

Locating and repairing faults in a network with mobile agents^{*}

Colin Cooper^{1**}, 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, undirected, known network and a number of “mobile agents” which can traverse the network in synchronized steps. Some nodes in the network may be faulty and the agents are to find the faults and repair them. The agents could be software agents, if the underlying network represents a computer network, or robots, if the underlying network represents some potentially hazardous physical terrain. Assuming that the first agent encountering a faulty node can immediately repair it, it is easy to see that the number of steps necessary and sufficient to complete this task is $\Theta(n/k + D)$, where n is the number of nodes in the network, D is the diameter of the network, and k is the number of agents. We consider the case where one agent can repair only one faulty node. After repairing the fault, the agent dies. We show that a simple deterministic algorithm for this problem terminates within $O(n/k + D \log f / \log \log f)$ steps, where $f = \min\{n/k, n/D\}$, assuming that the number of faulty nodes is at most $k/2$. We also demonstrate the worst-case asymptotic optimality of this algorithm by showing a network such that for any deterministic algorithm, there is a placement of $k/2$ faults forcing the algorithm to work for $\Omega(n/k + D \log f / \log \log f)$ steps.

Keywords: Distributed computing, Graph exploration, Mobile agents

1 Introduction

The *black hole search* problems, which have been recently extensively studied, assume that a mobile agent traversing a network is terminated immediately upon entering a faulty node. The task for a group of agents is to explore the network to identify all such faulty nodes, called black holes. In this paper we consider a weaker type of faults: agents are able to repair them, but one agent

^{*} Partially supported by the project ALPAGE of the ANR “Masse de données: Modélisation, Simulation, Applications”, the project CEPAGE of INRIA, the EC COST-TIST Action 293 “Graphs and Algorithms in Communication Networks” (GRAAL) and Action 295 “Dynamic Communication Networks” (DYNAMO), and the Royal Society IJP grant “Algorithms to find dense clusters in large networks”.

^{**} Partially supported by the UK EPSRC grant EP/D059372/1

can repair only one faulty node. The task for a group of agents is now to explore the network to repair all faulty nodes. In this section we first review the black hole search model and introduce in more detail our repairable hole model. Then we summarise our results and compare them with previous relevant results.

Black-hole faults. The black hole search problems have been motivated by the following scenario. Mobile software agents can move through a network of computers, but some hosts (black holes) terminate any agent visiting it. The problem of protecting mobile agents from such malicious hosts has been studied in [11, 12, 15, 16], with the focus on protecting sensitive information which mobile agents may carry.

From a more theoretical point of view, the researchers have investigated the problem of agents co-operatively exploring the network to identify the locations of black holes. Some of the agents might die, but the surviving ones should learn where the black holes are (to avoid them in the future). This problem was initially considered in the *asynchronous* model, and the majority of the results are for this model. An efficient protocol for two agents locating one black hole in an asynchronous ring network was presented in [6] and was extended to arbitrary two-connected networks in [7]. In subsequent research special network topologies were considered [4] and restrictions on communication between the agents were investigated, replacing the initially used whiteboard model with communication by means of a limited number of identical pebbles [5, 8].

In the *synchronous* model, which we consider in this paper, the agents traverse the network in globally timed *steps*. In each step each agent can perform (unlimited) local computation, which includes exchanging information with other agents who are at the same step in the same node, and can then move to a neighbouring node (or remain in the same node) [1–3, 13, 14]. Initially, all 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. 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, but an agent entering a black hole disappears completely. The agents can communicate only when they meet, not being allowed to leave any messages in any form at the nodes. An agent learns that a node v is not a black hole either by visiting it (and surviving) or by meeting another agent who has already visited that node (and survived). An agent may deduce that a node v is a black hole if this agent is supposed to meet another agent, at some node other than v , but that other agent does not show up. The objective is to design a communication algorithm for the agents to identify all black holes reachable from the start node, minimizing the loss of agents and the number of steps required.

Most of the research on the synchronous black hole search problems has been concerned so far with the special case of two agents exploring a network which may have at most one black hole. The problem is to compute an “exploration schedule” for the agents which has the minimum possible number of steps. The first results regarding the computational hardness of this problem and approxi-

mation algorithms were presented in [2, 3] and were subsequently improved in [13, 14]. A more general case of locating black holes with k agents, where $k \geq 2$ is a parameter, was considered in [1]. The recent survey paper [9] discusses both asynchronous and synchronous models, various variants of the black hole search problem and solution techniques.

Repairable faults. In reality many types of faults can be fixed after some amount of trying, and with the expenditure of some effort. Thus there is a spectrum of problems with unfixable faults (black holes) at one end, and fixable faults (holes) at the other. If faults can be fixed, then we have to decide what is the appropriate way of modeling the cost of repairing a fault. For example, an agent fixing a fault may not be able to do anything else: the fault has been fixed but the agent has to be sacrificed. In other scenarios only the “content” part of the agent (the “repair kit”) may have to be sacrificed to fix a fault, while the “carrier” part can remain mobile and can communicate with other agents, or can return to the starting node (the “depot”) to pick up another repair kit.

Scenarios when the agent is sacrificed include robots traveling a road network, seeking to trigger land-mines, and software agents moving from node to node in a computer network to repair faults. In the latter example, the software agent is executable by the runtime environment at a faulty node, fixing that fault but permanently remaining at the node. A physical example where the contents of the agent are consumed, but the carrier survives, is that of trucks with loads of gravel travelling a road network in order to fill holes caused by, for example, flash flooding.

In this paper we consider the following synchronous model of repairable holes. All agents are initially in the start (or *source*) node s and move through the network synchronously, traversing one edge in one time step. If an agent encounters a hole at a vertex v , it will sacrifice itself to repair it. After the repair, which is completed instantaneously, the node functions normally and other agents can pass through it as if the fault never existed. The first agent encountering a given hole must repair it. If two or more agents encounter the same hole at the same time, one of them repairs it, while the other agents can continue with their exploration. Given a network (the whole topology is known in advance), node s and k agents, the aim is to design an *exploration algorithm*, which repairs all holes in the network within a small number of steps.

Our results. Since an exploration algorithm must ensure that each node is visited by at least one agent, then $\Omega(n/k + D)$ is an obvious lower bound on the required number of steps, where n and D are the number of nodes and the diameter of the network, respectively. We show in Section 3 that a simple algorithm completes exploration in $O(n/k + D \log f / \log \log f)$ steps, where $f = \min\{n/k, n/D\}$, if there are at most $k/2$ holes. In Section 4 we prove that this algorithm is asymptotically optimal in the worst case. We do this by showing a tree network $T(n, d)$ with $\Theta(n)$ nodes and diameter $\Theta(d)$, for any $n \geq d \geq 1$, such that for any deterministic exploration algorithm, there is a placement of at most $k/2$ holes in $T(n, d)$, for any $n \geq k \geq 2$, which forces the algorithm to run

for $\Omega(n/k + d \log f / \log \log f)$ steps. The assumption that the number of holes is at most $k/2$ and possible generalisations are discussed in Section 5.

The previous work which is most closely related to our results are the multi-agent black-hole search algorithms which we gave in [1]. Our repairable-hole model is stronger than the black-hole model, so multi-agent black-hole search algorithms can be adapted to the repairable hole model in a natural way. The two black-hole search algorithms from [1] adapted to the repairable-hole model run in $O((n/k) \log n / \log \log n + kD)$ and $O(n/k + D\sqrt{n})$ steps. No lower bounds were given in [1] other than the obvious $\Omega(n/k + D)$.

2 Preliminaries

The input is an undirected graph $G = (E, V)$ representing the network, a source node $s \in V$ and an integer $k \geq 2$, which is the number of available agents. We assume that n , the number of nodes in G , is sufficiently large, and the diameter of G is $D \geq 5$. The latter assumption can be dropped by replacing in the algorithm and its analysis D with $\bar{D} = \max\{D, 5\}$. We denote by b the (unknown) initial number of holes and assume that $b \leq k/2$.

A connected subgraph S of a tree T is called a subtree of T . If T is a rooted tree, then a subtree S is rooted at the node of S closest to the root of T . The size of a subtree S is defined as the number of nodes in S and is denoted by $|S|$. If T is rooted, then for a node v in T , the subtree of T rooted at v is the subtree of T spanning node v and all its descendants. During exploration of a network, the *active agents* are the agents which are still alive, that is, the agents who have not repaired any hole yet.

3 Exploration algorithm

A natural approach to exploring a network with multiple agents is first to compute a cover of the network with connected regions, and then to have separate agents explore separate regions in parallel. Considering regions of size $\Theta(D)$ seems to be a good idea since an agent may need anyway $\Theta(D)$ steps to reach a region. We describe below a simple way of computing $q = O(n/D)$ regions, each of size $O(D)$, which cover the whole network.

For the problem of exploring a network which does not have any faults/holes, with the objective of having each node visited by at least one agent, such regions are the basis of a simple, asymptotically optimal $\Theta(n/k + D)$ -steps algorithm. If $k < q$, then each agent explores $q/k = O(n/(kD))$ regions and exploration of one region takes $O(D)$ time (that is, each agent explores $O(n/k)$ nodes, and whole exploration is completed in $\Theta(n/k)$ steps). If $k \geq q$, then q agents explore the q regions in parallel in $\Theta(D)$ steps. The other agents may remain idle since this asymptotic bound cannot be improved.

Our algorithm, for our model where each agent can repair only one hole, consists of a sequence of rounds, and each round is similar to the exploration scheme sketched above. After each round, the active agents return to the start

node s and learn which regions may still contain (unrepaired) holes. These are the regions from which no agent returned. In the next round, the active agents are sent in equal size groups to explore these remaining regions in parallel. We describe below the details of the algorithm and give in Theorem 1 a worst-case bound on its running time (the number of steps).

Computation of regions. The algorithm first computes a breadth-first tree T of G rooted at s . Then it computes subtrees S_1, S_2, \dots, S_q of tree T with the following properties:

1. each subtree S_i has size at most D ;
2. subtrees S_1, S_2, \dots, S_q cover tree T , that is, each node of T belongs to at least one subtree S_i ;
3. $q = O(n/D)$.

These subtrees define the regions of the network and can be computed by the following straightforward iterative process. Let $T_1 = T$. At the beginning of iteration i , $i \geq 1$, tree T is covered by subtrees S_1, \dots, S_{i-1} and T_i . The subtree T_i is rooted at s and represents the remaining part of tree T which has yet to be covered. If the size of T_i is at most D , then we set $S_i = T_i$, $q = i$, and the covering of T has been computed. Otherwise, when the size of T_i is greater than D , let v_i be a node in T_i such that the size of the subtree of T_i rooted at v_i is at least D but the size of each subtree of T_i rooted at a child of v_i is less than D . Let w_1, w_2, \dots, w_j be the children of node v_i ordered according to the sizes of the subtrees of T_i rooted at these nodes, starting from the largest. Let r be the first index such that the sum of the sizes of the subtrees rooted at nodes w_1, w_2, \dots, w_r is at least $D/2$. Set S_i to the subtree of T_i comprising node v_i and the subtrees rooted at nodes w_1, w_2, \dots, w_r . If S_i includes all children of v_i in T_i (that is, if S_i is the subtree of T_i rooted at v_i), then tree T_{i+1} is obtained from T_i by cutting off subtree S_i (observe that in this case v_i cannot be the root of T_i , so $T_{i+1} \neq \emptyset$). Otherwise, tree T_{i+1} is obtained from T_i by cutting off the subtrees rooted at nodes w_1, w_2, \dots, w_r . Node v_i and its subtrees rooted at nodes w_{r+1}, \dots, w_j remain in T_{i+1} .

It should be clear that the subtrees S_1, S_2, \dots, S_q constructed in this way satisfy Properties 1 and 2. If $i \leq q - 1$, then S_i has at least $D/2 + 1$ nodes. If a node in T belongs to two different subtrees S_r and S_j , $r \neq j$, then it must be the root of at least one of them. Thus sets $V(S_i) \setminus \{v_i\}$, for $i = 1, 2, \dots, q$, are pairwise disjoint and each of them other than the last one has at least $D/2$ nodes. This implies $(D/2)(q - 1) < n$, so $q < (2n)/D + 1$ and Property 3 is satisfied as well.

Exploration. In our exploration algorithm, the agents move through the graph only along the edges of tree T . The exploration consists of two phases, and each of them consists of rounds. At the beginning of one round all active agents are at the source node s . They are partitioned into groups and each group explores during this round one of the *active trees* S_i . Initially all trees S_i are active. A group of l agents explores a tree S_i by first walking together along the path in

T from s to the root of S_i , then fully traversing S_i in $O(|S_i|)$ steps (say, by following an Euler tour around tree S_i , which passes twice along each edge of S_i), and finally walking back to s . All agents in one group keep moving together from node to node. If they encounter a hole which has not been repaired yet (either on the path from s to the root of S_i or within S_i), then one of them repairs it, while the other agents move on. If two or more groups meet at a hole, then one agent from one group (an arbitrary agent from an arbitrary group) repairs it.

If at least one agent from the group exploring tree S_i returns back to s , then all holes in S_i and on the path in T from s to S_i have been repaired, and tree S_i is no longer active. If no agent from this group returns to s , then tree S_i remains active, but we know that at least l additional holes must have been repaired.

During phase 1 of the algorithm, each tree S_i is explored with only one agent (single-agent groups). More specifically, in round 1 of phase 1, the k agents explore in parallel trees S_1, S_2, \dots, S_k , one agent per tree. In round 2, the $k' \leq k$ agents active at the beginning of this round explore in parallel the next k' trees $S_{k+1}, S_{k+2}, \dots, S_{k+k'}$, and so on, until each tree has been explored once. If $k \geq q$, then there is only one round in phase 1. At the end of phase 1, there are at most b remaining active trees, because for each of these trees at least one hole must have been repaired (either in this tree or on the path from s to this tree).

We now describe phase 2 of the exploration. Let k_j, m_j and b_j denote the number of active agents, the number of active trees and the (unknown) number of remaining holes, respectively, at the beginning of round $j, j \geq 1$. During round j , the k_j active agents are partitioned into groups of size $\lfloor k_j/m_j \rfloor$ (some active agents may be left idle, if k_j/m_j is not an integer), and the groups explore in parallel the remaining m_j active trees, one group per tree. If a tree S_i remains active at the end of this round, then each of the $\lfloor k_j/m_j \rfloor$ agents assigned to this tree must have repaired one hole. Thus at least $m_{j+1} \lfloor k_j/m_j \rfloor$ additional holes are repaired during round j . The exploration ends when no active tree is left.

Theorem 1. *Assuming $k \geq 2b$, the exploration algorithm runs in*

$$O\left(\frac{n}{k} + D \frac{\log f}{\log \log f}\right) \quad (1)$$

steps, where $f = \min\{n/k, n/D\}$.

Proof. Each round consists of $O(D + \max\{|S_i|\}) = O(D)$ steps. The assumption that $k \geq 2b$ implies that there are always at least $k/2$ active agents. Thus during phase 1, at least $k/2$ new trees are explored in each round other than the last one. Therefore the number of rounds in phase 1 is at most $q/(k/2) + 1 = O(1 + n/(kD))$.

Now we bound the number of rounds in phase 2 using numbers k_j, m_j and b_j introduced above. Consider any round j other than the last one ($m_{j+1} \geq 1$). In this round at least $m_{j+1} \lfloor k_j/m_j \rfloor$ additional holes are repaired, so we have

$$b_{j+1} \leq b_j - m_{j+1} \left\lfloor \frac{k_j}{m_j} \right\rfloor \leq b_j - \frac{m_{j+1} k_j}{m_j} \leq b_j - \frac{m_{j+1} k}{m_j} \frac{k}{4}. \quad (2)$$

The second inequality above holds because $k_j \geq m_j$ (we have $m_j \leq m_1 \leq b \leq k/2 \leq k_j$). The third inequality holds because $k_j \geq k/2$. Using (2) we get

$$\begin{aligned} 0 \leq b_{j+1} &\leq b_1 - \frac{k}{4} \left(\frac{m_2}{m_1} + \frac{m_3}{m_2} + \cdots + \frac{m_{j+1}}{m_j} \right) \\ &\leq b_1 - \frac{jk}{4} \left(\frac{m_{j+1}}{m_1} \right)^{1/j} \\ &\leq \frac{k}{2} - \frac{jk}{4} \left(\frac{m_{j+1}}{m_1} \right)^{1/j}. \end{aligned} \quad (3)$$

The third inequality above holds because the geometric mean $(m_{j+1}/m_1)^{1/j}$ of numbers m_{i+1}/m_i , $i = 1, 2, \dots, j$, is not greater than their arithmetic mean. Inequality (3) implies

$$\frac{m_1}{m_{j+1}} \geq \left(\frac{j}{2} \right)^j. \quad (4)$$

We have

$$m_1 \leq q \leq \frac{4n}{D}. \quad (5)$$

Let round $j+2$ be the last one. We have $m_{j+1} \geq m_{j+2} \geq 1$. It also must hold that $k_{j+1}/m_{j+1} \leq 2D$, or otherwise round $j+1$ would be the last one. Indeed, if $k_{j+1}/m_{j+1} > 2D$, then each tree S_i active at the beginning of round $j+1$ would be explored during this round with $\lfloor k_{j+1}/m_{j+1} \rfloor \geq 2D \geq |S_i| + D$ agents, so all holes in S_i and on the path from s to S_i would be repaired in this round. Therefore,

$$m_{j+1} \geq \max \left\{ 1, \frac{k_{j+1}}{2D} \right\} \geq \max \left\{ 1, \frac{k}{4D} \right\}. \quad (6)$$

Inequalities (5) and (6) imply

$$\frac{m_1}{m_{j+1}} \leq \frac{4n/D}{\max \{1, k/(4D)\}} \leq 16 \frac{n}{\max \{k, D\}} = 16 \min \left\{ \frac{n}{k}, \frac{n}{D} \right\} \leq 16f. \quad (7)$$

Inequalities (4) and (7) imply

$$\left(\frac{j}{2} \right)^j \leq 16f,$$

so

$$j = O \left(\frac{\log f}{\log \log f} \right).$$

Thus the total number of steps throughout the whole exploration algorithm is $O(D(n/(kD) + j))$, which is the bound (1). \square

4 Lower bound

For positive integers $n \geq d \geq 1$, we define $T(n, d)$ as the tree which is rooted at node s and has the following $\lfloor n/d \rfloor$ subtrees. Each subtree of the root is a d -node path with d leaf nodes attached to its end. Thus tree $T(n, d)$ has $1 + 2d\lfloor n/d \rfloor = \Theta(n)$ nodes and diameter $\Theta(d)$.

We show that for any (deterministic) exploration algorithm for this tree network, we (as the adversary) can place holes in such a way that the agents will be forced to go up and down the tree $\Omega(\log f / \log \log f)$ times. To decide where holes should be placed, we keep watching the movement of the agents and decide *not* to put any additional holes in the subtrees of $T(n, d)$ where currently relatively many agents are. These agents will have to go back up to the root of $T(n, d)$ and then down into other subtrees, where relatively few agents have gone before, to look for further holes.

Theorem 2. *For integers $n \geq d \geq 1$ and $n \geq k \geq 2$, and any algorithm exploring tree $T = T(n, d)$ with k agents starting from the root, there exists a placement of $b \leq k/2$ holes in T which forces the algorithm to run in*

$$\Omega\left(\frac{n}{k} + d \frac{\log f}{\log \log f}\right). \quad (8)$$

steps, where $f = \min\{n/k, n/d\}$.

Proof. Let \mathcal{A} be any algorithm exploring tree $T = T(n, d)$ with k agents, and let

$$\alpha = \max\{i - \text{positive integer: } i^i \leq f\}. \quad (9)$$

We have

$$\alpha = \Theta\left(\frac{\log f}{\log \log f}\right),$$

and we assume in our calculations that f is sufficiently large, so that $\alpha \geq 4$. Since for any exploration algorithm and for any network the number of steps must be $\Omega(n/k)$, it suffices to show that algorithm \mathcal{A} requires in the worst case $\Omega(d \log f / (\log \log f))$ steps.

We place the holes only at leaves of T . We simulate algorithm \mathcal{A} to decide which subtrees get how many holes. We look at intervals of d consecutive steps and call them *rounds* of the exploration. We will place the holes in such a way that the algorithm needs at least α rounds to complete the exploration.

We look first at round 1, and observe that the way the agents move during this round is independent of the distribution of holes in the leaves. This is because no agent can reach further than distance d from the root, so cannot reach any leaf of T . Let $n_0 = \lfloor n/d \rfloor$ and let $\mathcal{S}^{(0)}$ be the set of the n_0 subtrees of the root s in T . We sort these subtrees into the ascending sequence $S_1^{(0)}, \dots, S_{n_0}^{(0)}$, according to the number of agents in the subtrees at the end of the round. We take the $n_1 = \lfloor n_0/\alpha \rfloor$ lowest subtrees in this order to form the set $\mathcal{S}^{(1)}$ (where α is given in (9)). We decide that there are no holes in the subtrees in $\mathcal{S}^{(0)} \setminus \mathcal{S}^{(1)}$, that is,

all holes are in the subtrees in $\mathcal{S}^{(1)}$. Informally, if many agents have gone into a subtree, then we decide not to put any holes in this subtree (this would be a subtree in $\mathcal{S}^{(0)} \setminus \mathcal{S}^{(1)}$). Instead we will be putting holes in the subtrees where relatively few agents have gone (the subtrees in $\mathcal{S}^{(1)}$).

Now we observe what the algorithm does during round 2. Note that at the beginning of this round, there must be a subtree in $\mathcal{S}^{(0)} \setminus \mathcal{S}^{(1)}$ with at most $\lfloor k/(n_0 - n_1) \rfloor$ agents, so each subtree in $\mathcal{S}^{(1)}$ has at most $\lfloor k/(n_0 - n_1) \rfloor$ agents. Whenever an agent visits during this round a new leaf of a subtree in $\mathcal{S}^{(1)}$ (not visited before by any agent), we place a hole there. Only the agents which are in a subtree at the beginning of a round can explore a leaf of this subtree during this round. Thus at the end of round 2, in each subtree in $\mathcal{S}^{(1)}$, all but at most

$$\left\lfloor \frac{k}{n_0 - n_1} \right\rfloor$$

leaves are still unexplored. We sort the subtrees in $\mathcal{S}^{(1)}$ into the ascending sequence $S_1^{(1)}, \dots, S_{n_1}^{(1)}$, according to the number of agents in the subtrees at the end of the round, and we take the $n_2 = \lfloor n_1/\alpha \rfloor$ lowest subtrees in this order to form the set $\mathcal{S}^{(2)}$. We decide not to put more holes in the subtrees in $\mathcal{S}^{(1)} \setminus \mathcal{S}^{(2)}$, that is, all additional holes will be placed in the subtrees in $\mathcal{S}^{(2)}$.

We now generalise round 2 into a round $i \geq 2$. At the beginning of this round, we have (by induction) a set $\mathcal{S}^{(i-1)}$ of n_{i-1} subtrees of the root of T such that each of these subtrees has at most $\lfloor k/(n_{i-2} - n_{i-1}) \rfloor$ agents. We have already placed holes at some leaves of T in previous rounds. We will not put more holes at leaves of the subtrees in $\mathcal{S}^{(0)} \setminus \mathcal{S}^{(i-1)}$, but we may put additional holes in subtrees in $\mathcal{S}^{(i-1)}$. We observe how the agents explore tree T during this round, and whenever an agent visits a new leaf of a subtree in $\mathcal{S}^{(i-1)}$ (not visited before by any agent), we place a hole there. Only the agents which are in a subtree at the beginning of a round can explore a leaf of this subtree during this round. Thus at the end of round i , in each subtree in $\mathcal{S}^{(i-1)}$, all but at most

$$\left\lfloor \frac{k}{n_0 - n_1} \right\rfloor + \left\lfloor \frac{k}{n_1 - n_2} \right\rfloor + \dots + \left\lfloor \frac{k}{n_{i-2} - n_{i-1}} \right\rfloor \quad (10)$$

leaves are still unexplored, and the number of holes we have placed in the network so far is at most

$$n_1 \left\lfloor \frac{k}{n_0 - n_1} \right\rfloor + n_2 \left\lfloor \frac{k}{n_1 - n_2} \right\rfloor + \dots + n_{i-1} \left\lfloor \frac{k}{n_{i-2} - n_{i-1}} \right\rfloor \quad (11)$$

We sort the subtrees in $\mathcal{S}^{(i-1)}$ into the ascending sequence $S_1^{(i-1)}, \dots, S_{n_{i-1}}^{(i-1)}$, according to the number of agents in the subtrees at the end of round i , and take the $n_i = \lfloor n_{i-1}/\alpha \rfloor$ lowest subtrees in this order to form the set $\mathcal{S}^{(i)}$. We decide not to put more holes in the subtrees in $\mathcal{S}^{(i-1)} \setminus \mathcal{S}^{(i)}$, that is, all additional holes will be placed in the subtrees in $\mathcal{S}^{(i)}$. At the end of the round there must be a subtree in $\mathcal{S}^{(i-1)} \setminus \mathcal{S}^{(i)}$ with at most $\lfloor k/(n_{i-1} - n_i) \rfloor$ agents, so each subtree in $\mathcal{S}^{(i)}$ has at most $\lfloor k/(n_{i-1} - n_i) \rfloor$ agents.

Round i is not the last one, if the following three conditions hold.

1. $n_i \geq 1$ (that is, there is at least one subtree in the set $\mathcal{S}^{(i)}$).
2. The upper bound (10) on the number of explored nodes in a subtree in $\mathcal{S}^{(i)}$ at the end of round i is less than d , the number of leaves in one subtree.
3. The upper bound (11) on the number of holes placed in the network by the end of round i is less than $k/2$.

We show that the above three conditions hold for $i = \alpha/4$, assuming for convenience that $\alpha/4$ is integral. We have $n_0 = \lfloor n/d \rfloor \geq f \geq \alpha^\alpha$, and for $i = 0, 1, \dots, \alpha$,

$$n_i = \lfloor n_{i-1}/\alpha \rfloor, \quad (12)$$

so

$$n_i \geq \alpha^{\alpha-i}. \quad (13)$$

In particular, $n_{\alpha-1} \geq \alpha \geq 1$. Thus Condition 1 holds for $i = \alpha - 1$, so must also hold for $i = \alpha/4$.

Now we bound (10) for $i = \alpha - 1$:

$$\begin{aligned} & \left\lfloor \frac{k}{n_0 - n_1} \right\rfloor + \left\lfloor \frac{k}{n_1 - n_2} \right\rfloor + \dots + \left\lfloor \frac{k}{n_{\alpha-3} - n_{\alpha-2}} \right\rfloor \\ & \leq \frac{2k}{n_0} + \frac{2k}{n_1} + \dots + \frac{2k}{n_{\alpha-3}} \\ & \leq \frac{2k}{n_{\alpha-3}} \left(\frac{1}{\alpha^{\alpha-3}} + \frac{1}{\alpha^{\alpha-4}} + \dots + 1 \right) \\ & \leq \frac{4k}{n_{\alpha-3}}. \end{aligned} \quad (14)$$

The first inequality above holds because $n_i \leq n_{i-1}/\alpha \leq n_{i-1}/2$, ($\alpha \geq 4$). The second inequality holds because $n_{i+j} \leq n_i/\alpha^j$, so $n_i \geq n_{\alpha-3}\alpha^{\alpha-3-i}$. If $k \leq d$, then we continue (14) in the following way:

$$\frac{4k}{n_{\alpha-3}} \leq \frac{4d}{\alpha^3} < d,$$

where the first inequality follows from (13) and the second one from the assumption that $\alpha \geq 4$. If $k > d$, then $\alpha^\alpha \leq \lfloor f \rfloor = \lfloor n/k \rfloor$, so

$$n_0 = \left\lfloor \frac{n}{d} \right\rfloor \geq \left\lfloor \frac{k}{d} \right\rfloor \left\lfloor \frac{n}{k} \right\rfloor \geq \left\lfloor \frac{k}{d} \right\rfloor \alpha^\alpha,$$

and (13) becomes

$$n_i \geq \left\lfloor \frac{k}{d} \right\rfloor \alpha^{\alpha-i}. \quad (15)$$

Thus in this case we can continue (14) in the following way:

$$\frac{4k}{n_{\alpha-3}} \leq \frac{4k}{\lfloor k/d \rfloor \alpha^3} \leq \frac{8d}{\alpha^3} < d,$$

where the last inequality follows from $\alpha \geq 4$. Since in both cases the bound (10) is less than d for $i = \alpha - 1$, then Condition 2 holds for $i = \alpha - 1$, so it must also hold for $i = \alpha/4$.

Now we bound (11) for $i = \alpha/4$:

$$\begin{aligned} n_1 \left\lfloor \frac{k}{n_0 - n_1} \right\rfloor + n_2 \left\lfloor \frac{k}{n_1 - n_2} \right\rfloor + \cdots + n_{i-1} \left\lfloor \frac{k}{n_{i-2} - n_{i-1}} \right\rfloor \\ \leq \frac{k}{n_0/n_1 - 1} + \frac{k}{n_1/n_2 - 1} + \cdots + \frac{k}{n_{i-2}/n_{i-1} - 1} \\ \leq \frac{\alpha}{4} \cdot \frac{k}{\alpha - 1} < \frac{k}{2} \end{aligned}$$

The second inequality above follows from (12) and the fact that we consider $i = \alpha/4$. Thus also Condition 3 holds for $i = \alpha/4$.

We conclude that algorithm \mathcal{A} requires in the worst case $d\alpha/4 = \Omega(d \log f / (\log \log f))$ steps. \square

5 Conclusions

We have introduced a variation of the black hole search model by enabling the agents to repair the faulty nodes. We have shown matching worst-case upper and lower bounds on the running time of a k -agent exploration algorithm in this new model. These bounds imply that the trivial lower bound of $\Omega(n/k + D)$ is not always tight.

Our arguments assume that the number of holes b is at most $k/2$. This assumption can be weakened to $b \leq ck$ for an arbitrary constant $c < 1$ without affecting the asymptotic bounds given in Theorems 1 and 2. To repair all holes and to have at least one agent left at the end of the exploration, we only need to assume that $b \leq k - 1$. An adaptation of our upper and lower bound arguments to this general case would change the bounds by replacing n/k with $n/(k - b)$.

Our algorithm is *worst-case* asymptotically optimal. If we view it as an *approximation* algorithm, then its ratio bound is $O(\log f / \log \log f)$ and it can be shown that it is not constant. An interesting question is whether one can find a better approximation algorithm. The problem of designing a k -agent exploration with the minimum number of steps for the special case when there are no faults (or when each agent can repair any number of faults) is equivalent to the following k -TSP problem. Find k cycles which cover all nodes of a given undirected graph, minimising the length of the longest cycle. A $(5/2 - 1/k)$ -approximation algorithm for this problem is shown in [10]. It is not clear, however, whether this or other approximation algorithms for the k -TSP problem could be used effectively in our model, when there may be many faults and agents die after repairing one fault.

Most of the work on the synchronous black hole search problems, and this paper as well, assume that the topology of the whole network is known in advance and only the locations of faults are unknown. An interesting direction for further

research is to consider the case when the topology of the network is not known or only partially known.

References

1. Cooper, C., Klasing, R., Radzik, T.: Searching for black-hole faults in a network using multiple agents. In Shvartsman, A.A., ed.: Principles of Distributed Systems, 10th International Conference, OPODIS 2006, Proceedings. LNCS vol. 4305, Springer (2006) 320–332
2. Czyzowicz, J., Kowalski, D.R., Markou, E., Pelc, A.: Searching for a black hole in tree networks. In Higashino, T., ed.: Principles of Distributed Systems, 8th International Conference, OPODIS 2004, Proceedings. LNCS vol. 3544, Springer (2005) 67–80
3. Czyzowicz, J., Kowalski, D., Markou, E., Pelc, A.: Complexity of searching for a black hole. *Fundamenta Informaticae* **71**(2-3) (2006) 229–242
4. Dobrev, S., Flocchini, P., Kralovic, R., Ruzicka, P., Prencipe, G., Santoro, N.: Black hole search in common interconnection networks. *Networks* **47**(2) (2006) 61–71 (Preliminary version: “Black hole search by mobile agents in hypercubes and related networks” in Proceedings of the 6th International Conference on Principles of Distributed Systems, OPODIS 2002, 169-180).
5. Dobrev, S., Flocchini, P., Kralovic, R., Santoro, N.: Exploring an unknown graph to locate a black hole using tokens. In Navarro, G., Bertossi, L., Kohayakwa, Y., eds.: Fourth IFIP International Conference on Theoretical Computer Science, TCS 2006. IFIP International Federation for Information Processing vol. 209, Springer (2006) 131–150
6. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. *Algorithmica* **48**(1) (2007) 67–90 (Preliminary version in: Distributed Computing, 15th International Conference, DISC 2001, Proceedings, LNCS vol. 2180, Springer (2001) 166-179).
7. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. *Distributed Computing* **19**(1) (2006) 1–18 (Preliminary version in: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing, PODC 2002, ACM (2002) 153-161).
8. Dobrev, S., Kralovic, R., Santoro, N., Shi, W.: Black hole search in asynchronous rings using tokens. In Calamoneri, T., Finocchi, I., Italiano, G.F., eds.: Algorithms and Complexity, 6th Italian Conference, CIAC 2006. LNCS vol. 3998, Springer (2006) 139–150
9. Flocchini, P., Santoro, N.: Distributed security algorithms by mobile agents. In Chaudhuri, S., Das, S.R., Paul, H.S., Tirthapura, S., eds.: Distributed Computing and Networking, 8th International Conference, ICDCN 2006. LNCS vol. 4308, Springer (2006) 1–14
10. Frederickson, G.N., Hecht, M.S., Kim, C.E.: Approximation algorithms for some routing problems. *SIAM J. Comput.* **7**(2) (1978) 178–193
11. Hohl, F.: Time limited blackbox security: Protecting mobile agents from malicious hosts. In Vigna, G., ed.: Mobile Agent and Security. LNCS vol. 1419, Springer (1998) 92–113
12. Hohl, F.: A framework to protect mobile agents by using reference states. In: Proceedings of the 20th International Conference on Distributed Computing Systems, ICDCS 2000, IEEE Computer Society (2000) 410–417

13. Klasing, R., Markou, E., Radzik, T., Sarracco, F.: Approximation bounds for black hole search problems. In Anderson, J.H., Prencipe, G., Wattenhofer, R., eds.: Principles of Distributed Systems, 9th International Conference, OPODIS 2005, Proceedings. LNCS vol. 3974, Springer (2006)
14. Klasing, R., Markou, E., Radzik, T., Sarracco, F.: Hardness and approximation results for black hole search in arbitrary networks. *Theor. Comput. Sci.* **384**(2-3) (2007) 201–221 (Preliminary version in: Structural Information and Communication Complexity, 12th International Colloquium, SIROCCO 2005, Proceedings, LNCS vol. 3499, Springer (2005), 200-215).
15. Ng, S.K., Cheung, K.W.: Protecting mobile agents against malicious hosts by intention spreading. In Arabnia, H.R., ed.: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 1999, CSREA Press (1999) 725–729
16. Sander, T., Tschudin, C.: Protecting mobile agents against malicious hosts. In Vigna, G., ed.: Mobile Agent Security. LNCS vol. 1419, Springer (1998) 44–60