eliminated by a black hole or black link then

4: **if** v is *unexplored* **then** ▷ A new node
5: add v to its map and mark it *visited*
6: set v's parent as $[v, u]$
7: traverse from v through $[v, u]$ to u ▷ Second step of cautious walk
8: traverse from u to r by the shortest known path
9: access root marker
10: $r.rmarker.c_e \leftarrow r.rmarker.c_e + (d(v) - 1)$
11: **if** u has no unexplored links **then** $r.rmarker.c_f \leftarrow r.rmarker.c_f + 1$ **end if**
12: update *rmarker*'s map to add v to the tree and mark its links for exploration
13: **else** ▷ v must be *visited* or *explored*
14: traverse from v through $[v, u]$ to u ▷ Second step of cautious walk
15: traverse from u to r by the shortest known path
16: access root marker
17: $r.rmarker.c_v \leftarrow r.rmarker.c_v + 1$
18: update *rmarker*'s map marking $[u, v]$ for verification and noting v
19: **end if**
20: activate inactive agents
21: release root marker
22: **end procedure**

internal on the map and the verification of the link is done. If $r'.rmarker.id < r.rmarker.id$ and $T_{r'}$ has a verifying agent, agent a becomes an exploring agent in $T_{r'}$; since a is still the verifying agent for T_r and a tree can only have one verifying agent, no new verifying agent can emerge on T_r. Finally, if $r'.rmarker.id > r.rmarker.id$ or $T_{r'}$ has no verifying agent—because all the agents have been eliminated or no agent currently holds that role—then agent a performs a merger.

The *merging* process—a subprocess of the verification process—works as follows. Agent a—which we shall call the *merging agent*—starts the merger by picking up the root marker, $r'.rmarker$, activating any agents waiting at r' for work, and carrying the root marker to the root of the tree containing u; recall that a is verifying the link $[u, v]$), and notice that, due to concurrency and asynchrony, it is possible that this root is no longer r. Agent a thus first returns to v reversing the parent pointers along the path, and traverses $[v, u]$ setting u as the parent of v; it then follows the parent pointers until it finds and accesses the root marker of the tree $T_{r''}$ containing u (as we will show, the agent will always successfully do so). Notice that, during this movement of the root marker $r'.rmarker$ by a, any agents trying to access

Algorithm 3 VERIFICATION

Agent a has chosen to verify link $[u, v]$ between node u in its own tree and node v, which may be in a another tree. Let r be the root of u's tree, T_r and r' be the root of v's tree, $T_{r'}$.

```
1: procedure VERIFY LINK
2:     if v is in the root marker map then
3:         mark the link as internal on the map                ▷ The agent never leaves r
4:         release root marker
5:     else
6:         release root marker
7:         CHASE ROOT
8:     end if
9: end procedure

10: procedure CHASE ROOT
11:     traverse from r to u by the shortest known path
12:     traverse from u to v by the link being verified
13:     traverse from v to r' using the parent pointers and marking the path from v to r'
14:     access root marker
15:     if a.rmarkerid = r'.rmarker.id then          ▷ r = r', so [u, v] is an internal edge
16:         mark the link as internal on the root marker's map
17:         release root marker
18:     else if r'.rmarker.id < a.rmarkerid then                          ▷ r'.id < r.id
19:         if r' has a verifying agent then
20:             become exploring agent
21:             take copy of r''s id and map                          ▷ Start working for T_r'
22:             update r''s map marking [v, u] for verification
23:             release root marker
24:         else
25:             pick up root marker
26:             activate inactive agents
27:             RETURN HOME
28:         end if
29:     else if r'.rmarker.id > a.rmarkerid then                          ▷ r'.id > r.id
30:         pick up root marker
31:         activate inactive agents
32:         RETURN HOME
33:     end if
34: end procedure
```

Algorithm 4 TREE MERGING

Agent a is verifying verify link $[u, v]$ between node u in its own tree, r and node v, in another tree, r'. The agent has picked up and is moving r''s root marker.

```
1: procedure RETURN HOME
2:     traverse from r' to r
3:         reverse the parent pointers along the path from r' to v
4:         add a parent pointer from v to u
5:     merge root markers                    ▷ Merge r''s root marker with r's
6:         if not verifying agent for r.rmarker then cease to be verifying agent end if
7:     activate inactive agents
8:     release root marker
9: end procedure
```

$r'.rmarker$ or trying to return to r' follow the parent pointers until they reach the new root and the root marker becomes accessible. Since the links can be considered FIFO and all the non-merging agents are following the same path as the merging agent, the merging agent is guaranteed to arrive first.

When a accesses the root marker of the tree $T_{r''}$, a merges the information in $r'.rmarker$ with that in $r''.rmarker$, discarding $r'.rmarker$; in particular, it updates the map with the reversal of the parent pointers and the addition of the verified link as a tree link. It also activates any agents waiting at r'' for work, and the verification of the link $[u, v]$ is completed. In the case where $r''.rmarker.id = r.rmarker.id$, agent a is back at its old tree and continues to be the tree's verifying agent. In the case where $r" \neq r$, if $T_{r''}$ already has a verifying agent, a becomes an exploring agent in $T_{r''}$; otherwise, it becomes the verifying agent of $T_{r''}$.

When an active agent finishes its work, it does so on a root, where it checks for new work. If a verifying agent finds that there is no verification work available ($rmarker.c_v = 0$), it switches to be an exploring agent and checks for exploration work. If an exploring agent finds that there is no exploration work available ($rmarker.c_e = 0$), it becomes inactive and waits to be activated when new work arrives.

For clarity, we provide the entire algorithm in pseudo-code. The main loop can be found in Algorithm 1. Exploration is detailed in Algorithm 2 and verification and tree merging in Algorithms 3 and 4, respectively.

4 Correctness and complexity

We first introduce some common terminology. During the execution of our algorithm, the agents are either *alive* or *eliminated*. The agents that are alive are either *active* or *waiting*. An agent is waiting at time t if it is at its root, there is no work available (there are no known links to explore or verify), and there is still work to be done in the system. Active agents are either *exploring* or *verifying*. Exploration and verification both start and finish

at a root.

Lemma 1. *At any point in time, there is at least one alive agent.*

Proof. Because of the use of cautious walk, at most one agent will die on each link in the frontier F_B and at most two on every black link in E_B (one in each direction). Since, by definition, $k > |F_B| + 2|E_B|$ the lemma follows. □

Lemma 2. *The movement of any root marker will terminate in finite time after at most 2n moves.*

Proof. (sketch) Let a verifying agent a_l be verifying an external link between its own tree T_l and an adjacent tree T_{l+1}. Let the link be $[u_l, v_{l+1}]$ where u_l is the node in T_l incident to the link and v_{l+1} is the node in T_{l+1}. Let us consider the case when a_l picks up the root marker from root r_{l+1}. By construction, an agent only grabs the root marker when the root marker's id is higher than its own, $r_l.rmarker.id < r_{l+1}.rmarker.id$, (see Algorithm 3.30) or the tree to which the root marker belongs, T_{l+1} in this case, has no verifying agent (see Algorithm 3.25). The path from the root where the agent picked up the root marker to the agent's root has three segments. The first segment is the path $\pi(r_{l+1}, v_{l+1})$ in T_{l+1} from the root r_{l+1} to the link being verified. Since a_l is holding the root marker $r_{l+1}.rmarker.id$, the length of this segment cannot change and within finite time the agent traverses it while reversing the parent pointers along the way. The second segment is the link being verified, $[v_{l+1}, u_l]$, which is added to the tree and traversed in finite time. The last segment is the path $\pi(u_l, r_l)$ from the link being verified to the agent's root, r_l. $\pi(u_l, r_l)$, If the root marker $r_l.rmarker$ has not moved, then the agent will find it when it arrives at r_l, terminating the entire operation; since the trees are disjoint, the movement of the root marker would cost in this case at most $n - 1$ moves. If the root marker $r_l.rmarker$ has moved, it could only have been moved by a verifying agent a_{l-1} from another tree, T_{l-1}, where $r_{l-1}.rmarker.id < r_l.rmarker.id$, with a_{l-1} picking up $r_l.rmarker$ and moving it towards r_{l-1}. In general, there can be a sequence of verifying agents $a_l, a_{l-1}, ..., a_2, a_1$ such that a_i is moving $r_{i+1}.rmarker$ towards r_i with $r_i.rmarker.id < r_{i+1}.rmarker.id$, where $1 <= i <= l$ and $1 <= l <= k - 1$ (see Figure 1). From the total order of the ids it follows that a_1 will in finite time bring $r_2.rmarker$ to r_1. Hence within finite time all a_is will bring $r_{i+1}.rmarker$ to r_1. Therefore, agent a_l finishes its last segment, $\pi(u_l, r_1)$, in finite time.
The number of moves performed by a_l will thus be no more than

$$\sum_{i=1,...,l} |\pi(r_{i+1}, u_i)| + |\pi(u_i, r_i)|.$$

The trees are disjoint but the paths $\pi(u_i, r_i)$ and $\pi(r_i, v_i)$ might not be (see Figure 1). Let $n_i = |T_i|$ be the number of nodes in T_i; clearly $|\pi(u_i, r_i)| + |\pi(r_i, v_i)| < 2n_i$. Thus, since $\sum_{i=1,...,l+1} n_i \leq n$ the lemma follows. □

The following two lemmas now follow directly from Lemma 2.

Lemma 3. *A verifying agent verifying a link will always find a root marker.*

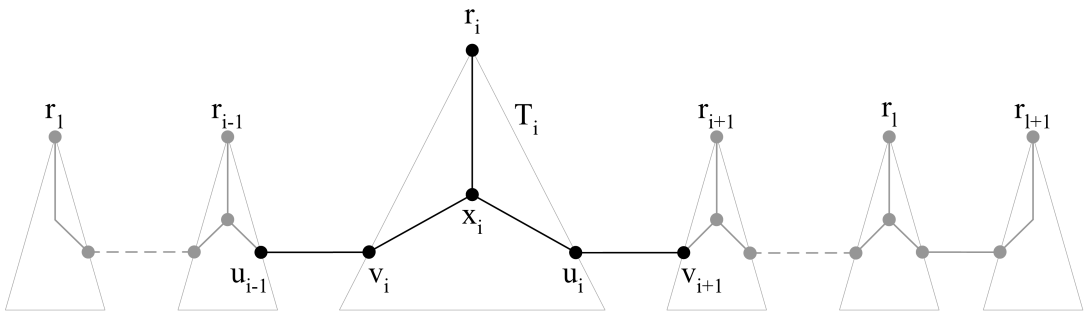

Figure 1: A chain of $l+1$ trees as discussed in Lemma 2.

Lemma 4. *A verifying agent returning with a root marker will find a root marker that has the same or a smaller id than the root for which it is working.*

Lemma 5. *If an agent picks up a root marker, in finite time, that root marker will be discarded and its information merged with that of another root marker.*

Proof. A root marker can only be picked up by a verifying agent. By Lemma 2, the root marker's movement terminates within finite time; by construction, termination ends in a merger, which is the combination of two root markers into a new root marker.At the end of the merger, the moved root marker will be discarded. □

As a consequence of Lemmas 3–5, it follows that

Lemma 6. *An agent that is verifying will finish verifying within finite time.*

Lemma 7. *At most $k-1$ root markers are moved, and therefore discarded, and the total cost for moving all the root markers during mergers is at most $O(kn)$.*

Proof. There are at most k home bases and therefore at most k root markers in G. By Lemma 5, the movement of a marker will result in the deletion of that marker. By Lemma 2, any movement of a root marker terminates after at most $2n$ moves. Therefore, the maximum total cost for moving all the root markers is $2n(k-1)$. □

Lemma 8. *Verification costs $O(nm)$ moves.*

Proof. (sketch) A verifying agent verifies a link by first checking if its incident nodes are on the map; if this is the case, the verification costs nothing to since there is no movement. If this is not the case, the verifying agent verifies the link by traversing it and going to the root of the tree on the other side. The agent performs up to $n-1$ moves down to the link, traverses it. The agent then moves towards the root of the other tree (the location of the root marker). The root marker of that tree might have moved, but by Lemma 3, the agent will reach a root marker within $2n$ moves. If the agent now terminates the verification (because the link is internal or the other root marker has lower id and a verifying agent) the cost

will be at most $3n$. Otherwise, the agent grabs the root marker and moves towards its own root, which might have been moved. Again, by Lemma 3, the agent will reach a root marker within $2n$ moves. In this case, the agent will have travelled at most $5n$ moves. Each edge is verified at most twice, once in each direction; hence the lemma follows. □

Lemma 9. *An agent that is exploring will finish exploring within finite time after at most $O(n)$ moves.*

Proof. (sketch) Let exploring agent a choose link $[u, v]$ between node u in the agent's tree and some incident node v. Like the verifying agent in the proof of Lemma 2, the exploring agent's path has three segments. The first segment is from the agent's root r to u, which is across known safe links and takes finite time. The second segment is the cautious walk across $[u, v]$ and back. If $[u, v]$ is a black link or v is a black hole, a is eliminated and its exploration is completed in finite time in at most n moves. If v is safe, the last segment of a's path is from u towards its root. Since the root marker might have moved, the agent must follow the pointer until it finds the root marker. By Lemma 2, it follows that the agent will find the root marker within at most $2n$ moves, for a total of $O(n)$ moves in the worst case. □

Lemma 10. *Exploration costs $O(nm)$ moves.*

Proof. Each edge is explored at most twice, once in each direction. By Lemma 9, the total number of moves follows. □

Lemma 11. *At time t, there must be at least one agent alive that is not waiting.*

Proof. (sketch) By Lemma 1, there is always at least one alive agent. By contradiction, assume that there exists a time t when all live agents are waiting at the root marker of some tree or trees in G_S. Consider any such tree T_r. First observe that the last agent to become waiting at root r must have done so because there was no work (i.e., no links to explore, no links to verify) and the condition for termination ($r.rmarker.c_f = n_s$) is not satisfied. Since all agents are waiting, this situation will not change in time. In other words, all the links incident on the nodes in T_r have been explored and, with the exception of black links and those leading to black holes, have also been verified. But this implies that T_r contains all safe nodes (i.e., $r.rmarker.c_f = n_s$), which is a contradiction. □

Lemma 12. *Every link in $G_S \cup F_B \cup E_B$ is eventually explored and those in G_S are also verified.*

Proof. (sketch) An active agent that is not waiting is either exploring or verifying. By Lemma 11, there is always one such agent. By Lemmas 6 and 9, each exploration and verification will be completed. □

Before we go on to the next lemma, we must define which agents are actually working for a tree. Every exploring agent is exploring an edge from node u to node v, $[u, v]$. An agent a is working for tree T_r if a *is exploring* $[u, v]$ where $u \in T$ or if it is the verifying agent for T_r and it is not an exploring agent in any other tree. All agents working for T_r comprise the $team(T_r)$ with size $k_r = |team(T_r)|$.

Lemma 13. *The root marker of a tree whose team of agents have all been eliminated is eventually moved.*

Proof. (sketch) Let us consider a tree T_r whose team of agents have all been eliminated by by black links or black holes. By definition, the tree's agents must all have been exploring; hence, the tree has no verifying agent. Consider first the case where the tree contains all the safe nodes. This would imply that $k_r = k$, that is that all agents have been eliminated, which contradicts Lemma 1. Therefore, there must exist at least one unexplored link leading to a safe node outside the tree. By Lemma 12, this link will eventually be verified by some agent a. Since T_r has no verifying agent, its root marker will eventually be moved by a. □

Lemma 14. *If a verifying agent finds a root marker with a lower id and that has a verifying agent then its own root marker is eventually moved.*

Proof. (sketch) Let a be the verifying agent for tree T_r verifying link $[u, v]$ and let $T_{r'}$ be the tree for which a is now exploring. By construction, this situation can only occur if $T_{r'}$ has a verifying agent a' and the id of the root marker is smaller. Note that no new verifying agent can be created on T_r. When a becomes an exploring agent in $T_{r'}$ it first marks $[v, u]$ for verification in r'''s root marker's map. By Lemma 12, this link is eventually going to be verified. The verification will be performed by a verifying agent from a root marker with a smaller id than that of T_r. In fact, either this agent is a', which we know is from a smaller id tree, or an agent from a tree that has absorbed $T_{r'}$; in the latter case, since $T_{r'}$ has a verifying agent, it can only be absorbed by a tree which has a smaller id than the root marker of $T_{r'}$. Therefore, the verifying agent arriving at r will move its root marker. □

From Lemmas 8–14 the main result follows

Theorem 1. *Within finite time, after at most $O(nm)$ moves, a rooted spanning tree of G_S will be constructed, all safe edges will be indicated as such, all ports in G_S leading to a black hole or to a black edge will be marked as dangerous.*

References

[1] I. Averbakh and O. Berman. A Heuristic with Worst-Case Analysis for Minimax Routing of Two Travelling Salesmen on a Tree. *Discrete Applied Mathematics*, 68(1–2):17–32, June 1996.

[2] M. A. Bender, A. Fernández, D. Ron, A. Sahai, and S. Vadhan. The Power of a Pebble: Exploring and Mapping Directed Graphs. In *STOC '98: Proceedings of the 30th ACM symposium on Theory of computing*, pages 269–278, New York, NY, USA, 1998. ACM.

[3] M. A. Bender and D. K. Slonim. The Power of Team Exploration: Two Robots Can Learn Unlabeled Directed Graphs. In *FOCS 1994: Proceedings of the 35th Symposium on Foundations of Computer Science*, pages 75–85, November 1994.

[4] M. Blum and D. Kozen. On the Power of the Compass (or, Why Mazes Are Easier to Search than Graphs). In *FOCS 1978: Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 132–142. IEEE, October 1978.

[5] J. Chalopin, S. Das, and N. Santoro. Rendezvous of Mobile Agents in Unknown Graphs with Faulty Links. In *DISC 2007*, volume 4731 of *LNCS*, pages 108–122. Springer, 2007.

[6] C. Cooper, R. Klasing, and T. Radzik. Searching for Black-Hole Faults in a Network Using Multiple Agents. In *OPODIS 2006*, volume 4305 of *LNCS*, pages 320–332. Springer, 2006.

[7] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of Searching for a Black Hole. *Fundamenta Informaticae*, 71(2,3):229–242, 2006.

[8] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a Black Hole in Synchronous Tree Networks. *Combinatorics, Probability and Computing*, 16(4):595–619, July 2007.

[9] S. Das, P. Flocchini, A. Nayak, and N. Santoro. Distributed Exploration of an Unknown Graph. In *SIROCCO 2005*, volume 3499 of *LNCS*, pages 99–114. Springer, 2005.

[10] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree Exploration with Little Memory. *Journal of Algorithms*, 51(1):38–63, April 2004.

[11] S. Dobrev, P. Flocchini, R. Kralovic, P. Ruzicka, G. Prencipe, and N. Santoro. Black Hole Search in Common Interconnection Networks. *Networks*, 47(2):61–71, March 2006.

[12] S. Dobrev, P. Flocchini, R. Kralovic, and N. Santoro. Exploring an Unknown Graph to Locate a Black Hole Using Tokens. In *TCS 2006*, volume 209 of *IFIP International Federation for Information Processing*, pages 131–150. Springer, December 2006.

[13] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Multiple Agents RendezVous in a Ring in Spite of a Black Hole. In *OPODIS 2003*, volume 3144 of *LNCS*, pages 34–46. Springer, 2003.

[14] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a Black Hole in Arbitrary Networks: Optimal Mobile Agents Protocols. *Distributed Computing*, 19(1):1–19, September 2006.

[15] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile Search for a Black Hole in an Anonymous Ring. *Algorithmica*, 48(1):67–90, June 2007.

[16] S. Dobrev, P. Flocchini, and N. Santoro. Improved Bounds for Optimal Black Hole Search with a Network Map. In *SIROCCO 2004*, volume 3104 of *LNCS*, pages 111–122. Springer, 2004.

[17] S. Dobrev, P. Flocchini, and N. Santoro. Cycling Through a Dangerous Network: A Simple Efficient Strategy for Black Hole Search. In *ICDCS 2006*, pages 1–8. IEEE, 2006.

[18] S. Dobrev, R. Kralovic, N. Santoro, and W. Shi. Black Hole Search in Asynchronous Rings Using Tokens. In *CIAC 2006*, volume 3998 of *LNCS*, pages 139–150. Springer, 2006.

[19] S. Dobrev, N. Santoro, and W. Shi. Scattered Black Hole Search in an Ring using Tokens. *Int. Journal of Foundations of Computer Science*, 2007. to appear. Preliminary version in IPDPS 2007.

[20] P. Fraigniaud, L. Gasieniec, D. R. Kowalski, and A. Pelc. Collective Tree Exploration. *Networks*, 48(3):166–177, October 2006.

[21] P. Fraigniaud and D. Ilcinkas. Digraphs Exploration with Little Memory. In *STACS 2004*, volume 2996 of *LNCS*, pages 246–257. Springer Berlin / Heidelberg, 2004.

[22] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation Algorithms for Some Routing Problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

[23] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Approximation Bounds for Black Hole Search Problems. In *OPODIS*, volume 3974 of *LNCS*, pages 261–274. Springer, 2005.

[24] 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, October 2007.

[25] D. Kozen. Automata and Planar Graphs. In *FCT 1979: Proceedings of the Conference on the Fundamentals of Computation Theory*, pages 243–254, 1979.

[26] P. Panaite and A. Pelc. Exploring Unknown Undirected Graphs. *Journal of Algorithms*, 33(2):281–295, November 1999.