# Hardness and Approximation Results for Black Hole Search in Arbitrary Graphs*

Ralf Klasing**, Euripides Markou***, Tomasz Radzik†, and Fabiano Sarracco‡

**Abstract.** A black hole is a highly harmful stationary process residing in a node of a network and destroying all mobile agents visiting the node, without leaving any trace. We consider the task of locating a black hole in a (partially) synchronous arbitrary network, assuming an upper bound on the time of any edge traversal by an agent. For a given graph and a given starting node we are interested in finding the fastest possible Black Hole Search by two agents (the minimum number of agents capable to identify a black hole). We prove that this problem is NP-hard in arbitrary graphs, thus solving an open problem stated in [2]. We also give a 7/2-approximation algorithm, thus improving on the 4-approximation scheme observed in [2]. Our approach is to explore the given input graph via some spanning tree. Even if it represents a very natural technique, we prove that this approach cannot achieve an approximation ratio better than 3/2.

**Keywords:** approximation algorithm, black hole search, graph exploration, mobile agent, NP-hardness

## 1 Introduction

Problems related to security in a network environment have attracted many researchers. For instance protecting a host, i.e., a node of a network, from an agent's attack [11, 12] as well as protecting mobile agents from "host attacks", i.e., harmful items stored in nodes of the network, are important with respect to security of a network environment. Various methods of protecting mobile agents against malicious hosts have been discussed, e.g., in [8–13].

* Research supported in part by the European project IST FET CRESCCO (contract no. IST-2001-33135), the Royal Society Grant ESEP 16244, EGIDE, and the Ambassade de France en Grèce/Institut Français d' Athènes. Part of this work was done while E. Markou, T. Radzik and F. Sarracco were visiting the MASCOTTE project at INRIA Sophia Antipolis.

** MASCOTTE project, I3S-CNRS/INRIA/Université de Nice-Sophia Antipolis, 2004 Route des Lucioles, BP 93, F-06902 Sophia Antipolis Cedex (France), email Ralf.Klasing@sophia.inria.fr

*** Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, email emarkou@softlab.ece.ntua.gr

† Department of Computer Science, King's College London, London, UK, email radzik@dcs.kcl.ac.uk

‡ Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", email Fabiano.Sarracco@dis.uniroma1.it

We consider here malicious hosts of a particularly harmful nature, called *black holes* [1–6]. A black hole is a stationary process residing in a node of a network and destroying all mobile agents visiting the node, without leaving any trace. We are dealing with the issue of locating a black hole: assuming that there is at most one black hole in the network, at least one surviving agent must find the location of the black hole if it exists, or answer that there is no black hole, otherwise. The only way to locate the black hole is to visit it by at least one agent, hence, as observed in [4], at least two agents are necessary for one of them to locate the black hole and survive. Throughout the paper we assume that the number of agents is the minimum possible for our task, i.e., 2, and that they start from the same node, known to be safe.

The issue of efficient black hole search was extensively studied in [3–6] in many types of networks under the scenario of a totally asynchronous network (i.e., no upper bound on this time needed for an edge traversal). In this setting it was observed that, in order to solve the problem, the network must be 2-connected. Moreover, it is impossible to answer the question of whether a black hole actually exists in an asynchronous network, hence [3–6] work under the assumption that there is exactly one black hole and the task is to locate it.

In [1, 2] the problem is studied under the scenario we consider in this paper as well. The network is partially synchronous, i.e. there is an upper bound on the time needed by an agent for traversing any edge. This assumption makes a dramatic change to the problem: the black hole can be located by two agents in any graph; moreover the agents can decide if there is a black hole or not in the network. If, without loss of generality, we normalize to 1 the upper bound on edge traversal time, then we can define the cost of a Black Hole Search as the time taken under the worst-case location of the black hole (or when it does not exist in the network), assuming that all the edge traversals take time 1. In [2] the BHS problem is studied in tree topologies, while in [1] a variant of the problem is studied in which the black hole can be located only in a given set of nodes (which is a subset of the set of nodes of the graph) and it is proved that this variant is NP-hard in an arbitrary graph.

In this paper we show that the problem of finding the minimum cost Black Hole Search by two agents in an arbitrary graph is NP-hard even under the restricted scenario of one safe node (the starting node), thus solving an open problem stated in [2]. Moreover, we give a 7/2-approximation algorithm for this problem, i.e., we construct a polynomial time algorithm which, given a graph and a starting node as input, produces a Black Hole Search whose cost is at most 7/2 times larger than the best Black Hole Search for this input. This result improves on the 4-approximation scheme observed by [2]. Finally, we show that any Black Hole Search that explores the given input graph via some spanning tree cannot have an approximation ratio better than 3/2.

## 2 Model and Terminology

Let $G = (V, E)$ be a connected graph. We assume that the nodes of $G$ can be partitioned into two subsets: a set of BLACK HOLES $B \subsetneq V$, i.e. nodes destroying any agent visiting them without leaving any trace; and a set of SAFE nodes $V \setminus B$. During a Black Hole Search (or simply BHS), a set of agents starts from a special node $s \in V \setminus B$ (which we call STARTING NODE), and explores the graph $G$ by traversing its edges. Obviously $s$ is known to be a safe node; more generally, there exists a subset $\widehat{S} \subseteq V \setminus B$ of nodes initially known to be safe. The target of the agents is to report to $s$ the information on which nodes of $G$ are black holes.

In this paper we consider the following restricted version of the problem: $|B| \leq 1$ (i.e. there can be either one black hole or no black holes at all in $G$), $\widehat{S} = \{s\}$ (only the starting node is known to be safe), the set of agents has size 2, agents have a complete map of $G$, agents have distinct labels (we will call them *Agent*-1 and *Agent*-2) and they can communicate only when they are in the same node (and not, e.g., by leaving messages at nodes). Finally, the network is (at least partially) synchronous. We consider the following formalization, called the *Minimum Cost BHS Problem*, or simply BHS problem.

**Instance** : a graph $G = (V, E)$, and a node $s \in V$.
**Solution** : an EXPLORATION SCHEME $\mathcal{E} = (\mathbb{X}, \mathbb{Y})$ for $G$ and $s$, i.e. two equal-size sequences of nodes in $G$, $\mathbb{X} = \langle x_0, x_1, \ldots, x_T \rangle$ and $\mathbb{Y} = \langle y_0, y_1, \ldots, y_T \rangle$ which satisfy the feasibility constraints listed below.
**Measure** : the cost of the BHS based on $\mathcal{E}$.

When a BHS based of $\mathcal{E}$ is performed in $G$, *Agent*-1 follows the path defined by $\mathbb{X}$ while *Agent*-2 follows the path defined by $\mathbb{Y}$, in synchronized steps. In other words, at the end of the $i$-th step of the exploration scheme, *Agent*-1 is in node $x_i$, while *Agent*-2 is in node $y_i$. We say that each step has length one time unit. As soon as an agent deduces the existence and the exact location of the black hole, it "aborts" the exploration and returns to the starting node $s$ by traversing nodes in $V \setminus B$. A pair of sequences of nodes $\mathbb{X}$ and $\mathbb{Y}$ defines a feasible exploration scheme for a graph $G$ and a starting node $s$, which can be effectively used as a basis for a BHS on $G$, if it satisfies the following four constraints.

**Constraint 1:** $x_0 = y_0 = s$, $x_T = y_T$.
**Constraint 2:** for each $i = 0, \ldots, T-1$, either $x_{i+1} = x_i$, or $(x_i, x_{i+1}) \in E$; similarly for *Agent*-2, either $y_{i+1} = y_i$ or $(y_i, y_{i+1}) \in E$.
**Constraint 3:** $\bigcup_{i=0}^{T} \{x_i\} \cup \bigcup_{i=0}^{T} \{y_i\} = V$.

We need some further definitions to state the fourth constraint. Given an exploration scheme $\mathcal{E} = (\mathbb{X}, \mathbb{Y})$, the EXPLORED TERRITORY at step $i$ is

$$S_i = \begin{cases} \bigcup_{j=0}^{i} \{x_j\} \cup \bigcup_{j=0}^{i} \{y_j\}, & \text{if } x_i = y_i; \\ S_{i-1}, & \text{otherwise.} \end{cases}$$

Observe that, by Constraint 1, $S_0 = \{s\}$ and, by Constraint 3, $S_T = V$. A node $v$ is EXPLORED at step $i$ if $v \in S_i$, otherwise it is UNEXPLORED. The definition of explored territory covers the assumption that, whenever the two agents are in the same node, they communicate to each other that the nodes of the network they visited are safe. A MEETING STEP (or simply MEETING) is the step 0 and every step $1 \leq j \leq T$ such that $S_j \supsetneq S_{j-1}$. Observe that for each meeting step $j$, we must have $x_j = y_j$ (but not necessarily the opposite); we call this node a MEETING POINT. Each sequence of steps $\langle j+1, j+2, \ldots, k\rangle$ between two consecutive meetings $j$ and $k$ is a PHASE of length $k-j$. Now we can give the last constraint for a feasible exploration scheme. It says that during each phase, an agent can visit at most one unexplored node, and the same unexplored node cannot be visited by both agents (see [2]).

**Constraint 4:** for each phase $\langle j+1, \ldots, k\rangle$,
$|\{x_{j+1}, \ldots, x_k\} \setminus S_j| \leq 1$, $|\{y_{j+1}, \ldots, y_k\} \setminus S_j| \leq 1$, and
$\{x_{j+1}, \ldots, x_k\} \setminus S_j \neq \{y_{j+1}, \ldots, y_k\} \setminus S_j$.

**Lemma 1.** *If $k$ is a meeting step of exploration scheme $\mathcal{E}$, then $x_k = y_k \in S_{k-1}$. Each phase of $\mathcal{E}$ has length at least two.*[1]

Any length-2 phase $\langle j+1, j+2\rangle$ at the end of which the explored territory increases by 2 nodes must have the following structure. Let $m$ be the meeting point at step $j$. During step $j+1$, *Agent-i* visits an unexplored node $v_i$ adjacent to $m$. In step $j+2$, the agents meet in a safe node adjacent to both $v_1$ and $v_2$. Note that this node can be either $m$, and in this case we denote the phase as *b-split*$(m, v_1, v_2)$, or a distinct node $m'$, and in this case we denote the phase as *a-split*$(m, v_1, v_2, m')$.

For an exploration scheme $\mathcal{E} = (\mathbb{X}, \mathbb{Y})$ and a location of a black hole $B$, where $B = \emptyset$ or $B = \{b\} \in (V \setminus \{s\})$, the EXECUTION TIME is defined as follows. If $B = \emptyset$, then the execution time is equal to the length $T$ of the exploration scheme, plus the shortest path distance from $x_T (= y_T)$ to $s$. In this case the agents must perform the full exploration, and then get back to $s$. If $B = \{b\}$, then let $j$ be the first step in $\mathcal{E}$ such that $b \in S_j$. The execution time in this case is equal to $j$ plus the shortest path distance from $x_j (= y_j)$ to $s$ avoiding $b$. One agent, say *Agent*-1, vanishes into the black hole $b$ during the phase ending at step $j$, so it cannot meet *Agent*-2 at the expected meeting point $x_j = y_j$. Thus the surviving *Agent*-2 knows at the end of step $j$ the exact location of the black hole (see Constraint 4), so it can go straight back to $s$. The COST of the BHS based on an exploration scheme $\mathcal{E} = (\mathbb{X}, \mathbb{Y})$ is the maximum of the execution times of $\mathcal{E}$ for all possible locations of the black hole $B$.

It is easy to check that if $G$ is a tree, then the case $B = \emptyset$ gives always the maximum execution time among all possible locations of the black hole in the nodes of $G$. If $G$ is an arbitrary graph, then this property does not always hold.

[1] Some proofs are omitted due to the space restrictions. reasons.

## 3 NP-Hardness of the BHS problem in Arbitrary Graphs

In this section, we prove the NP-hardness of the BHS problem in arbitrary graphs by providing a reduction from a particular version of the Hamiltonian Circuit problem to the decision version of the BHS problem.

cpHC problem

**Instance** : cubic planar graph $G = (V, E)$, and an edge $(x, y) \in E$;
**Question** : does $G$ contain a Hamiltonian cycle that includes edge $(x, y)$?

dBHS problem

**Instance** : graph $G' = (V', E')$, with a starting node $s \in V'$, and a positive integer $X$;
**Question** : does there exist an exploration scheme $\mathcal{E}$ for $G'$ starting from $s$, such that the BHS based on $\mathcal{E}$ has cost at most $X$?

One can check that the reduction from the 3-SAT problem to the Hamiltonian Cycle problem given in [7] proves actually that the cpHC problem is NP-hard. For an arbitrary instance of the cpHC problem (i.e. for each planar cubic graph $G = (V, E)$ and edge $(x, y) \in E$), we construct in linear time a corresponding instance of the dBHS problem (i.e. a graph $G'$, a starting node $s$, and an integer $X$) such that the original instance is a positive instance of the cpHC problem if and only if the constructed instance is a positive instance of the dBHS problem.

Since $G$ is planar, we can find in linear time an (arbitrary) combinatorial planar embedding of $G$, i.e. a clockwise order $L_v$ of the neighbors of each node $v \in V$. We then construct both $G'$ and its embedding, as an extensions of $G$ and its (combinatorial planar) embedding, in the following five steps.

1. $G'$ has originally the same nodes (*original nodes*), the same edges and the same embedding as $G$.
2. Replace the edge $(x, y)$ with the edges $(x, s)$ and $(s, y)$, where $s \notin V$ is a new node. The node $s$ replaces the node $y$ in $L_x$, and the node $x$ in $L_y$.
3. For each edge $(v, w)$ in the current graph (the current set of edges is $E \cup \{(x, s), (s, y)\} \setminus \{(x, y)\}$) add two nodes $z_1^{(v,w)}$ and $z_2^{(v,w)}$ (twin nodes) and four edges $(z_1^{(v,w)}, v)$ , $(z_1^{(v,w)}, w)$, $(z_2^{(v,w)}, v)$ and $(z_2^{(v,w)}, w)$. For the embedding, place $z_1^{(v,w)}$ before and $z_2^{(v,w)}$ after $w$ in $L_v$. Similarly, place $z_1^{(v,w)}$ after and $z_2^{(v,w)}$ before $v$ in $L_w$.
4. For each node $v \in V \cup \{s\}$ and for each pair of nodes $z_i^{(v,w)}$, $z_j^{(v,u)}$ consecutive in $L_v$, add an edge between these $z_i^{(v,w)}$ and $z_j^{(v,u)}$. We call this edge a *shortcut edge*. Let $z_j^{(v,u)}$ follow $z_i^{(v,w)}$ in $L_v$. Then, in the embedding of $G'$, place $z_j^{(v,u)}$ immediately before $v$ in the order of the neighbors of $z_i^{(v,w)}$, and place $z_i^{(v,w)}$ immediately after $v$ in the order of the neighbors of $z_j^{(v,u)}$.
5. For each node $v \in V \cup \{s\} \setminus x$, add a new node $v^F$ (*flag node*) and an edge $(v, v^F)$. For the embedding, $v^F$ can be in any place in $L_v$.

Figure 1, which will be used as an illustration for the proof of Lemma 5, illustrates also the construction of graph $G'$. If $n = |V|$ and $e = |E|$, are the number of nodes and edges in $G$, then graph $G'$ has $n$ original nodes, one starting node $s$, $n$ flag nodes and $2(e+1)$ twin nodes. Since in cubic graphs $e = \frac{3}{2}n$, the total number of nodes in $G'$ is $5n+3$. This construction can be done in linear time with respect to the size of $G$. We assume that $s$ is the starting node, while the remaining $5n+2$ nodes are initially unexplored. We set $X = 5n+2$.

**Lemma 2.** *If $u$ and $w$ are two original nodes having a common neighbor $v$ in $G$, then there exists in $G'$ a path $[u, z', z'', w]$ where $z'$ is a twin node for the edge $(u, v)$ and $z''$ is a twin node for the edge $(v, w)$.*

**Lemma 3.** *Each twin node in $G'$ has degree 4.*

**Lemma 4.** *If the graph $G$ has a Hamiltonian cycle that includes edge $(x, y)$, then there exists an exploration scheme $\mathcal{E}_{HC}$ on $G'$ starting from $s$, such that the BHS based on it has cost at most $5n+2$.*

*Proof.* Let $\{v_1 = y, e_1, v_2, \ldots, e_{n-1}, v_n = x, e_n, v_1 = y\}$ be such Hamiltonian cycle in $G$. Consider the exploration scheme $\mathcal{E}_{HC}$ defined by the following sequence of phases:

1. **b-split**$(s, s^F, y)$, where $s^F$ is the flag node of $s$;
2. **a-split**$(s, z_1, z_2, y)$, where $z_1$ and $z_2$ are the twin nodes of the edge $(s, y)$;
3. for each node $v_i$ of the Hamiltonian cycle, with $(i = 1, \ldots, n-1)$:
   (a) let $v_j$ be the third neighbor of $v_i$, other than $v_{i-1}$ and $v_{i+1}$; if $j > i$ then **b-split**$(v_i, z_1, z_2)$, where $z_1$ and $z_2$ are the twin nodes of $(v_i, v_j)$;
   (b) **b-split**$(v_i, v_i^F, v_{i+1})$, where $v_i^F$ is the flag of $v_i$;
   (c) **a-split**$(v_i, z_1, z_2, v_{i+1})$, where $z_1$ and $z_2$ are the twin nodes of the edge $(v_i, v_{i+1})$;
4. **a-split**$(x, z_1, z_2, s)$, where $z_1$ and $z_2$ are the twin nodes of the edge $(x, s)$.

Now let us compute the length of $\mathcal{E}_{HC}$. As we have seen in Section 2, each *a-split* and *b-split* phase has length 2, and increases the explored territory by 2 nodes. The overall number of phases is therefore $(5n+2)/2$ and hence $\mathcal{E}_{HC}$ has length $5n+2$. Notice that this is also the exploration time of $\mathcal{E}_{HC}$, for the case $B = \emptyset$, since $\mathcal{E}_{HC}$ ends in $s$.

*Claim.* Consider the meeting step when the agents are to meet at a node $v_i$. If a black hole has been just discovered, then the remaining cost of the BHS is not greater than the remaining cost in the case of no black hole.

*Proof. (Sketch)* It suffices to show that if there is a black hole in $G'$, then the surviving agent can keep following the Hamiltonian Cycle (possibly by using a shortcut edge), and get to $s$ in less time units than in the case $B = \emptyset$.

This implies that also the cost of the BHS based on $\mathcal{E}_{HC}$ is $5n+2$, i.e. there is no allocation of the black hole that yields a larger exploration time. Observe that the BHS defined above is optimal since it is not possible to explore $5n+2$ nodes in less that $5n+2$ time units. □

**Lemma 5.** *If there exists an exploration scheme on $G'$ starting from $s$ such that the cost of the BHS based on it has cost at most $5n+2$, then the graph $G$ has a Hamiltonian cycle that includes edge $(x, y)$.*

*Proof.* Let $\mathcal{E}_\sigma$ be such exploration scheme. By Lemma 1, each phase of $\mathcal{E}_\sigma$ has length at least two and cannot explore more than two unexplored nodes. Since $G'$ has $5n+2$ unexplored nodes, $\mathcal{E}_\sigma$ must end in $s$, and each of its phases must be either an *a-split* or a *b-split*. Consider the sequence $M_\sigma$ of the meeting points established for $\mathcal{E}_\sigma$ at the end of each *a-split*, excluding the last one which is $s$.

*Claim.* Nodes $x$ and $y$ must be the two endpoints of $M_\sigma$ and $s$ cannot be in $M_\sigma$.

Each meeting point $v_i$ in $M_\sigma$ other than $s$ must have at least degree 5 since one neighbor is needed for the initial exploration of $v_i$, two unexplored neighbors are needed for the *a-split* that ends in $v_i$ and two further unexplored neighbors are needed for the *a-split* that leaves $v_i$. For this reason only the original nodes of $G'$ (neither flags nor twins) can be in $M_\sigma$. Finally, each flag node has to be explored with a *b-split* having as meeting point the original node adjacent to it, hence each original node of $G'$ (i.e. each node of $G$) must be in $M_\sigma$. Now we prove that the sequence $M_\sigma$ defines a Hamiltonian cycle on $G$, i.e.:

*a)* each node of $G$ appears exactly once in $M_\sigma$;

*b)* if nodes $v_i$ and $v_j$ are consecutive in $M_\sigma$, then the edge $(v_i, v_j)$ must be in $G$.

We start by proving *a)*. We have seen that each node of $G$ is in $M_\sigma$, thus we have to prove that no node appears twice or more. Compute the neighbors needed by a node $v_i$ in $M_\sigma$: at least one neighbor is needed for the initial exploration of $v_i$ (two neighbors, if it is done through an *a-split*). Then, for each occurrence of $v_i$ in $M_\sigma$, two unexplored neighbors are needed for meeting in $v_i$ with an *a-split*, and two additional unexplored neighbors are needed for leaving $v_i$ with an *a-split*. Moreover the flag $v_i^F$ has to be explored with a *b-split* from $v_i$, hence another unexplored neighbor of $v_i$ is needed. If the node $v_i$ occurs $k$ times in $M_\sigma$, then the total number of neighbors needed by $v_i$ is at least $1 + 4k + 2 = 3 + 4k$. Since each original node in $G'$ has only 10 neighbors (as $G$ is a cubic graph), it must be $k \leq 1$, thus each node appears exactly once in $M_\sigma$.

Now we prove property *b)* of $M_\sigma$. According to the structure of $G'$, *a-split* operations can either explore two twin nodes of an original edge (in this case property *b)* is verified since the meeting point is adjacent in $G$ to the previous one), or explore two original nodes of $G'$ and meet in another original node which may not be adjacent to the previous meeting point, thus violating property *b)*. Suppose that this latter kind of split (a *big a-split*) happens from a node $A$ to a node $B$; see Figure 1. In order to do this, $A$ must have two unexplored original neighbors ($C$ and $D$ in the figure) both having $B$ as a neighbor. $B$ must be already explored, therefore the last original neighbor of $B$ ($E$ in the figure) must have already been a meeting point (we can suppose without loss of generality that the one from $A$ to $B$ is the first *big a-split* in $M_\sigma$). At this point no other *big a-splits* can be performed from $B$ (all its original neighbors are now explored)

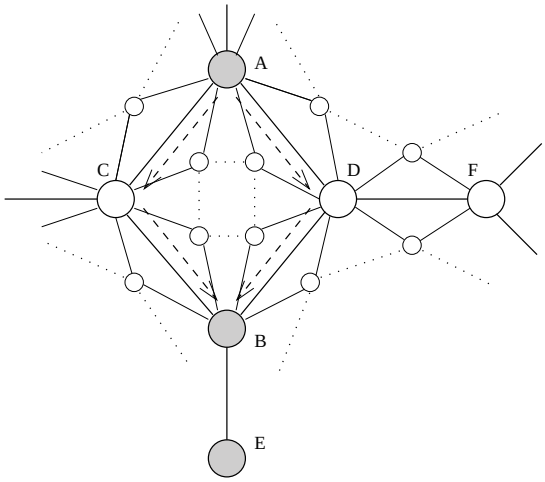


**Fig. 1.** A big *a-split* from $A$ to $B$. Big circles – original nodes in $G$; small circles – twin nodes; shortcut edges are dotted; flag nodes are not represented.

and, by property *a)*, $E$ cannot be again a meeting point, thus the sequence $M_\sigma$ can have either $C$ or $D$ as the next meeting point. Supposing that $C$ is that one, consider the instant when $D$ becomes a meeting point. We cannot get to $D$ with a *big a-split*, since $D$ does not have two neighbors in $G$ that are unexplored, hence also $F$ has been already a meeting point. Now all the original neighbors of $D$ have already been a meeting point in $M_\sigma$, and none of them can be $s$, thus there is no way to leave $D$ without violating property *a)*. Therefore there cannot be any *big a-split* in $\sigma$, and thus also property *b)* is verified. □

## 4 An Approximation Algorithm for the BHS problem

One may approach the BHS problem in an arbitrary graph $G$ in the following way. First select a spanning tree in $G$ and then search the graph using the tree edges. As observed in [2], this approach guarantees an approximation ratio of 4 since the following exploration of a $n$-node tree requires at most $4(n-1)$ steps. Both agents traverse the tree together in, say, the depth-first order and explore each new node $v$ with a two-step *probe phase*: one agent waits in the parent $p$ of $v$ while the other goes to $v$ and back to $p$.

To follow this "spanning-tree" approach effectively, we need good exploration schemes for trees and good spanning trees for those schemes. Intuitively a good heuristic for the former problem should be to minimize the time spent by one agent waiting for the other one. A good heuristic for the latter problem should be to minimize the number of nodes without siblings in the selected spanning tree. It may be difficult, if possible at all, to schedule exploration of such nodes not using probe phases (which imply waiting).

We assume throughout this section that the starting node $s$ in $G$ has degree at least 2. In Sections 4.1 and 4.2 we present algorithms *Search-Tree*($T$) and *Generate-Tree*($G$), which implement the above general heuristics. Algorithm *Search-Tree*($T$) generalizes the algorithm proposed in [2] for so-called *bushy trees*. The *Spanning Tree Exploration (STE)* algorithm returns for a given graph $G$ the exploration scheme computed by *Search-Tree*($T_G$), where $T_G$ is the spanning tree computed by *Generate-Tree*($G$). In Section 4.3 we show that the approximation ratio of the *STE* algorithm is at most 7/2.

### 4.1 Exploration Schemes for Trees

Let $T$ be a rooted $n$-node tree and let $s$ denote its root. Our algorithm *Search-Tree*($T$) for constructing an exploration scheme for $T$ uses the following order $L(T)$ of the nodes of $T$ other than the root. We first order the children of each node according to the number of descendants: a child with more descendants comes before a child with fewer descendants and the ties are resolved arbitrarily. Thus from now on $T$ is an *ordered* rooted tree. Let $\langle w_1, w_2, \ldots, w_p \rangle$ be the sequence of the internal nodes of $T$ ordered according to their depth-first-search numbers. The order $L(T)$ is this sequence with each node $w_i$ replaced by the (ordered) list of its children. The $i$-th node in the order $L(T)$ will be denoted by $v_i$ and called the $i$-th node of the tree. The odd (even) nodes of $T$ are the nodes at the odd (even) positions in $L(T)$.

We classify all nodes other than the root $s$ into the following three types. The *type-1* nodes are the leaves of $T$; the *type-3* nodes are the internal nodes with at least one sibling; and the *type-4* nodes are the internal nodes (other than the root) without siblings. Informally, in the exploration scheme which we produce for $T$ a *type-i* node contributes $i$ units to the total cost. Note that there is no type 2. We denote by $x_t$ the number of *type-t* nodes. We consider first the case when $T$ does not have any *type-4* nodes ($x_4 = 0$) and has an odd number of nodes (an even number of unexplored nodes). *Agent*-1 (*Agent*-2) will be responsible for exploring the odd (even) nodes in the order $L(T)$.

We construct first the exploration sequence $\mathbb{Y}_T$ for *Agent*-2. Initially $\mathbb{Y}_T = \langle w'_1 = s, w'_2, \ldots, w'_{2p-1} = s \rangle$ is the depth-first traversal of the $p$ internal nodes $w_1, w_2, \ldots, w_p$ of $T$. For each internal node $w$ in $T$, if $v_{2(i+1)}, v_{2(i+2)}, \ldots, v_{2(i+k)}$ are the children of $w$ which are at even positions in $L(T)$, then replace in $\mathbb{Y}_T$ the first occurrence of $w$ with the sequence $w, v_{2(i+1)}, w, v_{2(i+2)}, w, \ldots, v_{2(i+k)}, w$. That is, *Agent*-2 traverses the internal nodes of the tree in the depth-first manner, and whenever it arrives during this traversal at an internal node $w$ for the first time, before proceeding to the next node it first explores all children of $w$ which are even nodes of $T$. The exploration sequence $\mathbb{X}_T$ for *Agent*-1 is constructed analogously. Since $T$ has an odd number of nodes, both sequences $\mathbb{Y}_T$ and $\mathbb{X}_T$ have the same length $2p - 1 + (n - 1) = x_1 + 3x_3 + 1$. Lemma 6 says how these sequences are actually properly synchronized to form a valid (feasible) exploration scheme for $T$. It can be proven by induction, considering different kinds of relative positions of nodes $v_{2i-2}$, $v_{2i-1}$, $v_{2i}$ and their parents. Lemma 7 follows from Lemma 6 and the formula for the length of sequences $\mathbb{Y}_T$ and $\mathbb{X}_T$.

**Lemma 6.** *Let $T$ be a tree rooted at $s$ with $2q+1 \geq 3$ nodes and with no* type-4 *nodes. For the exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ and $i = 1, \ldots, q$,*
*1) there is the $i$-th meeting step $m(i)$ in $\mathcal{E}_T$, so there is phase $i$ in $\mathcal{E}_T$;*
*2) the set of the explored nodes st step $m(i)$ is $S_{m(i)} = \{s\} \cup \{v_1, \ldots, v_{2i}\}$;*
*3) the meeting point at the end of phase $i$ is the parent of node $v_{2i}$.*

**Lemma 7.** *Let $T$ be a tree rooted at $s$ which has $n = 2q+1 \geq 3$ nodes and does not have any* type-4 *nodes. The exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ is valid, can be computed in linear time and its cost is equal to $x_1 + 3x_3$.*

Now we consider a general tree $T$, which may have *type-4* nodes. For each *type-4* node $v$ in $T$, we add a new leaf $l$ with parent $v$, placing $l$ at the end of the list of the children of $v$. If the total number of nodes, including the added nodes, is even, then we add one more leaf as the last child of an arbitrary internal node. The obtained ordered tree $T'$ is as required in Lemma 7. We obtain $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ for $T$ from $\mathcal{E}_{T'} = (\mathbb{X}_{T'}, \mathbb{Y}_{T'})$ for $T'$ by replacing the traversals of the added edges with waiting: if an added leaf $l$ is, say, an odd node in $T'$ and has parent $v$, then replace $l$ in $\mathbb{X}_{T'}$ with $v$. Tree $T'$ has $x'_1 = x_1 + x_4 + \beta$ leaves, for $\beta \in \{0, 1\}$, and $x'_3 = x_3 + x_4$ *type-3* nodes. Thus, using Lemma 7, the cost of $\mathcal{E}_T$ is as given in the following lemma.

**Lemma 8.** *The exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ for a rooted tree $T$ is valid, can be computed in linear time and its cost is at most $x_1 + 3x_3 + 4x_4 + 1$.*

### 4.2 Generating a Good Spanning Tree of a Graph

We describe now our algorithm *Generate-Tree*($G$) which computes a spanning tree $T_G$ of a graph $G$ such that the cost of the exploration scheme for $G$ computed by algorithm *Search-Tree*($T_G$) has cost at most 7/2 times worse than the minimum cost of an exploration scheme for $G$. Algorithm *Generate-Tree* tries to minimize the number of *type-4* nodes. Following the terminology used in [2], we define as BUSHY the rooted trees in which each internal node has at least two children. Bushy trees do not have *type-4* nodes. Algorithm *Search-Tree* computes for a bushy tree the same exploration scheme as the algorithm proposed in [2], and it was proven in [2] that that exploration scheme is optimal.

Algorithm *Generate-Tree* uses procedure *Bushy-Tree*($G', v$) which for a given graph $G' = (V', E')$ and a node $v \in V'$ computes a maximal bushy tree $T$ in $G'$ rooted at $v$ (that is, there is no bushy tree $T'$ rooted at $v$ such that $T \subsetneq T' \subseteq E'$).

Algorithm *Generate-Tree* consists of three steps. In Step 1 we compute vertex disjoint trees $T_0, T_1, \ldots, T_k$. using procedure *Bushy-Tree*. Tree $T_0$ is rooted at $s$. Tree $T_i$, $i = 1, 2 \ldots, k$ is returned by *Bushy-Tree*($G', v$), where $G'$ is the subgraph of $G$ induced by the set of nodes not covered by the previous trees $T_0, \ldots, T_{i-1}$ and $v$ is an arbitrary node in $G'$ with degree (in $G'$) at least 3. At the end of Step 1, the graph $G'$ induced by the remaining uncovered nodes does not have a node of degree greater than 2. In Step 2 the nodes of $G'$ (EXTERIOR NODES) are appended to the trees $T_0, T_1, \ldots, T_k$ by creating shortest paths to the leaves

of these trees. In the final Step 3 all trees $T_0, T_1, \ldots, T_k$ are linked together into a spanning tree $T_G$ of $G$. Figure 2 shows an example of a final spanning tree, including the details about the types of the nodes used later in the analysis. The algorithm can be implemented to run in linear time.

**Algorithm 1** Algorithm Generate-Tree $(s, G)$

*Step 1 (collecting bushy trees):*
$T_0 \leftarrow$Bushy-Tree $(s,G)$; $F \leftarrow \emptyset$;
Let $G'$ be the subgraph of $G$ induced by the nodes of $G$ not in $T_0$;
**while** there exists a node $u$ in $G' : d_{G'}(u) \geq 3$ **do**
  $T \leftarrow$Bushy-Tree $(u,G')$; $F \leftarrow F \cup \{T\}$;
  Remove from $G'$ the nodes in $T$ and all edges incident to them;
**end while**
Let $X$ be the set of nodes still in $G'$ (*exterior nodes*);
*Step 2 (appending exterior nodes):*
Let $X_e$ be the set of nodes in $X$ adjacent to trees in $\{T_0\} \cup F$ (*type-e* nodes);
Append each node in $X_e$ to one of the trees $T_i$ adjacent to it;
Let $X_m = X - X_e$ (*type-m* nodes);
Append nodes in $X_m$ by creating shortest paths to *type-e* nodes;
*Step 3 (linking the trees):*
$T_G \leftarrow T_0$;
**while** $F \neq \emptyset$ **do**
  Find an edge $(u, v)$ such that $u \in T_G$ and $v \in T \in F$;
  Add $(u, v)$ and $T$ to $T_G$; $F \leftarrow F - \{T\}$;
**end while**

The set $X$ of exterior nodes can then be partitioned into the nodes adjacent to a leaf of some $T_i$ (*type-e* nodes) and the nodes not adjacent to any node of $T_i$ (*type-m* nodes). Note that we use two classifications of the nodes of $G$. The first one partitions the nodes into three *types 1, 3* and *4*, and the second one which assigns *types e* and *m* to some of the nodes. We will need to introduce further types and sub-types to the second classification, We will need also to refer to intersections of types from these two classifications; for example, a *type-4e* node is a node which is both a *type-4* and a *type-e* node.

**Lemma 9.** *Each* type-m *node has degree at most 2 in G. For each maximal path* $\langle v_0, v_1, \ldots, v_h \rangle$ *of* type-m *nodes in G, one of the two end nodes is adjacent to a* type-e *node, while the other has degree 1 or is adjacent to a* type-e *node.*

On the basis of the above lemma, the computation done during Step 2 can be viewed in the following way. Append first each *type-e* node to a leaf of a tree $T_i$, and then consider the maximal paths of *type-m* nodes. Append the length-0 paths and the paths with one end node having degree 1 (in $G$) to the adjacent *type-e* nodes. For each remaining path, remove its middle edge, breaking the possible tie arbitrarily, and append the resulting two paths to the adjacent *type-e* nodes (after the removal of the middle edges, each path is adjacent to exactly

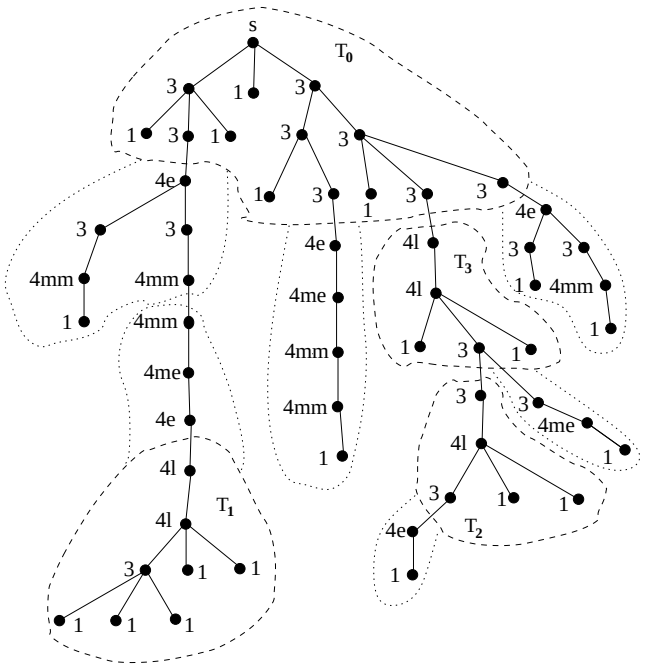


**Fig. 2.** An example of a final spanning tree computed by algorithm *Generate-Tree*. Types of nodes 1, 3 or 4, and further subdivision of *type-4* nodes are indicated.

one *type-e* node). We sub-divide the *type-m* nodes into the *type-me* nodes which are adjacent to *type-e* nodes and the remaining *type-mm* nodes. During Step 2 paths composed of one *type-e* node, possibly one *type-me* node and possibly one or more *type-mm* nodes are appended to leaves of the trees $T_i$. During merging of tree in Step 3, at most 2 nodes of each appended tree may become *type-4* nodes. We call them *type-4l* nodes.

### 4.3 Approximation Ratio of the *STE* Algorithm

Lemma 8 implies that the cost of the exploration scheme computed by the *STE* algorithm for graph $G$ is

$$t_{ALG} \leq x_1 + 3x_3 + 4x_4 + 1, \tag{1}$$

where $x_i$ is the number of *type-i* nodes in tree $T_G$. Since the cost of the optimal exploration scheme is at least $t_{OPT} \geq n-1 = x_1+x_3+x_4$, the number $x_4$ should be closely analysed. We provide first a bound on $x_{4e} + x_{4me} + x_{4l}$ (Lemma 10). We then use the number $x_{4mm}$ to strengthen the lower bound on the cost of the optimal exploration scheme: no exploration scheme can keep exploring the *type-4mm* nodes at the average rate of one node per one step (Lemma 11). Lemmas 10 and 11 and the bound (1) imply our final result stated in Theorem 1.

**Lemma 10.** *For any tree $T_G$ generated by algorithm Generate-Tree,*

$$x_{4e} + x_{4me} + x_{4l} \leq 5x_1 + x_3 + x_{1mm} - 10. \tag{2}$$

*Proof.* Consider the maximal paths of *type-4* nodes in $T_G$, distinguishing two kinds of such paths. In an *leaf path* the last node (the node furthest from the root) has only one child (which must be a leaf); while in a *mid-tree path* the last node has at least two children. Each leaf path contains at most one *type-4e* node, at most one *type-4me* node and possibly one or more *type-4mm* nodes. Moreover, if a *type-4me* node is present in such a path, then the leaf attached to the last node of the path is a *type-1mm* node. Each mid-tree path contains at most two *type-4e* nodes, at most two *type-4me* nodes, at most two *type-4l* nodes and any number of *type-mm* nodes. Hence, denoting by $z'$ and $z''$ the number of the leaf paths and the number of the mid-tree paths, respectively, we have $x_{4e} \leq z' + 2z''$, $x_{4em} \leq x_{1mm} + 2z''$, and $x_{4l} \leq 2z''$, so

$$x_{4e} + x_{4me} + x_{4l} \leq 6z'' + z' + x_{1mm}. \tag{3}$$

The last node of a mid-tree path must be a branching node in $T_G$, so $z'' \leq x_1 - 2$. On the other hand, since different maximal path of *type-4* nodes are attached in $T_G$ to a different *type-3* nodes, we have $z'' \leq x_3 - z'$. Thus

$$6z'' \leq 5(x_1 - 2) + x_3 - z', \tag{4}$$

and Inequalities (3) and (4) give immediately (2). □

**Lemma 11.** *The minimum cost of an exploration scheme for graph $G$ is*

$$t_{OPT} \geq n + \frac{1}{2}x_{mm} = x_1 + x_3 + x_4 + \frac{1}{2}(x_{1mm} + x_{4mm}). \tag{5}$$

**Theorem 1.** *For any graph $G$, the ratio of the cost $t_{ALG}$ of the exploration scheme computed for $G$ by the STE algorithm to the cost $t_{OPT}$ of an optimal exploration scheme for $G$ is at most $7/2$.*

## 5 Limitations of BHS Based on Spanning Trees

The approximation algorithm for the BHS problem in arbitrary graphs which we presented in the previous section was based on the following two-part approach. Find first a suitable spanning tree $T$ of the graph to explore, and then explore $T$ using a good BHS for trees. We show now that no graph exploration using this technique can guarantee a better approximation ratio than $3/2$.

Let $G_c = (V, E)$ be an odd-length cycle with $V = \{v_1, v_2, \ldots, v_c\}$ and $E = \{(v_1, v_2), \ldots, (v_{c-1}, v_c), (v_c, v_1)\}$. A new graph $G'_c$ is obtained from $G_c$ using the construction for the NP-hardness proof given in Section 3, taking edge $(v_c, v_1)$ as $(x, y)$, with the following modification. The construction from Section 3 would add two shortcut edges for each node $v \in V \cup \{s\}$, but we add only one. If we trace the cycle $\langle s, v_1, v_2, \ldots, v_c\rangle$ in a planar embedding of $G'_c$, then the shortcut

edges alternate between both sides of the cycle. Graph $G'_7$ is shown in Figure 3. Graph $G'_c$ has $4c+3$ nodes and, using an argument as in the proof of Lemma 4, one can show that the cost of an optimal exploration scheme for $G'_c$ is $4c+2$.

Consider the spanning tree of $G'_c$ as shown in Figure 3. In the notation from Section 4.1, this tree has $x_3 = c-1$ *type-3* nodes $(v_1, v_2, \ldots, v_{c-1})$ and $x_1 = 3c+3$ *type-1* nodes. Lemma 7 implies that the cost of the exploration scheme computed for this tree by algorithm *Search-Tree* from Section 4.1 is $x_1 + 3x_3 = 6c$. We show now that this is essentially the best what an exploration scheme for a spanning tree of $G'_c$ can do. Lemma 12, proven in [2], and Lemma 13 imply that the cost of any exploration scheme for any spanning tree of $G'_c$ is at least $6c-2$, so at least $3/2 - O(1/c)$ times higher than the optimal cost of exploring $G'_c$.

**Lemma 12.** *[2] Let $T = (V_T, E_T, s)$ be a rooted tree with $n+1$ nodes. Let $x_\beta$ and $x_\gamma$ denote the number of nodes in $V_T \setminus \{s\}$ with exactly one descendant (*type-$\beta$ *nodes) and with at least two descendants (*type-$\gamma$ *nodes), respectively. Then the cost of any exploration scheme for $T$ is at least $n + x_\beta + 2x_\gamma$.*

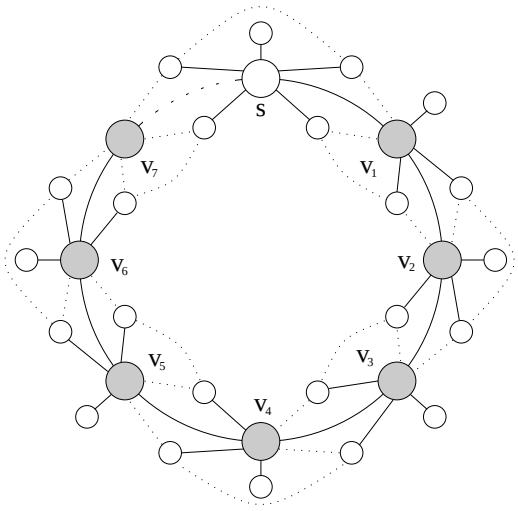


**Fig. 3.** Graph $G'_7$ and its "good" spanning tree (solid edges).

**Lemma 13.** *For any spanning tree $T$ of $G'_c$ rooted at $s$, $x_\beta + 2x_\gamma \geq 2c-4$.*

*Proof.* Each node in $V \setminus \{v_c\}$ has at least one descendant. Let $z$ be the number of *type-$\beta$* nodes in $V \setminus \{v_c\}$. At least $z-2$ shortcut edges must belong to $T$: for each *type-$\beta$* node $u$ in $V \setminus \{v_c\}$ except for at most two such nodes, the shortcut edge of $u$ must be in $T$. Thus there are at least $z-2$ twin nodes of type $\beta$ or $\gamma$, so $x_\beta + 2x_\gamma \geq z + 2(c-1-z) + z - 2 = 2c-4$.

## 6 Conclusion

We proved that producing an optimal exploration scheme for an arbitrary graph is NP-hard, thus solving an open problem stated in [2]. We also gave a polynomial time 7/2-approximation algorithm for the BHS problem, which improves the ratio of 4 observed in [2]. Finally, we showed that any BHS that explores a graph via some spanning tree, as our algorithm does, cannot have an approximation ratio better than 3/2. A natural open problem is to decrease the 7/2 approximation ratio. It would also be interesting to generalize the model used in Section 2, and to investigate complexity issues, for the case of $k \geq 3$ agents.

## References


1. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. 2004. manuscript.
2. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in tree networks. In *Proc. of 8th International Conference on Principles of Distributed Systems (OPODIS 2004)*, pages 34–35, 2004.
3. S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, and N. Santoro. Black hole search by mobile agents in hypercubes and related networks. In *Proc. of 6th International Conference on Principles of Distributed Systems (OPODIS 2002)*, pages 169–180, 2002.
4. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile agents searching for a black hole in an anonymous ring. In *Proc. of 15th International Symposium on Distributed Computing (DISC 2001)*, 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. In *Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 153–161, 2002.
6. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Multiple agents rendezvous on a ring in spite of a black hole. In *Proc. 7th International Conference on Principles of Distributed Systems (OPODIS 2003)*, 2003.
7. M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar hamiltonian circuit problem is np-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.
8. F. Hohl. Time limited black box security: Protecting mobile agents from malicious hosts. In *Proc. Conf. on Mobile Agent Security*, LNCS 1419, pages 92–113, 1998.
9. F. Hohl. A framework to protect mobile agents by using reference states. In *Proc. 20th Int. Conf. on Distributed Computing Systems (ICDCS 2000)*, pages 410–417, 2000.
10. S. Ng and K. Cheung. Protecting mobile agents against malicious hosts by intention of spreading. In *Proc. Int. Conf. on Parallel and Distributed Processing and Applications (PDPTA'99)*, pages 725–729, 1999.
11. T. Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. In *Proc. Conf. on Mobile Agent Security*, LNCS 1419, pages 44–60, 1998.
12. K. Schelderup and J. Ines. Mobile agent security – issues and directions. In *Proc. 6th Int. Conf. on Intelligence and Services in Networks*, LNCS 1597, pages 155–167, 1999.
13. J. Vitek and G. Castagna. Mobile computations and hostile hosts. In D. Tsichritzis, editor, *Mobile Objects*, pages 241–261. University of Geneva, 1999.