

Searching for Black-Hole Faults in a Network Using Multiple Agents*

Colin Cooper¹, Ralf Klasing², and Tomasz Radzik¹

¹ Department of Computer Science, King's College, London WC2R 2LS, UK
{Colin.Cooper, Tomasz.Radzik}@kcl.ac.uk

² LaBRI – Université Bordeaux 1 – CNRS, 351 cours de la Libération,
33405 Talence cedex, France
Ralf.Klasing@labri.fr

Abstract. We consider a fixed communication network where (software) agents can move freely from node to node along the edges. A *black hole* is a faulty or malicious node in the network such that if an agent enters this node, then it immediately “dies.” We are interested in designing an efficient communication algorithm for the agents to identify all black holes. We assume that we have k agents starting from the same node s and knowing the topology of the whole network. The agents move through the network in synchronous steps and can communicate only when they meet in a node. At the end of the exploration of the network, at least one agent must survive and must know the exact locations of the black holes. If the network has n nodes and b black holes, then any exploration algorithm needs $\Omega(n/k + D_b)$ steps in the worst-case, where D_b is the worst case diameter of the network with at most b nodes deleted. We give a general algorithm which completes exploration in $O((n/k) \log n / \log \log n + bD_b)$ steps for arbitrary networks, if $b \leq k/2$. In the case when $b \leq k/2$, $bD_b = O(\sqrt{n})$ and $k = O(\sqrt{n})$, we give a refined algorithm which completes exploration in asymptotically optimal $O(n/k)$ steps.

Keywords: Graph exploration, mobile agent, black hole faults.

1 Introduction

The network search problem which we consider in this paper is motivated by the following scenario. Mobile (software) agents can move through a network of computers, but some host, called *black holes* terminate any agent visiting it. The problem of protecting mobile agents from such malicious hosts has been studied in [6,7,10,11]. We assume that agents are a limited resource, so they should first locate black holes to avoid entering them and dying.

Initially, the agents are at the same *start node* s and know the topology of the whole network, but do not know the number and the location of black holes.

* Research supported in part by Royal Society Grant ESEP 16244, COST Action TIST 293 (GRAAL), and a visiting fellowship from LaBRI/ENSEIRB for Colin Cooper.

Also, no information about black holes is available in the safe nodes of the network (the nodes which are not black holes). Thus, in order to locate a black hole, at least one agent must visit it. An agent entering a black hole disappears there, so later this agent cannot show up where expected by the other agents, or communicate with them in any other expected way. On this basis, the other agents may be able to deduce the exact location of a black hole. We want to design an efficient communication algorithm for the agents to identify all black holes reachable from the start node. (If black holes disconnect the network into separate components, then we can only hope to explore the component which contains the start node.)

The black hole search was studied in [3,4,5] under the scenario of totally asynchronous networks, that is, without assuming any bound on the ratio of the times of two different edge traversals. The authors considered the case of two agents and one black hole. To solve the problem in this setting, the network must be 2-connected. The complexity measure considered is the total number of moves performed by the agents. For arbitrary networks of n nodes, it is shown in [5] that $\Theta(n \log n)$ moves are necessary and sufficient.

We consider synchronous networks and assume that agents can communicate, and exchange their knowledge, only when they meet. They cannot leave any messages in the nodes of the network. In one synchronized step, each agent can either stay in its current host or move to a neighbouring one. An agent X may infer the location of a black hole, if it expects to meet another agent Y , but agent Y does not show up. This model, but with the added restriction that there are only two agents and at most one black hole, has been previously studied in [1,2,8,9]. Those papers give NP-hardness results and approximation algorithms for the problem of calculating an optimal (shortest) traversal schedules.

In this paper, we show two efficient algorithms for black hole search with multiple agents. That is, the number k of initially available agents is an independent parameter. At the beginning of the computation all agents are at a start node s . Let D_b be the maximum distance from s to a node reachable from s in a network obtained from the given network by deleting up to b nodes (the maximum over all possible deletions). If the network has $b \leq k - 2$ black holes (see the next section for justification of considering this bound on the number of black holes), then a black hole search with k agents must have $\Omega((n/k) + D_b)$ steps in the worst case. If $b \leq k/2$, then our first algorithm takes $O((n/k) \log n / \log \log n + bD_b)$ steps in the worst case, while our second algorithm takes $O(n/k)$ steps, provided that additionally $k = O(\sqrt{n})$ and $bD_b = O(\sqrt{n})$. Our algorithms are the first algorithms for searching for black holes with multiple agents with non-trivial upper bounds on the number of steps. Note that k agents can trivially discover up to $k - 1$ black holes in $O(n)$ steps by exploring the network using edges of an arbitrary spanning tree.

In the running time of a black hole search algorithm we count only the number of “traversal” steps, but do not count the computational time of deciding which traversals the agents should take in the current step. In our algorithms, this computational time is polynomial.

2 Preliminaries

The input to our black-hole search problem is an undirected connected graph $G = (V, E)$, a subset of nodes $S \subseteq V$ known to be safe, a *start node* $s \in S$, and a positive integer k . The objective of the problem is to identify black holes in $V \setminus S$ using k agents. The agents are numbered from 1 to k and they are initially at the start node s . The agents move in synchronized steps. In each step, each agent can either wait in its current position $v \in V$, or move to a node adjacent to v . Agents communicate, exchanging their whole knowledge, when they are at the same node at the same step.

For a subset of nodes $W \subseteq V$ and a node $v \in W$, $G[W]$ denotes the subgraph of G induced by W , and $G_v[W]$ denotes the connected component of $G_v[W]$ containing node v .

An *exploration algorithm* works correctly, if it terminates, and at the termination there is at least one surviving agent, all surviving agents know the set of *identified* black holes $B \subseteq V \setminus S$, and they all know that all nodes in $G_s[V \setminus B]$ are safe. Observe that a node can be identified as a black hole or as a safe node only if it can be reached from s along a path of safe nodes. Thus, it is not possible to find out anything about the nodes in other components of $G[V \setminus B]$. Note that in our notation in this paper (unlike in some previous papers on this topic) B stands for the set of *identified* black holes, which is not necessarily the whole set of black holes in the network.

We assume that $k \geq 3$. If there was initially only one agent, then no exploration would be possible since the agent would risk entering a black hole by attempting any movement. The black-hole search problem with two agents has been comprehensively studied before.

Our exploration algorithms work correctly, if there are at most $k - 2$ black holes in G , losing only one agent for each identified black hole. If there are $k - 1$ or more black holes in the network, then the algorithms (slightly modified to stop when only one agent remains) return a set B of $k - 1$ black holes, but the surviving agent may not know whether all nodes in $G_s[V \setminus B]$ are safe (some nodes in $G_s[V \setminus B]$ may be left unexplored, so can be additional black holes, and cannot be explored with one agent).

The running time of an exploration algorithm is the number of steps executed until its termination. We define D_b to be the maximum diameter of $G_s[V \setminus B]$ over all $B \subseteq V$, $|B| \leq b$. Any exploration algorithm runs in the *worst case* placement of the black holes in $\Omega(n/k)$ time, if $|V \setminus S| = \Omega(n)$, and in $\Omega(D_b)$ time, if b black holes are identified. The asymptotic bounds on the performance of our algorithms include this parameter D_b , but we do not consider in this paper the problem of estimating D_b .

For a graph H , $V[H]$ denotes the set of nodes in H . A *subtree* of a rooted tree T is a connected subgraph of T . The root of a subtree H of T is the node in H closest to the root of T . We say that J is the subtree of T spanned by a vertex set X , if J is the smallest subtree of T containing the vertex set X . For a node v in T , the *subtree of T rooted at v* contains v and all its descendants in T .

3 Procedures Used in the Main Algorithms

Let P be a path with node s as one end. Two agents can explore P in the following simple way. The agents move together along the path *probing* unexplored nodes (nodes not known to be safe). That is, if they are to move from a node u to an unexplored node w then one agent moves from u to w and back to u , while the other agent waits in u . If the agent which has moved to w comes back to u , then both agents learn that u is a safe node, so they move there together in the next step. If the probing agent does not come back to u from w , then the waiting agent learns that u is a black hole and goes back to s . The whole exploration takes at most $4d$ steps, where d is the length of P . Next, we describe a natural extension of this simple algorithm from paths to trees. Note that by exploring a path or a tree we mean here a partial exploration, as no nodes in the path/tree which are “behind” the black holes are reached (even if in the overall network there are other edges than the path/tree edges leading to those nodes).

Let T be a tree rooted in a start node s , and S be the set of known safe nodes in T ($s \in S$). Let h denote the height of T and l denote the number of leaves in T . Procedure $\text{EXPLORE TREE}(T, s, S)$ explores T in $O(h)$ steps using $2l$ agents starting from s . (If there are more than $2l$ agents available at the beginning of the computation of this procedure, then still only $2l$ agent are used and the remaining agents wait in s .) The agents are assigned to the leaves of T , two agents per each leaf. The agents move from s towards their leaves probing each unexplored node. More precisely, if a number of agents are to move from a node u to an unexplored node w on their way towards the leaves in the subtree of T rooted at w , then one of the agents moves from u to w and back to u , while the others wait in u . If the selected agent comes back to u , then the waiting agents know that w is safe, so they all move there together. If the selected agent does not come back to u , then the waiting nodes learn that w is a black hole and go back to s . When two agents reach the leaf assigned to them, then they go back to s .

Procedure $\text{EXPLORE TREE}(T, s, S)$ returns the set B of discovered black holes, runs in at most $4h$ steps, loses only $|B|$ agents, and establishes that all nodes in the subtree of T obtained by removing the subtrees rooted in B are safe. At the termination, all surviving agents are back at the start node s . We use procedure $\text{EXPLORE TREE}(T, s, S)$ as a subroutine in our exploration algorithms for general graphs.

Another procedure which we use is $\text{EXPLORE SUBTREES}(T, s, S', \mathcal{H})$. Here T is a tree rooted in a start node s , S' is the set of initially known safe nodes in T ($s \in S'$), and \mathcal{H} is a family of rooted subtrees of T with the following properties.

1. \mathcal{H} covers all nodes in $V[T] \setminus S'$, that is, for each $v \in V[T] \setminus S'$, there is a subtree $H \in \mathcal{H}$ containing v .
2. If a node belongs to two subtrees in \mathcal{H} , then it is the root of one or both of them.
3. For each subtree $H \in \mathcal{H}$, each node on the path in T between s and the root of H , including the root of H , belongs to S' (that is, is initially known to be safe).

Procedure `EXPLORESUBTREES`(T, s, S', \mathcal{H}) explores T using $2|\mathcal{H}|$ agents in the following way. For each subtree in \mathcal{H} , two agents are assigned to explore it, and the exploration of all subtrees is conducted in parallel. The two agents assigned to a subtree $H \in \mathcal{H}$ first walk from s to the root of H (this path is safe by condition 3 above), and then walk along an Euler tour of H probing nodes not in S' . If they have found a black hole in H , then the surviving agent goes back to s . If they explore the whole subtree without finding a black hole, then they both go back to the root. Condition 2 and 3 implies that exploration of all subtrees in \mathcal{H} can be done in parallel and *independently*, since a possible common node of two trees must be safe. Procedure `EXPLORESUBTREES`(T, s, S', \mathcal{H}) terminates in $O(h + \max_{H \in \mathcal{H}} |H|)$ steps, where h is the height of T , and returns the set B of found black holes and the set S'' of established safe nodes ($S' \subseteq S''$). Observe that for each $H \in \mathcal{H}$, this procedure finds a black hole in H (leaving some nodes in H unexplored) or establishes that all nodes in H are safe.

In our algorithms for exploration of arbitrary graphs, we consider only balanced families of subtrees. For $T, S' \subseteq V[T]$, $s \in S'$, and a positive integer x , an x -balanced family \mathcal{H} of subtrees of tree T contains at most x subtrees, satisfies conditions 1 and 2 above, and for each $H \in \mathcal{H}$, $|V[H]| = O(|V[T]|/x)$ and $|V[H] \cap U| = O(|U|/x)$, where $U = V[T] \setminus S'$. An x -balanced family of subtrees is returned by procedure `BALANCEDSUBTREES`(T, s, U, x). We present details of this procedure in Section 6.

4 Algorithm for Arbitrary Networks

We describe our algorithms using the following notation:

- \bar{B} – the set of nodes already identified as black holes,
- $\bar{G} = G_s[V \setminus \bar{B}]$ – the current network,
- \bar{S} – the set of nodes already known to be safe, $S \subseteq \bar{S} \subseteq V[\bar{G}]$,
- \bar{k} – the number of live agents.

Initially, $\bar{B} = \emptyset$, $\bar{G} = G$, $\bar{S} = S$, and $\bar{k} = k$. The details of our first graph exploration algorithm `EXPLOREGRAPH1` are given in Figure 1. The algorithm consists of a number of *rounds*. In each round, the algorithm tries to explore more nodes. Analysing how the number of unexplored nodes decreases in each round will be the basis for bounding the total running time.

Each round consists of two parts. The first part, the “repeat” loop in lines 3–9, establishes a shortest path tree T in \bar{G} to all nodes in $V[\bar{G}] \setminus \bar{S}$ and a balanced family \mathcal{H} of $\lfloor \bar{k}/2 \rfloor$ subtrees which meets also condition 3 given in Section 3. This is done by an iterative process of computing a shortest path tree T , taking the $\lfloor \bar{k}/2 \rfloor$ -balanced family \mathcal{H} of subtrees of T returned by the procedure call `BALANCEDSUBTREES`($T, s, V[\bar{G}] \setminus \bar{S}, \lfloor \bar{k}/2 \rfloor$), and checking if this family satisfies condition 3. This checking is done by calling `EXPLORE TREE`($J, s, \bar{S} \cap V[J]$) for the subtree J of tree T containing only the tree paths from s to the roots of the subtrees in \mathcal{H} . If all nodes in subtree J turn out to be safe, then

```

EXPLOREGRAPH1( $G, s, S, k$ ):
1:  ( $\bar{G}, \bar{S}, \bar{k}, \bar{B}$ )  $\leftarrow$  ( $G, S, k, \emptyset$ );
    { the current graph, the current set of known safe nodes, the number of
    agents which are still alive, and the current set of known black holes; }
2:  while  $\bar{S} \neq V[\bar{G}]$  do
    { one round }
3:    repeat
    { all live agents are in the start node  $s$  }
4:     $T \leftarrow$  a shortest path tree in  $\bar{G}$  from  $s$  to all nodes in  $V[\bar{G}] \setminus \bar{S}$ ;
5:     $\mathcal{H} \leftarrow$  BALANCEDSUBTREES( $T, s, V[\bar{G}] \setminus \bar{S}, \lfloor \bar{k}/2 \rfloor$ );
    {  $\mathcal{H}$  is a balanced family of at most  $\lfloor \bar{k}/2 \rfloor$  subtrees of  $T$  covering
    all nodes not in  $\bar{S}$ ; }
6:     $J \leftarrow$  the subtree of  $T$  spanned by  $s$  and the roots of subtrees in  $\mathcal{H}$ ;
7:     $B_J \leftarrow$  EXPLORETREE( $J, s, \bar{S} \cap V[J]$ );
8:    update  $\bar{G}, \bar{S}, \bar{k}$ , and  $\bar{B}$ ;
9:    until  $V[J] \subseteq \bar{S}$ ; {that is, all nodes in tree  $J$  are safe }
10:   ( $B_{\mathcal{H}}, S_{\mathcal{H}}$ )  $\leftarrow$  EXPLORESUBTREES( $T, s, \bar{S}, \mathcal{H}$ );
11:   update  $\bar{G}, \bar{S}, \bar{k}$ , and  $\bar{B}$ ;
12: end_while;
13: return  $\bar{B}$ .
    
```

Fig. 1. EXPLOREGRAPH1— a graph exploration algorithm with $O(\log n / \log \log n)$ rounds

we have found required T and \mathcal{H} . If a black hole has been found in J , then the process continues by computing new T and \mathcal{H} .

For T and \mathcal{H} established in the first part of a round, in the second part the subtrees in \mathcal{H} are explored by calling procedure EXPLORESUBTREES($T, s, \bar{S}, \mathcal{H}$) (line 10). The algorithm proceeds to the next round, if there are still nodes left in $V[\bar{G}] \setminus \bar{S}$ (unexplored nodes).

If the number of black holes in G is $b \leq k - 2$, then algorithm EXPLOREGRAPH1 terminates correctly, and at the termination there are $k - b$ surviving agents and all nodes in $V[\bar{G}]$ have been identified as safe. The running time, however, can be as high as $\Theta(n)$, even if D_b is small. We show next a bound on the running time of EXPLOREGRAPH1, if $b \leq k/2$.

Theorem 1. *Let $G = (V, E)$ be an n -node connected graph, $S \subseteq V$ be the set of known safe nodes in G , $s \in S$, and an integer $k \geq 3$. If there are initially k agents at node s and the number of black holes in G is $b \leq k/2$, then the algorithm EXPLOREGRAPH1(G, s, S, k) completes the search for black holes in $O((n/k) \log n / \log \log n + bD_b)$ steps.*

Proof. We refer to the description of algorithm EXPLOREGRAPH1 given in Figure 1. At each point of the computation of algorithm EXPLOREGRAPH1, the number of live agents is $\bar{k} \geq k - b \geq k/2$. The agents walk through the network only when executing procedure EXPLORETREE in line 7 and procedure EXPLORESUBTREES in line 10. All other computation, including procedure BALANCEDSUBTREES, is done locally from the knowledge of the network and already located black holes, and is therefore not counted in the time complexity.

One execution of procedure EXPLORETREE in line 7 takes $O(h)$ steps, where h is the height of tree J . Since J is a shortest path tree in \bar{G} and the diameter of \bar{G} is at most D_b , so $h \leq D_b$. If no black hole is found during the current execution of procedure EXPLORETREE, then the inner (“repeat”) loop ends and procedure EXPLORESUBTREES is executed. If no black hole is found during this execution of EXPLORESUBTREES, then the black-hole search is completed and the whole algorithm terminates. Thus, if the current execution of procedure EXPLORETREE is not the last one, then between the beginning of the current execution of procedure EXPLORETREE and the beginning of the next execution of this procedure at least one black hole is found. Hence there are at most $b + 1$ executions of procedure EXPLORETREE throughout the whole execution of algorithm EXPLOREGRAPH1.

Since procedure EXPLORESUBTREES is applied in line 10 to subtrees of sizes $O(n/\bar{k})$, then the general bound on the running time of this procedure implies that its execution in line 10 takes $O((n/\bar{k}) + D_b) = O((n/k) + D_b)$ steps. Procedure EXPLORESUBTREES is executed once in each round (each iteration of the outer “while” loop) of algorithm EXPLOREGRAPH1. Let q denote the number of rounds. At least one new black hole is found in each round other than the last one, so $q \leq b + 1$. We show that we also have $q = O(\log n / \log \log n)$.

For round i , let \mathcal{H}_i denote the family of subtrees used in procedure EXPLORESUBTREES in line 10, and let b_i denote the number of black holes found by this procedure call. We have $\sum_{i=1}^q b_i \leq b$. Let $U_i = V[\bar{G}] \setminus \bar{S}$, $u_i = |U_i|$, and $k_i = \bar{k}$ before the last call to procedure BALANCEDSUBTREES in this round (in line 5), which calculates \mathcal{H}_i . A node v belongs to U_{i+1} , only if v belongs to U_i , v belongs to a subtree in \mathcal{H}_i containing a black hole, and v is not the root of this subtree. There are exactly b_i subtrees in \mathcal{H}_i which have black holes, and each of these subtrees contains at most $6u_i/\lfloor k_i/2 \rfloor$ nodes other than the root which belong to U_i (see Lemma 1, part 3). Since $k_i \geq k/2$ and $k \geq 3$, then $\lfloor k_i/2 \rfloor \geq k/6$, so

$$u_{i+1} \leq b_i \frac{6u_i}{\lfloor k_i/2 \rfloor} \leq \frac{36b_i}{k} u_i. \quad (1)$$

This implies

$$1 \leq u_q \leq \left(\frac{36}{k}\right)^{q-1} \left(\prod_{i=1}^{q-1} b_i\right) u_1 \leq \left(\frac{36b}{k(q-1)}\right)^{q-1} u_1 \leq \left(\frac{18}{q-1}\right)^{q-1} u_1, \quad (2)$$

using $\prod_{i=1}^{q-1} b_i \leq [(\sum_{i=1}^{q-1} b_i)/(q-1)]^{q-1}$. Inequalities (2) and $u_1 \leq n$ imply that $q = O(\log n / \log \log n)$.

Thus, all executions of procedure EXPLORETREE take together $O(bD_b)$ steps and all executions of procedure BALANCEDSUBTREES take together $O((n/k + D_b) \min\{b, \log n / \log \log n\})$ steps. Hence the execution of algorithm EXPLOREGRAPH1 takes $O((n/k) \log n / \log \log n + bD_b)$ steps. \square

5 An Algorithm for Networks with Bounded Diameter

The running time of algorithm EXPLOREGRAPH1 is $O((n/k) \log n / \log \log n)$, if there are at most $k/2$ black holes and D_b is small. Our second algorithm EXPLOREGRAPH2 has asymptotically optimal $O(n/k)$ running time, provided that some bounds on k and D_b are satisfied.

The details of algorithm EXPLOREGRAPH2 are given in Figure 2. This algorithm has only one round (lines 2–16), instead of $O(\log n / \log \log n)$ rounds of algorithm EXPLOREGRAPH1. The nodes which remain unexplored after this single round are then explored in the second part of the algorithm using procedure EXPLORETREE (lines 17–21). To guarantee that not too many unexplored nodes are left for the second part of the algorithm, in the first part of the algorithm we consider x -balanced families of subtrees for a parameter $x \geq k$. The number of subtrees in such a family \mathcal{H} can be greater than $k/2$, so we cannot check if \mathcal{H} satisfies condition 3 by only one call to procedure EXPLORETREE, and we cannot explore subtrees in \mathcal{H} by only one call to procedure EXPLORESUBTREES, as we did in one round of algorithm EXPLOREGRAPH1. Instead the first task is accomplished by a sequence of calls to procedures EXPLORETREE (lines 6–10), and the second task is accomplished by a sequence of calls to procedure EXPLORESUBTREES (lines 12–16).

If the number of black holes in G is $b \leq k - 2$, then algorithm EXPLOREGRAPH2 terminates correctly, and at the termination there are $k - b$ surviving agents and all nodes in $V[\bar{G}]$ have been identified as safe. We show next a bound on the running time of EXPLOREGRAPH2, if $b \leq k/2$.

Theorem 2. *Let $G = (V, E)$ be an n -node connected graph, $S \subseteq V$ be the set of known safe nodes in G , $s \in S$, and integers $x \geq k \geq 3$. If there are initially k agents at node s and the number of black holes in G is $b \leq k/2$, then algorithm EXPLOREGRAPH2(G, s, S, k, x) terminates in $O((n/k) + (bD_b/k)(x + (n/x)))$ steps.*

Proof. We refer to the description of algorithm EXPLOREGRAPH2 given in Figure 2. Throughout the computation of algorithm EXPLOREGRAPH2, the number of live agents is $\bar{k} \geq k - b \geq k/2$. We bound the total number of steps taken by procedures EXPLORETREE and EXPLORESUBTREES. All other computation is done locally and is not counted in the time complexity.

Each execution of procedure EXPLORETREE in lines 8 and 19 takes $O(D_b)$ steps. The outer “repeat” loop in lines 2–11 has at most $b + 1$ iterations, since at least one black hole is found in each iteration other than the last one. In each iteration of this loop, there are at most $O(x/k)$ iterations of the inner “repeat”

```

EXPLOREGRAPH2( $G, s, S, k, x$ ):
1:  $(\bar{G}, \bar{S}, \bar{k}, \bar{B}) \leftarrow (G, S, k, \emptyset)$ ;
2: repeat
    { all live agents are in the start node  $s$  }
3:  $T \leftarrow$  a shortest path tree in  $\bar{G}$  from  $s$  to all nodes in  $V[\bar{G}] \setminus \bar{S}$ ;
4:  $\mathcal{H} \leftarrow$  BALANCEDSUBTREES( $T, s, V[\bar{G}] \setminus \bar{S}, x$ );
5:  $J \leftarrow$  the subtree of  $T$  spanned by  $s$  and the roots of the subtrees in  $\mathcal{H}$ ;
6: repeat
7:    $Q \leftarrow$  subtree of  $J$  spanned by  $s$  and next  $\lfloor \bar{k}/2 \rfloor$  leaves in  $J$ ;
8:    $B_Q \leftarrow$  EXPLORETREE( $Q, s, \bar{S} \cap V[Q]$ );
9:   update  $\bar{G}, \bar{S}, \bar{k}$ , and  $\bar{B}$ ;
10:  until  $V[J] \subseteq \bar{S}$  or  $B_Q \neq \emptyset$ ;
11: until  $V[J] \subseteq \bar{S}$ ; {that is, all nodes in tree  $J$  are safe }
12: repeat
13:    $\mathcal{F} \leftarrow$  next  $\lfloor \bar{k}/2 \rfloor$  subtrees in  $\mathcal{H}$ ;
14:    $(B_{\mathcal{F}}, S_{\mathcal{F}}) \leftarrow$  EXPLORESUBTREES( $T, s, \bar{S}, \mathcal{F}$ );
15:   update  $\bar{G}, \bar{S}, \bar{k}$ , and  $\bar{B}$ ;
16: until all subtrees in  $\mathcal{H}$  have been considered;
17: while  $\bar{S} \neq V[\bar{G}]$  do
18:    $T \leftarrow$  a shortest path tree in  $\bar{G}$  from  $s$  to  $\lfloor \bar{k}/2 \rfloor$  nodes in  $V[\bar{G}] \setminus \bar{S}$ ;
19:    $B_T \leftarrow$  EXPLORETREE( $T, s, \bar{S} \cap V[T]$ );
20:   update  $\bar{G}, \bar{S}, \bar{k}$ , and  $\bar{B}$ ;
21: end.while;
22: return  $\bar{B}$ .

```

Fig. 2. EXPLOREGRAPH2: a graph exploration algorithm with a single round followed by an additional “cleaning-up” process using algorithm EXPLORETREE

loop. Thus, all executions of procedure EXPLORETREE in line 8 take together $O(xbD_b/k)$ steps.

Since procedure EXPLORESUBTREES is applied in line 14 to subtrees of sizes $O(n/x)$, then the general bound on the running time of this procedure implies that its one execution takes $O((n/x) + D_b)$ steps. There are $O(x/k)$ iterations of the “repeat” loop in lines 12–16, so all executions of procedure EXPLORESUBTREES take together $O((n/k) + (x/k)D_b)$ steps.

At the termination of the loop in lines 12–16, each node in $V[\bar{G}] \setminus \bar{S}$ belongs to a subtree in \mathcal{H} which has a black hole. There are at most b subtrees in \mathcal{H} which have black holes, and each of these subtrees contains $O(n/x)$ nodes. Hence $|V[\bar{G}] \setminus \bar{S}| = O(bn/x)$ at the beginning of the “while” loop in lines 17–21. Each execution of

procedure EXPLORETREE in line 19 other than the last one decreases the number of nodes in $V[\bar{G}] \setminus \bar{S}$ by at least $\lfloor \bar{k}/2 \rfloor$ or finds at least one new black hole. Therefore there are $O((bn)/(xk) + b)$ executions of procedure EXPLORETREE in line 19, and all these executions take together $O(D_b((bn)/(xk) + b))$ steps.

We conclude that the number of steps required by algorithm EXPLOREGRAPH2 is

$$O\left(\frac{xbD_b}{k}\right) + O\left(\frac{n}{k} + \frac{xD_b}{k}\right) + O\left(\frac{bnD_b}{xk} + bD_b\right) = O\left(\frac{n}{k} + \frac{bD_b}{k}\left(x + \frac{n}{x}\right)\right).$$

□

Corollary 1. *Let $G = (V, E)$ be an n -node connected graph, $S \subseteq V$ be the set of known safe nodes in G , $s \in S$, and an integer k such that $3 \leq k = O(\sqrt{n})$. If there are initially k agents at node s , the number of black holes in G is $b \leq k/2$, and $bD_b = O(\sqrt{n})$, then for $x = \max\{k, \lfloor \sqrt{n} \rfloor\}$, algorithm EXPLOREGRAPH2(G, s, S, k, x) terminates in $O(n/k)$ steps.*

6 Computing a Family of Balanced Subtrees

For a tree T rooted at node s , a subset of nodes $U \subseteq V[T]$ and a positive integer x , procedure BALANCEDSUBTREES(T, s, U, x) computes an x -balanced family \mathcal{H} of subtrees of T . The computation proceeds in iterations. In each iteration, a subtree H is cut off from the tree and added to \mathcal{H} .

Let \bar{T} denote the current tree, and for a node $v \in \bar{T}$, let $size(v)$ and $weight(v)$ denote the number of nodes in the subtree of \bar{T} rooted at v and the number of nodes in this subtree which belong to U , respectively. Similarly, for a subtree H of tree T , let $size(H)$ and $weight(H)$ denote the number of nodes in H and the number of nodes in H which are in U , respectively. In one iteration of procedure BALANCEDSUBTREES, a node v in \bar{T} is selected, which is a lowest (furthest from the root) node in \bar{T} such that $size(v) \geq 3n/x$ or $weight(v) \geq 3u/x$. Thus, for each child w of v , $size(w) < 3n/x$ and $weight(w) < 3u/x$. Order the children of node v in an arbitrary way. The subtree H selected in this iteration is obtained by taking node v as the root and adding the subtrees rooted in the first q children of node v which make $size(H) \geq 3n/x$ or $weight(H) \geq 3u/x$. This subtree H is added to \mathcal{H} and all nodes in H other than the root node v are removed from \bar{T} . Node v is also removed from \bar{T} , if v becomes a leaf. The computation continues until the remaining tree \bar{T} has less than $3n/x$ nodes and less than $3u/x$ nodes in U . If the final tree \bar{T} is not empty, then it is added to \mathcal{H} .

Next, we show that the family \mathcal{H} of subtrees of T computed by BALANCEDSUBTREES(T, s, U, x) is indeed x -balanced.

Lemma 1. *For an n -node tree T rooted at a node s , a subset U of u nodes in T , and a positive integer $x < u$, procedure BALANCEDSUBTREES(T, s, U, x) returns a family \mathcal{H} of subtrees of T with the following properties.*

1. *The subtrees in \mathcal{H} cover all nodes in U , that is, for each $v \in U$, there is a subtree $H \in \mathcal{H}$ containing v .*

```

BALANCEDSUBTREES( $T, s, U, x$ ):
1:  $(n, u) \leftarrow$  the number of nodes in  $T$  and  $U$ , respectively;
2:  $\mathcal{H} \leftarrow$  empty family of subtrees of  $T$ ;
3:  $\bar{T} \leftarrow T$ ; { the remaining tree to be covered }
4: while  $|V[\bar{T}]| \geq 3n/x$  or  $|V[\bar{T}] \cap U| \geq 3u/x$  do
5:   let  $size(v)$  and  $weight(v)$  be the number of nodes in the subtree of  $\bar{T}$ 
      rooted at  $v$  and the number of nodes in this subtree which are in  $U$ ;
6:    $v \leftarrow$  a lowest (furthest from the root) node in  $\bar{T}$  such that  $size(v) \geq 3n/x$ 
      or  $weight(v) \geq 3u/x$  (thus for each child  $w$  of  $v$ ,  $size(w) < 3n/x$ 
      and  $weight(w) < 3u/x$ );
7:    $H \leftarrow$  a subtree of  $\bar{T}$  obtained by taking node  $v$  as the root and adding sub-
      trees rooted at children of  $v$  until  $size(H) \geq 3n/x$  or  $weight(H) \geq$ 
       $3u/x$ ;
      { thus  $size(H) \leq (6n/x) + 1$  and  $weight(H \setminus \{v\}) \leq 6u/x$ ; }
8:   add  $H$  to  $\mathcal{H}$ ;
9:   update  $\bar{T}$ :
      remove from  $\bar{T}$  all nodes in  $H$  other than the root node  $v$ ;
      remove also  $v$ , if it becomes a leaf;
10: end_while;
11: if  $\bar{T}$  is not empty then add  $\bar{T}$  to  $\mathcal{H}$ ;
12: return  $\mathcal{H}$ .

```

Fig. 3. Computing balanced subtrees of a tree T covering the nodes in a given set U

2. If a node belongs to two subtrees in \mathcal{H} , then it is the root of one or both of them.
3. Each subtree in \mathcal{H} has at most $(6n/x) + 1$ nodes and at most $6u/x$ nodes other than the root which belong to U .
4. There are at most x subtrees in \mathcal{H} .

Proof. Parts 1, 2, and 3 of the lemma are easy consequences of the way procedure BALANCEDSUBTREES constructs the subtrees in \mathcal{H} . We give a detailed argument only for part 4 of the lemma.

Let \mathcal{H}_1 be the family of subtrees in \mathcal{H} with sizes at least $3n/x$, \mathcal{H}_2 be the family of subtrees in \mathcal{H} with weights at least $3u/x$, $h_1 = |\mathcal{H}_1|$, and $h_2 = |\mathcal{H}_2|$. Since each subtree in \mathcal{H} , except possibly the last one added in line 11, has size at least $3n/x$ or weight at least $3u/x$, then $|\mathcal{H}| \leq h_1 + h_2 + 1$. We have

$$\frac{3n}{x}h_1 \leq \sum_{H \in \mathcal{H}_1} |H| \leq h_1 + |\bigcup \mathcal{H}'| \leq h_1 + n, \quad (3)$$

where the second inequality holds because each node can be a non-root node in at most one tree in \mathcal{H} . Similarly we have

$$\frac{3u}{x}h_2 \leq \sum_{H \in \mathcal{H}_2} weight(H) \leq h_2 + u. \tag{4}$$

Inequality (3) implies that

$$h_1 \leq \frac{x}{3 - x/n} < \frac{x}{2},$$

so $h_1 \leq (x - 1)/2$, as x is an integer. Similarly (4) implies that $h_2 \leq (x - 1)/2$. Therefore,

$$|\mathcal{H}| \leq h_1 + h_2 + 1 \leq x. \quad \square$$

7 Conclusions

We showed two efficient algorithms for black hole search with multiple agents. If there are k agents and the network has n nodes and b black holes, then any exploration algorithm needs $\Omega(n/k + D_b)$ steps in the worst-case, where D_b is the worst case diameter of the network with at most b nodes deleted. We gave a general algorithm which completes exploration in $O((n/k) \log n / \log \log n + bD_b)$ steps for arbitrary networks, if $b \leq k/2$. In the case where $b \leq k/2$, $bD_b = O(\sqrt{n})$ and $k = O(\sqrt{n})$, we gave a refined algorithm which completes exploration in asymptotically optimal $O(n/k)$ steps. Our algorithms are the first algorithms for searching for black holes with multiple agents with non-trivial upper bounds on the number of steps.

Even though our second algorithm is asymptotically optimal for restricted values of k in networks with bounded diameter, it remains a question for further research whether an algorithm can be devised that is asymptotically optimal for general k and/or for arbitrary networks. Another important issue is to derive good bounds on the involved constants for practical implementations. Moreover, in this paper, we only considered the case when the topology of the whole network is known in advance. It would be interesting to consider the case when initially no, or only partial, information about the network is available.

Acknowledgements

The authors would like to thank the anonymous referees for their very helpful comments and suggestions.

References

1. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in tree networks. In *Proceedings of 8th International Conference on Principles of Distributed Systems (OPODIS 2004)*, pages 34–35, 2004. Also: Springer LNCS vol. 3544, pages 67-80, also to appear as “Searching for a Black Hole in Synchronous Tree Networks” in *Combinatorics, Probability and Computing*.

2. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71(2-3):229–242, 2006.
3. S. Dobrev, P. Flocchini, R. Královic, G. Prencipe, P. Ruzicka, and N. Santoro. Black hole search in common interconnection networks. *Networks*, to appear. Preliminary version under the title “Black Hole Search by Mobile Agents in Hypercubes and Related Networks” in *Proc. 6th Int. Conf. on Principles of Distributed Systems (OPODIS 2002)* pages 169-180, 2002.
4. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, to appear. Preliminary version in *Proc. 15th Int. Symposium on Distributed Computing (DISC 2001)*, Springer LNCS vol. 2180, pages 166-179, 2001.
5. 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. Preliminary version in *Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 153-161, 2002.
6. F. Hohl. Time limited black box security: Protecting mobile agents from malicious hosts. In G. Vigna, editor, *Proceedings of Conference on Mobile Agent and Security*, Springer LNCS vol. 1419, pages 90–111, 1998.
7. F. Hohl. A framework to protect mobile agents by using reference states. In *Proc. 20th Int. Conf. on Distributed Computing Systems (ICDCS '00)*, pages 410–417, 2000.
8. R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Approximation bounds for black hole search problems. In *Proceedings of 9th International Conference on Principles of Distributed Systems (OPODIS 2005)*, Springer LNCS vol. 3974, to appear.
9. R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary graphs. In *Proc. 12th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO 2005)*, Springer LNCS vol. 3499, pages 200–215, 2005. Also: *Theoretical Computer Science*, to appear.
10. S. Ng and K. Cheung. Protecting mobile agents against malicious hosts by intention of spreading. In H. Arabnia, editor, *Proc. Int. Conf. on Parallel and Distributed Processing and Applications (PDPTA '99) Vol. II*, pages 725–729, 1999.
11. T. Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. In *Proc. Conf. on Mobile Agent Security*, Springer LNCS vol. 1419, pages 44–60, 1998.