Hardness and Approximation Results for Black Hole Search in Arbitrary Networks[⋆]

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. The Black Hole Search is the task of locating all black holes in a network by exploring it with mobile agents. We consider the problem of designing the fastest Black Hole Search, given the map of the network and the starting node. We study the version of this problem that assumes that there is at most one black hole in the network and there are two agents, which move in synchronized steps. We prove that this problem is NP-hard in arbitrary graphs (even in planar graphs), solving an open problem stated in [1]. We also give a $3\frac{3}{8}$-approximation algorithm, showing the first non-trivial approximation ratio upper bound for this problem. Our algorithm follows a natural approach of exploring networks via spanning trees. We prove that this approach cannot lead to an approximation ratio bound better than 3/2.

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

1 Introduction

1.1 The Background and the Problem

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 [15, 16] as well as protecting mobile agents from "host attacks",

[⋆] Research supported in part by the European project IST FET CRESCCO (contract no. IST-2001-33135), the Royal Society Grant ESEP 16244, EGIDE, the Ambassade de France en Grèce/Institut Français d'Athènes, and EPEAEK 70/3/6865. 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 `Tomasz.Radzik@kcl.ac.uk`

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

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 [9, 10, 14–17].

We consider here malicious hosts of a particularly harmful nature, called *black holes* [2, 1, 3, 5, 6, 4]. A black hole is a node in a network which contains a stationary process destroying all mobile agents visiting this node, without leaving any trace. Since agents cannot prevent being annihilated once they visit a black hole, the only way of protection against such processes is first identifying the hostile nodes and then avoiding them. To identify a black hole, it must be to visited at least once. An agent which falls into a black hole is destroyed and will not turn up at a node where the other agents may expect it. This way the surviving agents infer the existence and location of a black hole. We assume in this paper that there may be at most one black hole in the network, there are exactly two agents, they start from the same given starting node s, which is known to be safe, and at least one agent must report back to s with information where exactly the black hole is or that there is none. We consider the problem of designing a black hole search scheme for a given network and a given starting node.

The issue of efficient black hole search was extensively studied in [3, 5, 6, 4] in many types of networks under the scenario of a totally asynchronous network, i.e., while every edge traversal by a mobile agent requires finite time, there is no upper bound on this time. In this setting it was observed that in order to solve the problem the network must be 2-connected. Moreover, in an asynchronous network it is impossible to answer the question of whether a black hole actually exists, hence it is assumed in [3, 5, 6, 4] that there is exactly one black hole and the task is to locate it.

In [2, 1] the problem is studied under the scenario we consider in this paper. The network is synchronous, i.e., there is an upper bound on the time needed by an agent for traversing any edge. The synchronous network 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. To measure the efficiency of a black hole search, it is assumed that each agent takes exactly one time unit (one synchronized step) to traverse one edge (and to make all necessary computations associated with this move). Then the cost of a given black hole search scheme in a given network G and from a given starting node s is defined as the total time the search takes under the worst-case location of the black hole (or when there is no black hole in the network).

The cost of a black hole search should be distinguished from the time complexity of an algorithm producing the scheme for the search. Informally, the former is the time of walking, while the latter is the time of preparing (planning) the walk. Following [2] and [1], we study the optimization problem of computing (preparing), for a given network G and the starting node s, a minimum-cost black hole search scheme. From now on, the Black Hole Search problem refers to this optimization problem.

In [1] the Black Hole Search problem is studied in tree topologies, and the main results given are an exact polynomial-time algorithm for some sub-class of trees and a 5/3-approximation algorithm for arbitrary trees. The existence of an exact polynomial-time algorithm for arbitrary trees is left open. In [2] the following variant of the problem is studied. The input instance is a triple $(G, s, \widehat{S})$, where G and s are, as above, a network and the starting node, and $\widehat{S} \supseteq \{s\}$ is a given subset of nodes known to be safe (no black hole can be located in any node in $\widehat{S}$). The main results presented in [2] are that for arbitrary graphs this variant of the Black Hole Search problem is NP-hard but can be approximated within a ratio bound 9.3. Observe that the problem we consider in this paper is the problem considered in [2] restricted to the case when $\widehat{S} = \{s\}$.

1.2 Our Results

We show that the problem of finding a minimum cost Black Hole Search in an arbitrary graph when only the starting node is initially known to be safe is NP-hard, thus solving an open problem stated in [1]. Moreover, we give a $3\frac{3}{8}$-approximation algorithm for this problem, i.e., we construct a polynomial time algorithm which for a graph and a starting node as input, produces a Black Hole Search whose cost is at most $3\frac{3}{8}$ times the best cost of a Black Hole Search for this input. This result improves on the 4-approximation scheme observed in [1], and it is the first non-trivial approximation ratio bound for this problem. Our approximation algorithm explores the input graph via some spanning tree. We show a limitation of this natural approach by presenting an infinite family of graphs such that the cost of any Black Hole Search which explores these graphs via spanning trees is at least $3/2 - O(1/n)$ times the optimal cost.

1.3 Structure of the Paper

Section 2 presents the model of the problem we study, and provides the terminology we will use in the rest of the paper; moreover some fundamental properties are stated. In Section 3 we prove that the minimum cost Black Hole Search problem in arbitrary graphs is NP-hard. In Section 4 we give a $3\frac{3}{8}$ approximation scheme for this problem. Finally, Section 5 is intended to investigate the limitations of the spanning tree based approach we use in this paper.

2 Model and Terminology

We represent a network as a connected undirected graph $G = (V, E)$, without multiple edges or self-loops, where nodes denote hosts and edges denote communication links. In the following we will use the terms graph and network, host and node, and link and edge interchangeably, although we tend to use the term graph to mean an abstract representation of a network. We assume that the nodes of G can be partitioned into two subsets:

- a set of BLACK HOLES $B \subsetneq V$, i.e., of nodes destroying any agent visiting them without leaving any trace;
- a set of SAFE NODES $V \setminus B$.

During a Black Hole Search (or simply BHS), agents start from a special node $s \in V \setminus B$ called the STARTING NODE, and explore graph G by traversing its edges. The starting node s is known to be a safe node; and generally a subset of nodes $\widehat{S}$ with $s \in \widehat{S} \subseteq V \setminus B$, which are known to be safe, may be given. The target of the agents is to report to s 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), there are two agents, agents have a complete map of G, agents have distinct labels (we will call them *Agent*-1 and *Agent*-2) and communicate only when they are in the same node (and not, e.g., by leaving messages at nodes). Finally, the network is synchronous. This means that there exists an upper bound on the time needed by any edge traversal; we normalize this bound, and assume that each traversal requires one time unit. We now formalize the problem we study in this paper, calling it the MINIMUM COST BHS PROBLEM, or simply the BHS problem.

BHS problem

Instance : a connected undirected graph $G = (V, E)$ and a node $s \in V$.
Solution : an EXPLORATION SCHEME $\mathcal{E}_{G,s} = (\mathbb{X}, \mathbb{Y})$ for G and s, where $\mathbb{X} = \langle x_0, x_1, \ldots, x_T \rangle$ and $\mathbb{Y} = \langle y_0, y_1, \ldots, y_T \rangle$ are two equal-length sequences of nodes in G, which satisfies the feasibility constraints 1–4 given below. The length of the exploration scheme $\mathcal{E}_{G,s}$ is defined to be T.
Measure : the cost of the BHS based on $\mathcal{E}_{G,s}$.

When the BHS based on a given exploration scheme $\mathcal{E}_{G,s}$ is performed in G, *Agent*-1 follows the path defined by $\mathbb{X}$ while *Agent*-2 follows the path defined by $\mathbb{Y}$. In other words, at the end of the i-th step of the exploration scheme (at time i), *Agent*-1 is in node x_i, while *Agent*-2 is in node y_i. 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 $V \setminus B$. The cost of the BHS based on a given exploration scheme $\mathcal{E}_{G,s}$ is defined later in this section.

Our definition of an exploration scheme might give the impression that we consider only "oblivious" exploration. However, since there are only two agents, at most one black hole, the whole graph is known in advance, and exploration is deterministic, there are no "more adaptive" explorations. Intuitively, for any exploration algorithm, if there is no black hole, then one agent follows some sequence of moves $\mathbb{A}$, while the other follows a sequence $\mathbb{B}$, and these sequences can be calculated before the exploration starts. If there is a black hole, then anyway the agents must follow sequences $\mathbb{A}$ and $\mathbb{B}$, until one agent realises that the other one has died.

If $\mathbb{X} = \langle x_0, x_1, \ldots, x_T\rangle$ and $\mathbb{Y} = \langle y_0, y_1, \ldots, y_T\rangle$ are two equal-length sequences of nodes in G, then $\mathcal{E}_{G,s} = (\mathbb{X}, \mathbb{Y})$ is a feasible exploration scheme for G and the starting node s (and can be effectively used as a basis for a BHS in G) if the constraints 1–4 stated below are satisfied.

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 $(x_i, x_{i+1}) \in E$; and similarly either $y_{i+1} = y_i$ or $(y_i, y_{i+1}) \in E$.

Constraint 3: $\bigcup_{i=0}^{T} \{x_i\} \cup \bigcup_{i=0}^{T} \{y_i\} = V$.

Constraint 1 corresponds to the fact that both agents start from the given starting node s. The requirement that the sequences $\mathbb{X}$ and $\mathbb{Y}$ end at the same node provides a convenient simplification of the reasoning without loss of generality. Constraint 2 models the fact that during each step, each agent can either WAIT in the node v where it was at the end of the previous step, or traverse an edge of the network to move to a node adjacent to v. Constraint 3 assures that each node in V is visited by at least one agent during the exploration. We need additional definitions to state Constraint 4.

Given an exploration scheme $\mathcal{E}_{G,s} = (\mathbb{X}, \mathbb{Y})$, for each $i = 0, 1, \ldots, T$, we call the EXPLORED TERRITORY at step i the set S_i defined in the following way:

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

Thus $S_0 = \{s\}$ by Constraint 1, $S_T = V$ by Constraint 1 and Constraint 3, and $S_{j-1} \subseteq S_j$ for each step $1 \le j \le T$. A node v is EXPLORED at a step i if $v \in S_i$, or UNEXPLORED otherwise. These definitions reflect the assumption that the agents communicate with each other, exchanging their full knowledge, when and only when they meet at a node. An unexplored node v may have been already visited by one of the agents, but it will become explored only when the agents meet (and communicate) next time. If both agents are alive at the end of step i, then the explored nodes at this step are all nodes which are known to *both* agents to be safe. Note that the explored territory is defined for an exploration scheme $\mathcal{E}_{G,s}$, not for the BHS based on $\mathcal{E}_{G,s}$, so it does not take into account the possible existence of the black hole. This is taken into account in the definition of the cost of the BHS based on $\mathcal{E}_{G,s}$.

A MEETING STEP (or simply MEETING) is the step 0 and every step $1 \le j \le T$ such that $S_j \ne S_{j-1}$. Observe that, for each meeting step j, we must have $x_j = y_j$, but not necessarily the opposite, and we call this node a MEETING POINT. The meeting steps are the steps when the agents meet and add at least one new node to the explored territory. A sequence of steps $\langle j+1, j+2, \ldots, k\rangle$ where j and k are two consecutive meetings is called a PHASE of length $k - j$. We give now the last constraint on a feasible exploration scheme.

Constraint 4: for each phase with a sequence of steps $\langle j+1, \ldots, k\rangle$,
(a) $|\{x_{j+1}, \ldots, x_k\} \setminus S_j| \le 1$ and $|\{y_{j+1}, \ldots, y_k\} \setminus S_j| \le 1$; and

(b) $\{x_{j+1}, \ldots, x_k\} \setminus S_j \neq \{y_{j+1}, \ldots, y_k\} \setminus S_j$.

Constraint 4(a) means that during each phase, one agent can visit at most one unexplored node. If it visited two unexplored nodes and one of them was a black hole, then the other agent would not know where exactly the black hole was. Constraint 4(b) says that the same unexplored node cannot be visited by both agents during the same phase, or otherwise they both may end up in a black hole (see [1]). From now on an exploration scheme means a feasible exploration scheme. The next two observations will be frequently used in our arguments.

Lemma 1. *If $k \geq 1$ is a meeting step for an exploration scheme $\mathcal{E}_{G,s}$, then $x_k = y_k \in S_{k-1}$.*

Proof. Let j be the last meeting step before step k, and hence $S_j = S_{j+1} = \ldots = S_{k-1}$. By definition $x_k = y_k \in S_k$. If $x_k = y_k$ is not in S_{k-1}, then it is in both $\{x_{j+1}, \ldots, x_k\} \setminus S_j$ and $\{y_{j+1}, \ldots, y_k\} \setminus S_j$. In this case, at least one of the conditions of Constraint 4 is violated. □

Lemma 2. *Each phase of an exploration scheme $\mathcal{E}_{G,s}$ has length at least two.*

Proof. Let us suppose, by contradiction, that there exists in $\mathcal{E}_{G,s}$ a phase of length 1, and hence two adjacent meeting steps j and $j+1$. The step $j+1$ is a meeting if and only if $S_{j+1} \supsetneq S_j$, but, by Lemma 1, $x_{j+1} = y_{j+1} \in S_j$, and hence $S_{j+1} = S_j$. Therefore there cannot exist in $\mathcal{E}_{G,s}$ a phase of length 1. □

We now present a notation for describing each phase of length 2, at the end of which the explored territory increases by 2 nodes. Any phase $\langle j+1, j+2 \rangle$ of this kind has to have the following structure. Let m be the meeting point at step j. During step $j+1$, *Agent*-1 visits an unexplored node v_1 adjacent to m, while *Agent*-2 visits an unexplored node v_2 adjacent to m as well, and $v_1 \neq v_2$. In step $j+2$, the agents meet in a node which has been already explored and is adjacent to both v_1 and v_2. This node can be either m, and in this case we denote the phase as ***b-split***(m, v_1, v_2), or a different node $m' \neq m$, and in this case we denote the phase as ***a-split***(m, v_1, v_2, m').

For an exploration scheme $\mathcal{E}_{G,s} = (\mathbb{X}, \mathbb{Y})$ and a location of a black hole B, where either $B = \emptyset$ or $B = \{b\}$ for $b \in (V \setminus \{s\})$, 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 (spending one time unit per step) and then get back to the starting node to report that there is no black hole in the network. If $B = \{b\}$, then let j be the first step in $\mathcal{E}_{G,s}$ such that $b \in S_j$. Observe that j must be a meeting step and $1 \leq j \leq T$ since $S_0 = \{s\}$ and $S_T = V$. One agent knows at step j that the other agent has died in b. The execution time in this case is equal to j plus the shortest length of a path from $x_j (= y_j)$ to s not including b. In this case one agent, say *Agent*-1, vanishes into the black hole during the phase ending at step j, so it does not show up to meet *Agent*-2 at node $x_j = y_j$. Since, by Constraint 4, *Agent*-1 has visited only one unexplored node during the phase, the surviving *Agent*-2 learns the exact location of the black hole and returns to s.

The COST of the BHS based on an exploration scheme $\mathcal{E}_{G,s} = (\mathbb{X}, \mathbb{Y})$ is the worst (maximum) execution time of $\mathcal{E}_{G,s}$ over all possible values of B. In other words, in computing the cost of a BHS, we allow a malicious adversary, which exactly knows $\mathcal{E}_{G,s}$, to place the black hole (or not to place it at all) in such a way that the BHS requires as many time units as possible. It is not difficult to see 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 (a detailed argument for this fact is included in the proof of Lemma 9). However, if G is an arbitrary graph, then this property does not always hold, that is, the case $B = \emptyset$ may not give the maximum execution time. For example, consider the n-node ring graph $\langle s, v_1, v_2, \ldots, v_{n-1}\rangle$ and the following exploration. *Agent*-1 goes to v_1 and back to s, and then, provided that *Agent*-1 returns, both agents go to v_1. The agents continue in this way exploring next v_2, then v_3, and so on, until they go all the way around the ring. If there is no black hole, then the execution time is $3n + O(1)$. If node v_{n-1} is the black hole, then the execution time is $4n + O(1)$ because the surviving agent returns to s by tracing back the whole ring.

To summarize, the objective of the BHS problem is to find, for a given graph G and a starting node s, an exploration scheme $\mathcal{E}_{G,s}$ which minimizes the cost of the BHS based on it. In Section 3 we prove that this problem is NP-hard, and in Section 4 we describe a $3\frac{3}{8}$-approximation algorithm.

3 NP-Hardness of Black Hole Search

In this section we prove the NP-hardness of the BHS problem in planar graphs by providing a reduction from a specific version of the Hamiltonian Cycle problem to the decision version of the BHS problem.

Hamiltonian Cycle problem for cubic planar graphs (cpHC problem)

Instance : a cubic planar 2-edge-connected graph $G = (V, E)$, and an edge $(x, y) \in E$;
Question : does G contain a Hamiltonian cycle that includes edge (x, y)?

Decision Black Hole Search problem for planar graphs (dBHS problem)

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

The NP-completeness of the cpHC problem without the extra requirement that the Hamiltonian cycle passes through a given edge was proven in [8]. The version with that extra requirement is also NP-complete because of the following simple reduction. For a given cubic planar graph G, let D be any node in G and let A, B and C be its neighbors. Add to G six new nodes and replace the edges adjacent to D with the edges as in Figure 1(a) to obtain graph $\tilde{G}$. It should be clear that if graph $\tilde{G}$ has a Hamiltonian cycle containing edge (x, y), then graph G has a Hamiltonian cycle as well. Figure 1(b) shows that the implication in the other direction is also true: if graph G has a Hamiltonian cycle, then graph $\tilde{G}$ has a Hamiltonian cycle containing edge (x, y).

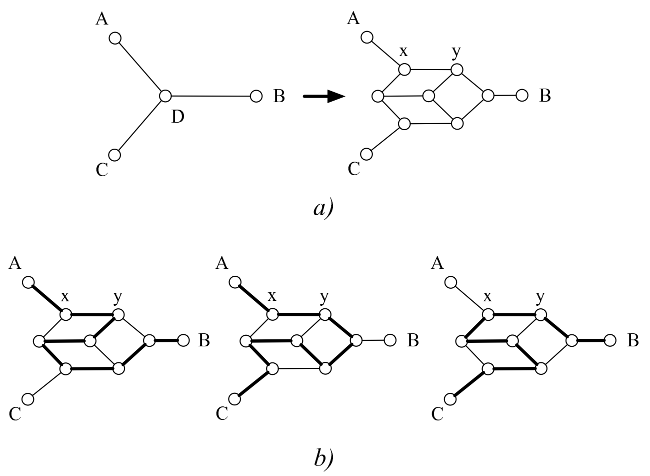

Fig. 1. a) Reduction from the cpHC problem with no fixed edge to the cpHC problem with a fixed edge (x, y). b) Extensions of Hamiltonian cycles in graph G to Hamiltonian cycles in graph $\tilde{G}$ passing through edge (x, y).

We describe now a polynomial time reduction from the cpHC problem to the dBHS problem. Let $G = (V, E)$ and $(x, y) \in E$ be an instance of the cpHC problem. We construct the corresponding instance of the dBHS problem, i.e., a graph G', a starting node s, and an integer X, by modifying graph G in the following steps.

1. Replace in G the edge (x, y) with the edges (x, s) and (s, y), where $s \notin V$ is a new node, obtaining graph $\bar{G}$.
2. Let $\mathcal{F}$ be the set of the faces of an arbitrary planar embedding of graph $\bar{G}$. We identify each face $f \in \mathcal{F}$ with the sequence of the consecutive edges adjacent to this face (starting with any edge adjacent to f and traversing the boundary of f in either of the two directions).
3. For each face $f \in \mathcal{F}$ and each edge (v, w) adjacent to f, add one new node $z_f^{(v,w)}$ and two edges $(v, z_f^{(v,w)})$ and $(w, z_f^{(v,w)})$.
4. For each face $f = \langle e_1, e_2, \ldots, e_q \rangle \in \mathcal{F}$ add the *shortcut edges* $(z_f^{e_1}, z_f^{e_2})$, $(z_f^{e_2}, z_f^{e_3}), \ldots, (z_f^{e_q}, z_f^{e_1})$.
5. For each node $v \in V \cup \{s\} \setminus \{x\}$, add a new node v^F, called the *flag node* of node v, and an edge (v, v^F).

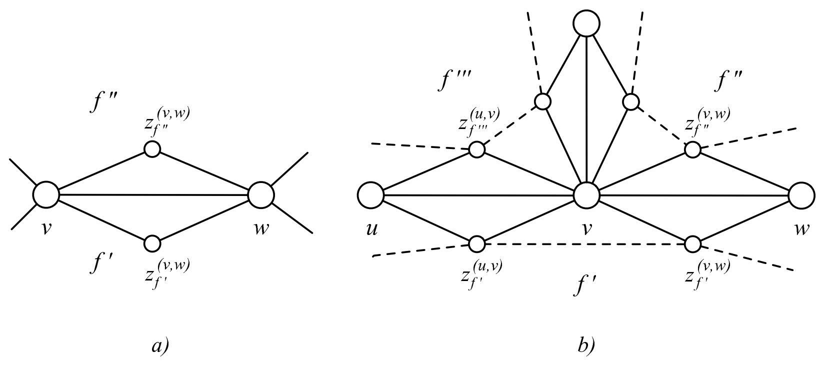

Fig. 2. In *a)*, the two twin nodes for the edge (v, w); in *b*), the twin nodes for the edges (u, v) and (v, w) and their neighborhood.

6. Let G' be the obtained graph. Set X to $n' - 1 = 5n + 2$, where $n' = n + 1 + 2(e+1)+n = 5n+3$ is the number of nodes in G' and n and e are, respectively, the number of nodes and edges in G (in a cubic graph, $e = (3/2)n$).

Since graph G is planar and 2-edge connected, each edge e in graph $\bar{G}$ is adjacent to exactly two different faces f' and f'' in $\mathcal{F}$. The two nodes $z^e_{f'}$ and $z^e_{f''}$ in G' added for edge e are called the *twin nodes* for edge e. The construction of graph G' is illustrated in Figure 2. Graph G' is planar and can be constructed in linear time. The nodes in G' inherited from graph $\bar{G}$ are called the *original nodes.*

The following lemma states one of the properties of graph G' which we use in further arguments.

Lemma 3. *Let $\langle u, v, w\rangle$ be a path in graph $\bar{G}$. Then there is a path $\langle u, z', z'', w\rangle$ in G' bypassing node v (that is $v \notin \{z', z''\}$).*

Proof. Since the degree of each node in $\bar{G}$ is at most 3, there must be a face $f \in \mathcal{F}$ to which both edges (u, v) and (v, w) are adjacent. By construction, the sequence of nodes $\left\langle u, z^{(u,v)}_f, z^{(v,w)}_f, w\right\rangle$ is a path in G'. □

Lemmas 4 and 5 prove that graph G has a Hamiltonian cycle passing through edge (x, y) if and only if there is an exploration scheme for graph G' and the starting node s with cost at most $X = 5n + 2$.

Lemma 4. *If graph G has a Hamiltonian cycle that includes edge (x, y), then there exists an exploration scheme $\mathcal{E}^*_{G',s}$ on graph G' from the starting node 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 $\mathcal{E}^*_{G',s}$ 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 node 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).

Let us compute the length of $\mathcal{E}^*_{G',s}$. Since *a-split* and *b-split* phases have length 2 and increase the explored territory by 2 nodes (see Section 2), the overall number of phases is $(5n+2)/2$ and hence $\mathcal{E}^*_{G',s}$ has length $5n+2$. Notice that this is also the exploration time for $\mathcal{E}^*_{G',s}$, in the case $B = \emptyset$, since $\mathcal{E}^*_{G',s}$ ends in s.

Now we prove that this is also the cost of the BHS based on $\mathcal{E}^*_{G',s}$, i.e. there is no allocation of the black hole that yields a larger exploration time. We first observe that the set of meeting points in $\mathcal{E}^*_{G',s}$ is $\{v_i : 1 \le i \le n\} \cup \{s\}$.

Claim. Consider the meeting step when the agents are to meet at a node v_i $(1 \le i \le n)$. If a black hole has been just discovered, then the remaining exploration time for this case is not greater than the remaining exploration time for the case $B = \emptyset$.

Proof. If the black hole is the flag node v_i^F (phase 3.b) or one of the twin nodes for the edge (v_{i-1}, v_i) or for the edge (v_i, v_j) (phase 3.c or 3.a), then the surviving agent can reach s by following the remaining part of the Hamiltonian Cycle, and hence the remaining cost is at most: $n + 1 - i$. If the black hole is at node v_{i+1} (phase 3.b), then, by Lemma 3, there is a path of length 4 in G' from v_i to v_{i+2} bypassing node v_{i+1} (where v_{i+2} is node s, if $i + 1 = n$). Therefore the surviving agent can reach node v_{i+2} (or s) by using this safe path and then, as before, he can follow the remaining part of the Hamiltonian Cycle to reach s. The remaining cost is at most $n + 2 - i$. If $B = \emptyset$, then the remaining cost is at least: $2(n + 1 - i) \ge n + 2 - i$. This concludes the proof of the claim.

Observe that the BHS defined above is optimal since, by Lemma 2, the exploration of $5n + 2$ nodes requires at least $5n + 2$ time units. □

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

Proof. By Lemma 2, each phase of $\mathcal{E}_{G',s}$ has length at least two and cannot explore more than two unexplored nodes. Since G' has $5n+2$ unexplored nodes, $\mathcal{E}_{G',s}$ must end in s, and each of its phases must be either an *a-split* or a *b-split*.

Consider now the sequence $M_{\mathcal{E}}$ of the meeting points established for $\mathcal{E}_{G',s}$ at the end of each *a-split*, excluding the last one which is s. Each meeting point v_i in $M_{\mathcal{E}}$ 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' can be in $M_{\mathcal{E}}$ (flag nodes have degree 1 and twin nodes have degree 4).

Claim. The nodes x and y must be the two endpoints of $M_{\mathcal{E}}$, node s cannot be in $M_{\mathcal{E}}$, and each node v in G must be in $M_{\mathcal{E}}$.

Proof. Since s is the only initially safe node, the very first phase has to be a *b-split* from s. The first *a-split* in $\mathcal{E}_{G',s}$ is from s to x or y, while the last *a-split* (ending in s) starts from the other of these two nodes x, y. If s is also an intermediate meeting point, then we need another *a-split* to s. Since each of these four phases requires two unexplored neighbors, s has to have degree at least 8, but, by construction, its degree is only 7. Contradiction.

Finally, for each node v in G, its flag node v^F has to be explored with a *b-split* having as meeting point node v. Hence v must be in $M_{\mathcal{E}}$.

Now we prove that the sequence $M_{\mathcal{E}}$ defines a Hamiltonian cycle on G by showing that it has also the following two properties:

a) each node of G appears at most once in $M_{\mathcal{E}}$;

b) if nodes v_i and v_j are consecutive in $M_{\mathcal{E}}$, then the edge (v_i, v_j) must be in G.

To prove *a)*, it suffices to count the number of neighbors needed by a node v_i in $M_{\mathcal{E}}$. 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 $M_{\mathcal{E}}$, two unexplored neighbors are needed for the *a-split* that ends in v_i, and two additional unexplored neighbors are needed for the *a-split* that leaves v_i. Moreover the flag node 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 $M_{\mathcal{E}}$, 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 at most once in $M_{\mathcal{E}}$.

Now we prove property *b)* of $M_{\mathcal{E}}$. According to the structure of G', *a-split* operations having original nodes as meeting points, can either explore two twin nodes of an original edge (in this case property *b)* is satisfied 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 3. 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 $M_{\mathcal{E}}$). At this point no other *big a-splits* can be performed from B (all its original neighbors are now explored) and, by property *a)*, E cannot be again a meeting point, thus the sequence $M_{\mathcal{E}}$

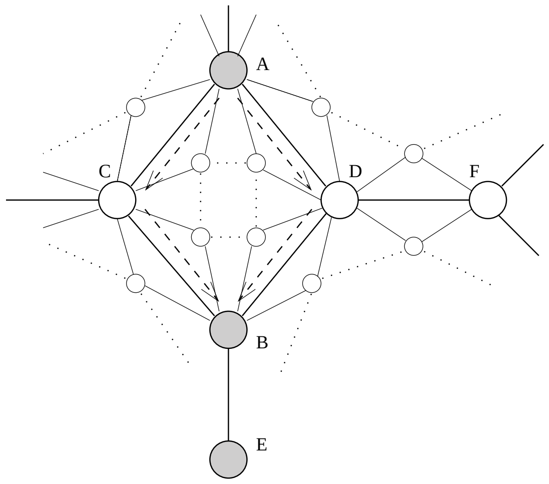

Fig. 3. A big *a-split* from A to B. Flag nodes are not shown, the shaded nodes are already explored.

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 $M_{\mathcal{E}}$, 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 $\mathcal{E}_{G',s}$, and thus property *b)* is verified.

We have proved that, if there exists an exploration scheme $\mathcal{E}_{G',s}$ for G', such that the BHS based on $\mathcal{E}_{G',s}$ has cost $5n+2$, then G has a Hamiltonian cycle that includes edge (x, y). □

Lemma 4, Lemma 5 and the fact that the cpHC problem is NP-hard imply the following theorem.

Theorem 1. *The* dBHS *problem for planar graphs is NP-hard.*

4 An Approximation Algorithm for the BHS Problem in Arbitrary Graphs

We consider the following natural approach to the BHS problem in an arbitrary graph G. First select a spanning tree in G and then explore the graph by traversing the tree edges. As observed in [1], this approach guarantees an approximation ratio of 4 since any exploration of an n-node graph requires at least $n-1$ steps while the following scheme explores an n-node tree within $4(n-1)-2l$ steps, where l is the number of leaves in the tree. 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 an algorithm for constructing "good" exploration schemes for trees and an algorithm for computing spanning trees which are "good" for those schemes. Czyzowicz *et. al.* [1] showed a linear-time algorithm for constructing optimal exploration schemes for trees where each internal node has at least 2 children (called *bushy trees* in [1]). In Section 4.1 we describe a linear-time algorithm *Search-Tree*(T, s) which extends the construction from [1] to the general rooted trees. This algorithm does not guarantee optimality of computed exploration schemes for trees other than bushy trees: the question of computing in polynomial time optimal exploration schemes for general trees remains open. We give a formula for the cost of the exploration scheme computed by our algorithm *Search-Tree*(T, s) as a function of the number of nodes of different types in tree T (Lemma 10). In Section 4.2 we present a heuristic algorithm *Generate-Tree*(G, s) for the problem of computing a rooted spanning tree T of graph G which gives a relatively small value of that formula.

Our *Spanning-Tree Exploration* (*STE*) algorithm returns, for a given graph G and a starting node s, the exploration scheme computed by *Search-Tree*(T_G, s), where T_G is the spanning tree computed by *Generate-Tree*(G, s). In Section 4.3 we show that the *STE* algorithm guarantees an approximation ratio of at most $3\frac{3}{8}$. In Section 4.4 we remark on other possible variants of exploring graphs via spanning trees.

4.1 Exploration Schemes for Trees

Let T be an n-node tree rooted at node s. We assume that $n \geq 2$. The exploration scheme for T constructed by our algorithm *Search-Tree*(T, s) may be viewed in the following way. For each internal node p in T, if p has x children, then they are partitioned into two groups of size $\lceil x/2 \rceil$ and $\lfloor x/2 \rfloor$. Both agents follow the depth-first traversal of the internal nodes of T, and whenever *Agent*-1 (*Agent*-2) comes during this traversal to an internal node p for the first time, it visits all children of p in group 1 (group 2) before continuing the traversal. Obtaining an efficient exploration scheme based on this approach and proving its correctness and cost turns out to be quite technical.

We use the following order L_T of the nodes of T other than the root (that is, all unexplored nodes in T). 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 $I_T = \langle w_1, w_2, \ldots, w_b \rangle$ be the sequence of the internal nodes of T in the depth-first order. The order L_T is this sequence with each node w_i replaced with the (ordered) list of its children. Observe that L_T contains indeed all nodes of tree T other than the root, and each of these nodes occurs in L_T exactly once. We denote the i-th node in the order L_T by v_i and call it the i-th node of the tree. The odd (even) nodes of T are the nodes at the

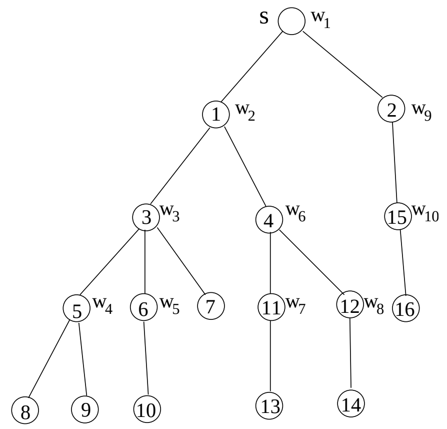

Fig. 4. An ordered rooted tree T. The value inside each node is the position of the node in the L_T order. The internal nodes are also marked to show their depth-first order $I_T = \langle w_1, w_2, \ldots, w_{10} \rangle$.

odd (even) positions in L_T. We denote the parent of node v_i by p_i. An example tree T and the L_T order of its nodes is given in Figure 4.

The two lemmas below, which follow from the construction of the sequence L_T, will be used to prove that algorithm *Search-Tree* returns feasible exploration schemes for trees.

Lemma 6. *In the sequence L_T, let the j-th node v_j be the parent of the i-th node v_i. Then $j < i$, and $i = j + 1$ if and only if node v_j does not have a sibling and node v_i is its first child.*

Proof. The parent p_j of node v_j precedes node v_j in the depth-first order I_T of the internal nodes. Thus all children of p_j, including node v_j, precede all children of v_j, including node v_i, in the sequence L_T, so $j < i$.

If node v_j does not have a sibling, then v_j must be immediately after p_j in the sequence I_T. In this case, when the sequence L_T is created from $I_T = \langle \ldots, p_j, v_j, \ldots \rangle$, the occurrence of node p_j in I_T is replaced with (its only child) v_j, while the occurrence of node v_j in I_T is replaced with the ordered list of its children. Thus if node v_i is the first child of node v_j, then v_i is immediately after v_j in the sequence L_T, that is, $i = j + 1$.

If node v_j has a right sibling r, then node r is after node v_j and before node v_i in L_T, so $i > j+1$. If node v_j has a left sibling l, then node l must have at least one child since the siblings are ordered according to the number of descendants and node v_j has at least one descendant. The children of node l are after node v_j and before node v_i in L_T, so $i > j + 1$. If node v_i is not the first child of node v_j, then all left siblings of v_i are after node v_j and before node v_i in L_T, so also in this case $i > j + 1$. □

Lemma 7. *Let v_i and v_{i+1} be two consecutive nodes in the sequence L_T, and let p_i and p_{i+1} be their parents. Then either nodes v_i and v_{i+1} are siblings, so $p_i = p_{i+1}$, or node p_{i+1} is the next node after node p_i in the depth-first order I_T of the internal nodes of T.*

Proof. Assume that nodes v_i and v_{i+1} are not siblings. Node p_i must occur in I_T before node p_{i+1}. If there was another (internal) node between p_i and p_{i+1} in I_T, then the children of this node would be between nodes v_i and v_{i+1} in L_T. □

We classify all nodes of tree T other than the root s into the following three disjoint types:

– *type-1* nodes: the leaves;
– *type-3* nodes: the internal nodes with at least one sibling;
– *type-4* nodes: the internal nodes (other than the root) without siblings.

Informally speaking, in the exploration scheme which we construct for tree T a *type-t* node can be viewed as contributing t steps to the total cost. Note that there are no *type-2* nodes. We denote by x_t the number of *type-t* nodes.

We consider first the case when T does not have any *type-4* node and has an odd number $n = 2q + 1 \geq 3$ of nodes (that is, tree T has an even number of unexplored nodes $v_1, v_2, \ldots, v_{2q}$). *Agent*-1 (*Agent*-2) will be following the depth-first traversal of the internal nodes of T, and whenever it comes to an internal node p for the first time, it will visit all children of p which are odd (even) nodes in T before continuing the traversal. We now formally specify this exploration scheme.

For nodes u and r in tree T, let $P(u, r\rangle$ be the sequence of the nodes on the tree path from u to r excluding the first node u. If $u = r$, then $P(u, r\rangle$ is the empty sequence. The exploration sequences $\mathbb{X}_T$ and $\mathbb{Y}_T$ for *Agent*-1 and *Agent*-2, respectively, are

$$\mathbb{X}_T = \langle s\rangle \circ \phi^1_1 \circ \phi^1_2 \circ \cdots \circ \phi^1_q,$$
$$\mathbb{Y}_T = \langle s\rangle \circ \phi^2_1 \circ \phi^2_2 \circ \cdots \circ \phi^2_q;$$

where

$$\phi^1_j = P(p_{2j-2}, p_{2j-1}\rangle \circ \langle v_{2j-1}, p_{2j-1}\rangle \circ P(p_{2j-1}, p_{2j}\rangle,$$
$$\phi^2_j = P(p_{2j-2}, p_{2j-1}\rangle \circ P(p_{2j-1}, p_{2j}\rangle \circ \langle v_{2j}, p_{2j}\rangle.$$

In the above formulas operation "$\circ$" is the concatenation of sequences, and we define $p_0 = s$. Note that the corresponding sub-sequences ϕ^1_j and ϕ^2_j in $\mathbb{X}_T$ and $\mathbb{Y}_T$ have the same length and end at the same node p_{2j}. In fact, we will show that ϕ^1_j and ϕ^2_j form the j-th phase of the exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ (Lemma 8). Figure 5 shows different types of relative locations of nodes v_{2j-2}, v_{2j-1}, v_{2j}, p_{2j-2}, p_{2j-1} and p_{2j}, which lead to different types of sequences ϕ^1_j and ϕ^2_j.

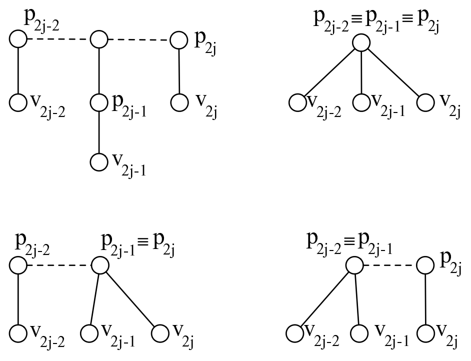

Fig. 5. Different relative positions of nodes v_{2j-2}, v_{2j-1} and v_{2j} consecutive in the L_T order and their parents p_{2j-2}, p_{2j-1} and p_{2j}. The tree does not have *type-4* nodes. The dashed lines represent paths. For examples of the two cases in the lower row, take $2j = 12$ and $2j = 10$ in Figure 4, respectively.

Observe that if we remove from sequences $\mathbb{X}_T$ and $\mathbb{Y}_T$ all segments $\langle v_{2j-1}, p_{2j-1}\rangle$ and $\langle v_{2j}, p_{2j}\rangle$, then both $\mathbb{X}_T$ and $\mathbb{Y}_T$ become the following sequence

$$\langle s\rangle \circ P\langle p_0, p_1\rangle \circ P\langle p_1, p_2\rangle \circ \cdots \circ P\langle p_{2q-1}, p_{2q}\rangle.$$

Lemma 7 implies that this sequence is the depth-first traversal of the internal nodes of tree T ending when the last internal node is visited.

We prove now that $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ is a feasible exploration scheme for tree T. It is straightforward to check that $\mathcal{E}_T$ satisfies the feasibility Constraints 1–3. The lemma below identifies the phases of scheme $\mathcal{E}_T$ and states that each phase satisfies the conditions given in Constraint 4.

Lemma 8. *For each $j = 1, 2, \ldots, q$, the sub-sequences ϕ^1_j and ϕ^2_j within $\mathbb{X}_T$ and $\mathbb{Y}_T$ form the j-th phase of the feasible exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$, and this phase satisfies the conditions stated in the feasibility Constraint 4.*

Proof. Let $m(0) = 0$, and for $j = 1, 2, \ldots, q$, let $m(j)$ denote the step in $\mathcal{E}_T$ where the sub-sequences ϕ^1_j and ϕ^2_j end. That is, the sub-sequences ϕ^1_j and ϕ^2_j occur within $\mathbb{X}_T$ and $\mathbb{Y}_T$, respectively, at the steps $\langle m(j-1)+1, \ldots, m(j)\rangle$. We prove by induction that for each $j = 1, \ldots, q$, the following statements are true.

1. The explored territory at step $m(j)$ is $S_{m(j)} = \{s, v_1, \ldots, v_{2j}\}$.
2. The sequence of steps $\langle m(j-1)+1, \ldots, m(j)\rangle$ in scheme $\mathcal{E}_T$ (where the sub-sequences ϕ^1_j and ϕ^2_j occur) is a phase and satisfies Constraint 4.

Note that, $S_{m(0)} = S_0 = \{s\}$. For the base step ($j = 1$), observe that

$$\phi^1_1 = P\langle p_0, p_1\rangle \circ \langle v_1, p_1\rangle \circ P\langle p_1, p_2\rangle = \langle v_1, s\rangle,$$
$$\phi^2_1 = P\langle p_0, p_1\rangle \circ P\langle p_1, p_2\rangle \circ \langle v_2, p_2\rangle = \langle v_2, s\rangle,$$

because $p_0 = p_1 = p_2 = s$. Thus $m(1) = 2$, $S_{m(1)} = \{s, v_1, v_2\}$, and the steps $\langle 1, 2\rangle$ form a phase satisfying Constraint 4 (this phase is *b-split*(s, v_1, v_2)) so both Statements 1 and 2 hold.

Consider now any index j, $1 \le j \le q$ and assume that both Statements 1 and 2 are true for $j-1$. This assumption implies that $S_{m(j-1)} = \{s, v_1, v_2, \ldots, v_{2j-2}\}$ and that step $m(j-1)$ is a meeting step. (If $j \ge 2$, then $m(j-1)$ is a meeting step as the last step of the phase $\langle m(j-2)+1, \ldots, m(j-1)\rangle$. If $j = 1$, then step $m(j-1) = 0$ is by definition a meeting step.) By the definition of sequences $\mathbb{X}_T$ and $\mathbb{Y}_T$, the agents are at step $m(j-1)$ at the node p_{2j-2} (the parent of the node v_{2j-2}, or s if $j = 1$). Now *Agent*-1 and *Agent*-2 follow the sequences of nodes ϕ_j^1 and ϕ_j^2, respectively. Lemma 6 implies that the nodes p_{2j-2} and p_{2j-1} are in $S_{m(j-1)}$. Lemma 6 also implies that $p_{2j} \in S_{m(j-1)}$: if $p_{2j} \ne s$, then p_{2j} has a sibling, so p_{2j} is a node v_k for some $k \le 2j-2$. Applying again Lemma 6, we conclude that all nodes in the sequences $P(p_{2j-2}, p_{2j-1}\rangle$ and $P(p_{2j-1}, p_{2j}\rangle$ must be in $S_{m(j-1)}$ as well, since each node in any of these two sequences is an ancestor of at least one of the nodes p_{2j-2}, p_{2j-1} and p_{2j}. Thus the only nodes in ϕ_j^1 and ϕ_j^2 which are not in $S_{m(j-1)}$ are node v_{2j-1} in ϕ_j^1 and node $v_{2j} \ne v_{2j-1}$ in ϕ_j^2. Therefore $S_{m(j)} = S_{m(j-1)} \cup \{v_{2j-1}, v_{2j}\}$ (so Statement 1 holds for j) and the sequence of steps $\langle m(j-1)+1, \ldots, m(j)\rangle$ satisfies Constraint 4. It remains to show that step $m(j)$ is the first meeting step after the meeting step $m(j-1)$, that is, to show that step $m(j)$ is the first step after step $m(j-1)$ when the explored territory increases.

Follow the agents' routes at steps $m(j-1)+1, \ldots, m(j)$ (see the diagrams in Figure 5). At the end of step $m(j-1)$ both agents are at the node p_{2j-2}, then they traverse together the (possibly empty) sequence of nodes $P(p_{2j-2}, p_{2j-1}\rangle$, not increasing the explored territory, and then they separate and meet again for the first time at step $m(j)$ at the node p_{2j}. At that step the explored territory increases from $S_{m(j-1)}$ to $S_{m(j)}$. Thus the sequence of steps $\langle m(j-1)+1, \ldots, m(j)\rangle$ is a phase in $\mathcal{E}_T$, so Statement 2 holds for j. This concludes the proof of the inductive step.

The lemma follows immediately from Statements 1 and 2. □

Lemma 9. *Let T be a tree rooted at s which has an odd number $n = 2q+1 \ge 3$ of nodes and does not have any* type-4 *nodes. The exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ is feasible, can be constructed in linear time, and the cost of the BHS based on $\mathcal{E}_T$ is equal to $x_1 + 3x_3$, where x_t denotes the number of* type-t *nodes in T.*

Proof. The feasibility of the exploration scheme $\mathcal{E}_T$ follows from Lemma 8. The execution time of this scheme in the case when there is no black hole is equal to the length of $\mathcal{E}_T$ plus the distance from p_{2p} to s, that is, the length of the sequence $\mathbb{Y}_T \circ P(p_{2q}, s\rangle$ minus 1. This is also the cost of the BHS based on $\mathcal{E}_T$, since generally for any feasible exploration scheme for a tree, the case when there is no black hole gives the worst execution time of the BHS. In fact, if there is a black hole, say at node v, then the surviving agent can keep following its part of the exploration scheme, replacing all occurrences of v and its descendants with the parent of v, and reaching s within the same number of steps.

To obtain the length of the sequence $\mathbb{Y}_T \circ P(p_{2q}, s\rangle$, we separate it into two sub-sequences:

$$\langle s\rangle \circ P(p_0, p_1\rangle \circ P(p_1, p_2\rangle \circ \cdots \circ P(p_{2q-1}, p_{2q}\rangle \circ P(p_{2q}, s\rangle, \quad \text{and}$$
$$\langle v_2, p_2\rangle \circ \langle v_4, p_4\rangle \circ \cdots \circ \langle v_{2q}, p_{2q}\rangle.$$

Lemma 7 implies that the first sub-sequence is the depth-first traversal of the b internal nodes of T, so its length is $2b-1$. The length of the second sequence is $2q = n-1$. Thus the cost of the exploration scheme $\mathcal{E}_T$ is $(2b-1)+(n-1)-1 = (n-1)+2(b-1) = (x_1+x_3)+2x_3 = x_1+3x_3$.

Sequences $\mathbb{X}_T$ and $\mathbb{Y}_T$ can be constructed in time linear in the length of these sequences, so linear in the size of tree T. □

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 as a sibling of v. If the total number of nodes, including the added nodes, is even, then we add one more leaf to an arbitrary internal node. The obtained tree T' is rooted at s, has an odd number of nodes and does not have any *type-4* nodes, so it satisfies the requirements of Lemma 9. We obtain an exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ for tree T from the exploration scheme $\mathcal{E}_{T'} = (\mathbb{X}_{T'}, \mathbb{Y}_{T'})$ for tree T' by replacing the traversals of the added edges with waiting. More precisely, if a node l is an added leaf, its parent is a node p, and l is an odd (even) node in tree T', then replace the unique occurrence of l in $\mathbb{X}_{T'}$ (in $\mathbb{Y}_{T'}$) with p.

Lemma 10. *Let T be a tree rooted in s with $n \geq 2$ nodes. The exploration scheme $\mathcal{E}_T = (\mathbb{X}_T, \mathbb{Y}_T)$ for T is feasible, can be constructed in linear time and its cost is at most*

$$x_1 + 3x_3 + 4x_4 + 1. \tag{1}$$

Proof. The feasibility of the exploration scheme $\mathcal{E}_T$ and its construction in linear time follow from Lemma 9. Let β be equal to 1 if the extra node was added to the tree to have an odd number of nodes, and 0 otherwise. The cost of scheme $\mathcal{E}_T$ is equal to the cost of scheme $\mathcal{E}_{T'}$. Lemma 9 implies that the cost of scheme $\mathcal{E}_{T'}$ is equal to $x'_1 + 3x'_3$, where $x'_1 = x_1 + x_4 + \beta$ is the number of leaves in tree T' and $x'_3 = x_3 + x_4$ is the number of *type-3* nodes in tree T' (each *type-4* node in tree T becomes a *type-3* node in tree T'). Thus the cost of scheme $\mathcal{E}_T$ is equal to $x'_1 + 3x'_3 = x_1 + 3x_3 + 4x_4 + \beta$. □

It can be shown that the cost of our exploration scheme $\mathcal{E}_T$ is at most $4/3 + O(1/n)$ times the optimal cost of an exploration scheme for T (see [11]). This improves the 5/3 approximation ratio bound of the exploration scheme for a tree presented in [1]. Our exploration scheme $\mathcal{E}_T$ could be further improved in some cases. For example, for the first diagram in Figure 5, *Agent*-2 obviously does not have to go to node p_{2j-1} on its way to explore node v_{2j}. If it omitted node p_{2j-1}, then the phase would have one step less (the agents would meet at the end of this phase in the predecessor of p_{2j} in the path $P(p_{2j-1}, p_{2j}\rangle$) and this local gain could reduce in some cases the overall cost of the search. However, this and similar improvements do not seem to lead to a tighter worst-case bound than the bound (1), which we use to bound the worst-case approximation ratio of algorithm *STE*. We also do not know how such improvements could decrease the $4/3 + O(1/n)$ approximation bound of $\mathcal{E}_T$.

4.2 Generating a Good Spanning Tree of a Graph

We describe now our heuristic algorithm *Generate-Tree*(G, s) for computing a spanning tree T_G of a graph $G = (V, E)$ rooted at a node $s \in V$ which tries to achieve a relatively small value for the formula (1). We believe that computing a rooted spanning tree which minimizes this formula is NP-hard, since the related problem of computing a spanning tree which maximizes the number of leaves is NP-hard [7]. In Section 4.3 we show that the exploration scheme constructed by algorithm *Search-Tree*(T_G, s) for the spanning tree T_G computed by algorithm *Generate-Tree*(G, s) yields a BHS with cost at most $3\frac{3}{8}$ times worse than the cost of an optimal BHS for graph G. If G is a path with s as an end node, then the optimal exploration scheme is obvious. Therefore we assume throughout this section that graph G is not of this form.

Algorithm *Generate-Tree*(G, s) tries to obtain a spanning tree with a small value of the formula (1) by trying to avoid creation of *type-4* nodes. More precisely, the algorithm grows in a greedy manner a spanning tree T, starting from node s, avoiding creation of internal nodes with only one child. A single child is a *type-4* node, unless it is a leaf. For the computation of the algorithm, let V_T denote always the set of nodes in the current tree T and let $\overline{V}_T = V \setminus V_T$; initially $V_T = \{s\}$. With respect to tree T, each node in V is either an *internal node*, or a *leaf*; it is an *external node* if it belongs to the set $\overline{V}_T$. An *external neighbor* of a node $u \in V$ is a neighbor of u in graph G which belongs to $\overline{V}_T$.

The pseudocode of algorithm *Generate-Tree* is given below. The algorithm consists of two parts. During part 1, the algorithm iteratively extends the current tree T rooted at s for as long as there is an *expandable leaf* in T or there is an *expandable external node* in $\overline{V}_T$. An *expandable leaf* in tree T is a leaf which has at least two external neighbors. An *expandable external node* (w.r.t. T) is a node in $\overline{V}_T$ which has at least one neighbor in T and at least two external neighbors, or has at least three external neighbors. The loop in part 1 of the algorithm maintains the following invariant: for the current tree T, there is no edge in G between an internal node and an external node. That is, each edge in G between the sets V_T and $\overline{V}_T$ is adjacent to a leaf of T.

If there is an expandable leaf in tree T, then extend T by selecting an arbitrary expandable leaf u and attaching to it all its external neighbors (see the left diagram in Figure 6). If there is no expandable leaf in T but there is an expandable external node, then extend T in the following way. Let $P = (u_1, u_2, \ldots, u_k)$ be a path in G consisting of external nodes such that node u_1 is the only node on P adjacent to T and node u_k is the only expandable external node on P. Let u_0 be a node in T adjacent to u_1 and let $w_1, w_2, \ldots, w_k$ be the neighbors of u_k which are neither in T nor on P. According to the invariant of the loop, node u_0 must be a leaf in tree T. Extend tree T by attaching path P to node u_0 and

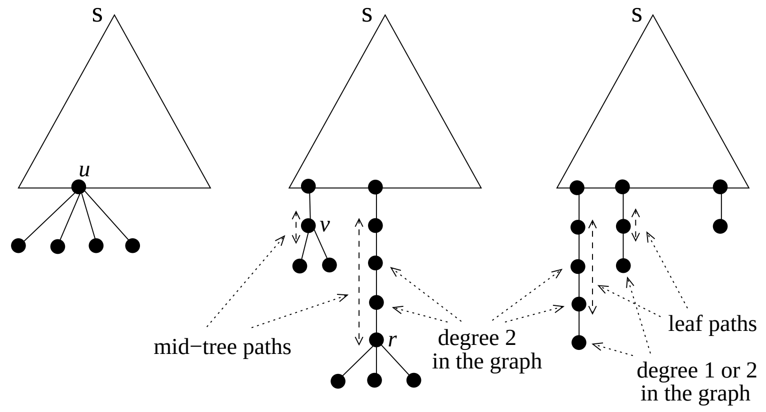

Fig. 6. Expansion of the tree during the computation of algorithm *Generate-Tree*: in part 1 of the algorithm using an expandable leaf u (the left diagram) and using mid-tree paths to expandable external nodes v and r (the middle diagram); and in part 2 of the algorithm (the right diagram).

nodes $w_1, w_2, \ldots, w_k$ as children of u_k. Path P, as a part of the new extended tree and a part of the final tree T_G, is called a *mid-tree path*. The middle diagram in Figure 6 illustrates the expansion of the tree using mid-tree paths.

Let T_1 denote the tree T at the end of part 1 of the algorithm. Since no expandable external node is left, each connected component of the subgraph of graph G induced by the set of external nodes must be now a path. Moreover, for each such path P, no node of P other than an end node is adjacent to T_1 (or otherwise such a node would be an expandable external node) but at least one end node of P is adjacent to tree T_1 (since G is connected). Let $\mathcal{P}$ denote the collection of these paths. If a path $P \in \mathcal{P}$ has at least two nodes and both end nodes are adjacent to T_1, then we replace P in $\mathcal{P}$ with paths P' and P'' obtained from P by removing the middle edge (or any of the two middle edges, if P has an odd number of nodes). Now for each path $P = (w_1, w_2, \ldots, w_k) \in \mathcal{P}$ where w_1 is adjacent to T_1 (exactly one end node of P is adjacent to T_1), we extend T by attaching P to a neighbor of w_1 in T_1, which must be a leaf in T_1 (see the last diagram in Figure 6). If path P has at least two nodes, then we call this path without the last node w_k a *leaf path*. When all paths from $\mathcal{P}$ are attached to tree T, tree T becomes a spanning tree T_G of G, and this tree is returned by the algorithm.

The whole algorithm *Generate-Tree* can be easily implemented to run in polynomial time, and it actually can be implemented to run in linear time. An example of a spanning tree produced by the algorithm is given in Figure 7. The next two lemmas summarize the properties of the algorithm which are important in our analysis.

Algorithm 1 Algorithm Generate-Tree (G, s)

```
1: $V \leftarrow$ set of nodes in $G$;  $E \leftarrow$ set of edges in $G$;
2: $T \leftarrow \emptyset$;  {the edges of the current tree}
3: let $V_T$ denote the set of nodes in $T$ (initially $V_T = \{s\}$), and let $\overline{V}_T = V \setminus V_T$;
4: {*Part 1: grow $T$ until there is no expandable leaf or expandable external node.*}
5: **loop**
6:    **if** there exists an expandable leaf in $T$ **then**
7:       $u \leftarrow$ an expandable leaf in $T$;
8:       $W \leftarrow$ the set of neighbors of $u$ in $\overline{V}_T$;
9:       $T \leftarrow T \cup \{(u, w) : w \in W\}$;
10:   **else if** there exists an expandable external node in $\overline{V}_T$ **then**
11:      $P = (u_1, \ldots, u_k) \leftarrow$ a path in $G$ such that each $u_i \in \overline{V}_T$, $u_1$ is the only node
         on $P$ adjacent to $T$ and $u_k$ is the only expandable external node on $P$;
12:      $u_0 \leftarrow$ a leaf in $T$ adjacent to $u_1$;
13:      $W \leftarrow$ the set of neighbors of $u_k$ which are neither in $T$ nor on $P$;
14:      $T \leftarrow T \cup \{(u_0, u_1)\} \cup P \cup \{(u_k, w) : w \in W\}$;
15:      { $P$ is a *mid-tree path* in $T$ }
16:   **else**
17:      exit the loop;
18:   **end if**
19: **end loop**
20: {*Part 2: attach to $T$ the remaining paths.*}
21: $T_1 \leftarrow T$;
22: $\mathcal{P} \leftarrow$ the set of connected components (paths) in the subgraph induced by $\overline{V}_T$;
23: **for all** $P = (u_1, u_2, \ldots, u_j) \in \mathcal{P}$, where $j \geq 2$ and $u_1$ and $u_j$ adjacent to $T_1$ **do**
24:    let $P' = (u_1, \ldots, u_k)$ and $P'' = (u_{k+1}, \ldots, u_j)$, where $k = \lfloor j/2 \rfloor$;
25:    $\mathcal{P} \leftarrow \mathcal{P} \setminus \{P\} \cup \{P', P''\}$;
26: **end for**
27: **for all** $P = (w) \in \mathcal{P}$ **do**
28:    $u \leftarrow$ a leaf in $T_1$ adjacent to $w$;  $T \leftarrow T \cup \{(u, w)\}$;
29: **end for**
30: **for all** $P = (u_1, u_2, \ldots, u_k) \in \mathcal{P}$, where $k \geq 2$ and $u_1$ adjacent to $T_1$ **do**
31:    $u_0 \leftarrow$ a leaf in $T_1$ adjacent to $u_1$;  $T \leftarrow T \cup \{(u_0, u_1)\} \cup P$;
32:    { path $(u_1, u_2, \ldots, u_{k-1})$ is a *leaf path* in $T$ }
33: **end for**
34: **return** $T$.
```

Lemma 11. *Consider any iteration of the loop in part 1 of algorithm Generate-Tree(G, s), and the current tree T at the beginning of this iteration. The following two properties hold.*

1. *No internal node of T is adjacent in G to any external node.*
2. *Each leaf in T has a sibling, unless this is the first iteration of the loop (when T contains only the root s).*

Proof. At the beginning of the first iteration of the loop, tree T does not have any internal nodes, so both Statements 1 and 2 are obviously true. Let T' be the tree T at the beginning of one iteration of the loop other than the last one, and let T'' be the tree T at the beginning of the next iteration. Assume inductively that Statements 1 and 2 are true for tree T'. Tree T'' is obtained from tree T' by adding children to an expandable leaf (lines 7–9 in the pseudocode) or, if T' does not have an expandable leaf, by adding a mid-tree path and children of the last node on this path (lines 11–14).

Consider the first case: tree T'' is obtained from T' by adding children to an expandable leaf u. Node u is the only new internal node in T'' and its children are the only new leaves. All neighbors of node u are now in T'', so Statement 1 is true for T''. Node u gets at least two children since u is an expandable leaf in tree T', so also Statement 2 is true for T''.

Consider now the second case: tree T' does not have an expandable leaf and tree T'' is obtained from tree T' by attaching a mid-tree path $P = (u_1, \ldots, u_k)$ to a leaf u_0 and attaching all remaining neighbors of u_k (the neighbors neither in tree T' nor on path P) as children of u_k. We check first that the new internal nodes $u_0, u_1, \ldots, u_k$ in tree T'' have all their neighbors in T''. Clearly node u_k has all its neighbors in tree T''. Node u_0 cannot have neighbors outside of T' other than node u_1 since node u_0 is not an expandable leaf in T'. If $k \geq 2$, then node u_1 cannot have neighbors outside T' other than u_2 since u_1 is not an expandable external node. If $k \geq 3$, then for each $i = 2, \ldots, k-1$, node u_i is not adjacent to T' and is not an expandable external node, so nodes u_{i-1} and u_{i+1} can be its only neighbors in graph G. Thus each new internal node in T'' has all its neighbors in T'', so Statement 1 holds for T''.

The new leaves in T'' are the children of u_k. Since u_k is an expandable external node (w.r.t. T'), it gets at least two children in T''. Indeed, if $k = 1$, then, by definition of expandable external node, node u_1 must have at least two external neighbors, which become its children in T''. If $k \geq 2$, then node u_k is not adjacent to tree T', so it must have at least 3 external neighbors. One of them is node u_{k-1} while the remaining ones are the children of u_k in T''. Thus Statement 2 holds for T''. □

Lemma 12. *Let T_1 denote the tree T at the end of part 1 of Algorithm Generate-Tree(G, s) and let $\mathcal{P}$ denote the set of connected components of the subgraph G' of graph G induced by the external nodes (w.r.t. T_1).*

1. *For each connected component of subgraph G', the edges of this component form a (simple) path.*

2. For each path $P \in \mathcal{P}$,
 (a) the internal nodes of P are not adjacent to tree T_1;
 (b) at least one end node of P is adjacent to tree T_1.

Proof. There is no expandable external node w.r.t. T_1. Thus each node in subgraph G' has degree at most 2 in G', since otherwise such a node would be an expandable external node. Therefore each connected component of G' is either a path (possibly a single node) or a cycle. However, if a connected component of G' were a cycle, then there would be a node on this cycle adjacent to tree T_1, since graph G is connected, and this node would be an expandable external node.

For a path P which is a connected component of subgraph G', if a node on P other than an end node were adjacent to tree T_1, then this node would be an expandable external node. Since graph G is connected, at least one end node of P must be adjacent to tree T_1. □

We look now at the *type-4* nodes in T_G to see how they were created and what their properties in graph G are. We view the mid-tree paths and the leaf paths in T_G in the direction from the root towards the leaves. That is, the first node on such a path is the node closest to the root.

Lemma 13. *A node in tree T_G is a* type-4 *node if and only if it belongs to a mid-tree path or a leaf path.*

Proof. Examine all possible extensions of the current tree T to a new tree T' during the computation of algorithm *Generate-Tree*.

In line 9 of the algorithm, node u changes its status from *type-1* in tree T to *type-3* in tree T' (Property 2 in Lemma 11 implies that u has a sibling in tree T) and all new nodes in tree T' are *type-1* nodes. In line 14, node u_0 changes its status from *type-1* in tree T to *type-3* in tree T', the new nodes $u_1, u_2, \ldots, u_k$, which form a mid-tree path, are *type-4* nodes in tree T', and the leaves attached to u_k are *type-1* nodes in tree T'. In line 28, node u changes its status from *type-1* in tree T to *type-3* in tree T' (Property 2 of Lemma 11 implies that u has a sibling in the tree T_1 constructed during the first part of the algorithm) and the new node w is a *type-1* node in tree T'. In line 31, node u_0 changes its status from *type-1* in tree T to *type-3* in tree T', the new nodes $u_1, u_2, \ldots, u_{k-1}$, which form a leaf path, are *type-4* nodes in tree T', and the leaf u_k attached to u_{k-1} is a *type-1* node in tree T'.

Thus a node in the final tree T_G is a *type-4* node if and only if this node has been added to the growing tree as a part of a mid-tree path or a leaf path. □

Lemma 14. *Each node on a mid-tree path in tree T_G other than the first node and the last node has degree 2 in G.*

Proof. Let T be the tree during the computation of algorithm *Generate-Tree* when a mid-tree path $P = (u_1, u_2, \ldots, u_k)$ is selected in line 11. For each $i = 2, 3, \ldots, k-1$, node u_i is a non-expandable external node with two external neighbors u_{i-1} and u_{i+1}, so the definition of the expandable external nodes implies that u_i is not adjacent to any node in T and nodes u_{i-1} and u_{i+1} must be its only external neighbors. □

Lemma 15. *Let $(u_1, \ldots, u_{k-1})$ be a leaf path in tree T_G, and let u_k be the leaf in T_G attached to u_{k-1}. Then the following properties hold.*

1. *Each node $u_2, u_3, \ldots, u_{k-1}$ has degree 2 in G.*
2. *Node u_k has degree at most 2 in G.*
3. *If node u_k has degree 2 in G and the length of the leaf path is at least 2 ($k \geq 3$), then both neighbors of u_k in G have degree 2.*

Proof. Let T_1 be the tree constructed in the first part of the algorithm, and let $P = (u_1, \ldots, u_{k-1}, u_k)$, $k \geq 2$, be one of the paths considered in lines 30–31. Path $(u_1, u_2, \ldots, u_{k-1})$ is a leaf path in the final tree T_G. There is no expandable external node w.r.t. tree T_1, so for each $i = 2, 3, \ldots, k-1$, node u_i is a non-expandable external node with two external neighbors u_{i-1} and u_{i+1}. The definition of the expandable external nodes implies that node u_i is not adjacent to any node in T_1 and nodes u_{i-1} and u_{i+1} must be its only external neighbors. Thus the degree of nodes u_i in G is 2.

Node u_k is a non-expandable external node, so it may be adjacent to at most one external node other than u_{k-1}. However, node u_k cannot be adjacent to T_1 because if it were, then path P would have been split into two paths in lines 24–25. Thus the degree of node u_k in G is at most 2.

If node u_k has degree 2 in G, then path P has been obtained by splitting a path $(u_1, \ldots, u_k, u_{k+1}, \ldots, u_j)$ of external nodes in lines 24–25, where $2k \leq j \leq 2k+1$. If $k \geq 3$, and hence $j \geq k+2$, neither of nodes u_{k-1} and u_{k+1} is adjacent to tree T_1 and, as non-expandable external nodes, they may have only two external neighbors each. Thus both u_{k-1} and u_{k+1} have degree 2 in G. □

4.3 Approximation Ratio of the *STE* Algorithm

Lemma 10 implies that the cost of the exploration scheme computed by the *STE* algorithm for a graph G and a starting node s is

$$t_{ALG} \leq x_1 + 3x_3 + 4x_4 + 1, \qquad (2)$$

where x_t is the number of the *type-t* nodes in the tree T_G computed by algorithm *Generate-Tree*(G, s). The cost of the optimal exploration scheme is at least $n - 1 = x_1 + x_3 + x_4$, so any upper bound on x_4 in a form of a linear function of x_1 and x_3 would give immediately an upper bound on the approximation ratio of algorithm *STE* as a constant less than 4. However this simple approach cannot work by itself since the ratio $x_4/(x_1 + x_3)$ can be arbitrarily large not only for tree T_G, but for the best possible spanning tree as well. For example, if graph G is a path, then in its unique spanning tree all nodes except node s, its two neighbors and the end points of the path are *type-4* nodes.

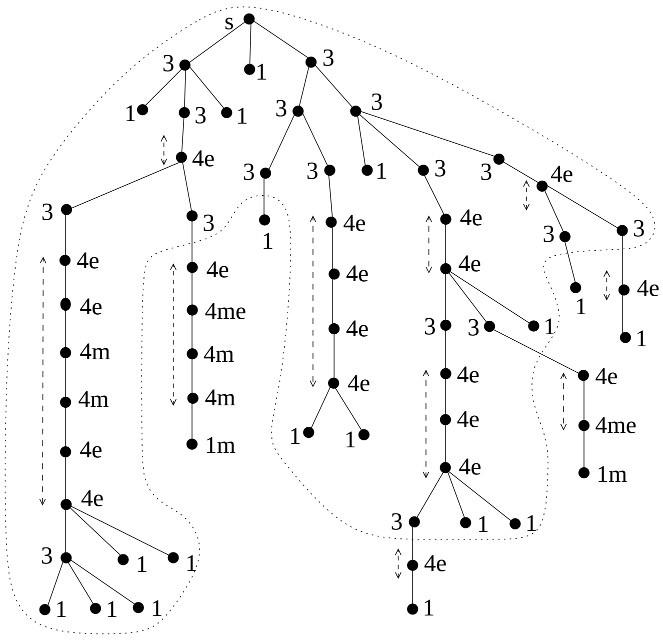

Fig. 7. An example of spanning tree produced by Algorithm *Generate-Tree*. Each node of the tree (excluding the root) is labeled with the corresponding type. The part of the tree produced during Part 1 of the algorithm is enclosed in the dotted curve. Arrows denote mid-tree paths and leaf paths.

Our analysis, which examines closer the *type-4* nodes in tree T_G, can be viewed as consisting of the following three steps. We first identify some nodes in graph G which "slow down" the optimal BHS in graph G so that its cost must be greater than the ideal $n-1$ (Lemma 16). We then show which *type-4* nodes in T_G must be among those "slowing down" nodes (Lemma 17). Finally we give a bound on the number of the other *type-4* nodes as a linear function of x_1 and x_3 (Lemma 18).

A node in graph G is a *type-d* node if its degree is at most 2 and the degrees of its neighbors are also at most 2.

Lemma 16. *The minimum cost of a BHS in graph G is*

$$t_{OPT} \geq n - 1 + \frac{1}{2}x_d \tag{3}$$

Proof. Informally, no BHS can explore *type-d* nodes at the average rate of one node per one step, requiring at least one additional step per two *type-d* nodes. Formally, consider any BHS and the case when there is no black hole. Each phase of the search when a *type-d* node v and another node u (which may be also a *type-d* node) are explored must consist of at least 3 steps. To see this, check that the distance from either v or u (or both) to the meeting point at the end of this phase must be at least 2. Thus

1. there are at least $(n-1)/2+\alpha$ phases in total, where $\alpha \geq 0$ is the number of phases when only one node is explored, and each phase consists of at least 2 steps;
2. there are at least $(x_d-\alpha)/2$ phases when a *type-d* node is explored together with another node, and each of these phases consists of at least 3 steps.

Hence the total number of steps is at least $n-1+2\alpha+(x_d-\alpha)/2 \geq n-1+x_d/2$. □

Lemma 13 says that the *type-4* nodes in tree T_G are the nodes on the mid-tree paths and the leaf paths. We further categorize these nodes in the following way. A *type-4e* node is a node which is one of the first two or the last two nodes of a mid-tree path or the first node of a leaf path. A *type-4me* node is the second node of a leaf path. All other nodes on the mid-tree paths and the leaf paths are *type-4m* nodes. We also introduce *type-1m* for the leaves attached to the leaf paths having length at least 2 (see the example in Figure 7). These definitions and Lemmas 14 and 15 immediately imply the following lemma.

Lemma 17. *Each* type-4m *or* type-1m *node in tree T_G is a* type-d *node in G.*

The next lemma gives bounds on the number of *type-4e* and *type-4me* nodes in tree T_G.

Lemma 18. *The number of* type-4e *nodes and the number of* type-4me *nodes in tree T_G satisfy the following relations.*

$$x_{4e} \leq 3x_1 + x_3 - 2, \tag{4}$$

$$x_{4me} = x_{1m}. \tag{5}$$

Proof. The fact that there are exactly as many *type-4me* nodes as *type-1m* nodes follows immediately from the definitions of these types. To show that Inequality (4) holds, denote by z' and z'' the number of the mid-tree paths and the number of the leaf paths in T_G, respectively. The definition of *type-4e* nodes imply that

$$x_{4e} \leq 4z' + z''. \tag{6}$$

The last node of a mid-tree path is a branching node in tree T_G (a node with at least two children) so $z' \leq x_1 - 1$ since T_G has at most $x_1 - 1$ branching nodes. We also have $z' + z'' \leq x_3 + 1$ since the parents of the first nodes of mid-tree paths and leaf paths must be distinct and each of them is either a *type-3* node or the root. Thus

$$4z' \leq 3(x_1 - 1) + x_3 + 1 - z'', \tag{7}$$

and Inequalities (6) and (7) give Inequality (4). □

We can now state our final theorem.

Theorem 2. *For any graph G and any starting node s, the ratio of the cost of a BHS based on the exploration scheme computed for G by the STE algorithm to the cost of an optimal BHS for G is at most $3\frac{3}{8}$.*

Proof. Starting from the bounds (2) and (3), we have

$$
\begin{aligned}
&\frac{27}{8}t_{OPT} - t_{ALG} \geq \\
&\quad \geq \frac{27}{8}(n-1+\frac{1}{2}x_d) - (x_1 + 3x_3 + 4x_4 + 1) && (8)\\
&\quad \geq \frac{27}{8}(x_1 + x_3 + x_{4e} + x_{4me} + \frac{3}{2}x_{4m} + \frac{1}{2}x_{1m}) && (9)\\
&\quad\quad -(x_1 + 3x_3 + 4x_{4e} + 4x_{4me} + 4x_{4m} + 1)\\
&\quad = \frac{19}{8}x_1 + \frac{3}{8}x_3 - \frac{5}{8}(x_{4e} + x_{4me}) + \frac{17}{16}x_{4m} + \frac{27}{16}x_{1m} - 1\\
&\quad \geq \frac{19}{8}x_1 + \frac{3}{8}x_3 - \frac{5}{8}(3x_1 + x_3 - 2 + x_{1m}) + \frac{17}{16}x_{4m} + \frac{27}{16}x_{1m} - 1 && (10)\\
&\quad = \frac{1}{4}(2x_1 - x_3) + \frac{17}{16}(x_{4m} + x_{1m}) + \frac{1}{4} \ \geq \ 0. && (11)
\end{aligned}
$$

Inequality (8) follows from (2) and (3), Inequality (9) follows from Lemma 17, and Inequality (10) follows from (4) and (5). Finally the inequality in line (11) holds because $x_3 \leq 2x_1 - 1$. To see that this is a valid bound on x_3, bound separately the number of *type-3* nodes which have only one descendant leaf and the number of the other *type-3* nodes. The number of *type-3* nodes which have only one descendant leaf is at most x_1, the number of leaves. Each *type-3* node which has at least 2 descendant leaves is either a branching node in T_G, or is the parent of the first node of a mid-tree path and the last node of this path is a branching node. Thus the number of *type-3* nodes which have at least 2 descendant leaves is at most the number of branching nodes in T_G, which is at most $x_1 - 1$. □

4.4 Additional Comments on Exploring a Graph via a Spanning Tree

One can also obtain a c-approximation algorithm for the BHS problem in graphs for a constant $c < 4$ using other ways of selecting a spanning tree than our algorithm *Generate-Tree*(G, s). In a preliminary version of this paper [12] we actually gave a different way, which was based on greedily selecting a maximal forest of bushy trees and then connecting the trees into one spanning tree. However, we could only show that that method led to an approximation ratio of $3\frac{1}{2}$.

Another possible good spanning tree of graph G is a spanning tree T which "locally" maximizes the number of leaves: no exchange of at most k tree edges for non-tree edges, for some constant k, can give a new spanning tree with more leaves than in T. Such a "locally maximized" spanning tree can be computed in polynomial time starting from any spanning tree. One can show that locally maximized spanning trees for $k = 2$, together with our *Search-Tree* algorithm, give an approximation algorithm for the BHS problem with an approximation ratio of $3\frac{7}{12}$.

We would like to mention that the straightforward algorithm for searching a tree outlined in the first paragraph of Section 4, together with good spanning-tree selection algorithms, can also give approximation algorithms with ratios less than 4, but greater than the approximation ratios which can be obtained using the *Search-Tree* algorithm. For example, the straightforward tree-searching algorithm gives approximation ratios of $3\frac{5}{8}$ and $3\frac{5}{6}$ for the BHS problem, if used together with the spanning trees computed by our *Generate-Tree* algorithm, and the locally maximized spanning trees, respectively.

Better approximation ratios can be obtained for some restricted graphs. For example, it is shown in [13] that any n-node graph with the minimum node degree at least 3 has a spanning tree with at least $n/4+2$ leaves, and a polynomial-time algorithm for computing such a spanning tree is given. This gives a c-approximation algorithm for the BHS problem for such graphs, where c is $3\frac{1}{2}$, if the straightforward tree-searching algorithm is used, or $3\frac{1}{4}$, if algorithm *Search-Tree* is used. It is also shown in [13] that for graphs with the minimum degree at least k one can compute in polynomial time spanning trees with at least $(1 - O((\log k)/k))n$ leaves. This gives a $(1 + O((\log k)/k))$-approximation algorithm for the BHS problem for this class of graphs.

5 Limitations of Exploration Schemes Based on Spanning Trees

The approximation algorithm for the BHS problem in arbitrary graphs which we presented in the previous section is based on the following two-part approach.

1. Find a suitable spanning tree T_G of the input graph G.
2. Using an algorithm for constructing exploration schemes for trees, construct an exploration scheme for T_G, and take it as an exploration scheme for G.

Even though this approach seems very natural (and it seems indeed difficult to analyze more general approaches), 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 nodes $v_1, v_2, \ldots, v_c$ and edges $(v_1, v_2), \ldots, (v_{c-1}, v_c), (v_c, v_1)$. 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 edge (x, y), with the following modification. Since the embedding of G_c has exactly two faces, the construction from Section 3 would add two shortcut edges bypassing 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 faces of the embedding of G_c. An example of graph G'_c, for $c = 7$, is shown in Figure 8. Graph G'_c has $4c+3$ nodes and by modifying appropriately the exploration scheme given 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 8. In the terminology and notation from Section 4.1, this tree has $x_3 = c - 1$ *type-3* nodes (the nodes $v_1, v_2, \ldots, v_{c-1}$) and $x_1 = 3c + 3$ *type-1* nodes. Lemma 9 implies that the cost of the exploration scheme computed for this tree by algorithm *Search-Tree* given in Section 4.1 is exactly $x_1 + 3x_3 = 6c$. We show below 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.

We use the following result from [1]. For a rooted tree T, the internal nodes of T other than the root are classified into two types: *type-β* nodes are the nodes with exactly one descendant, and *type-γ* nodes are the nodes with at least two descendants. The following lemma is Lemma 5.2 in [1] re-worded to fit our terminology.

Lemma 19. *[1] Let T be a rooted tree with $n + 1$ nodes, and let x_t denote the number of* type-t *nodes in T. The cost of any exploration scheme for T is at least $n + x_\beta + 2x_\gamma$.*

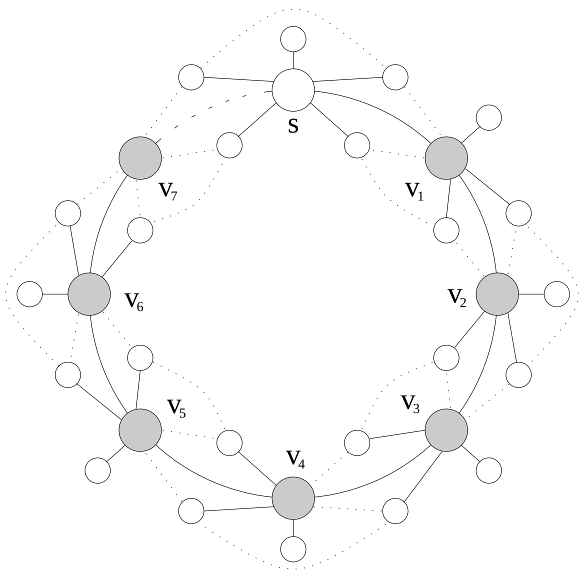

Fig. 8. Graph G'_7 and its "good" spanning tree (solid edges).

Lemma 20. *For any spanning tree T of G'_c rooted at s, $x_\beta + 2x_\gamma \geq 2c - 4$.*

Proof. All nodes in $V \setminus \{v_c\} = \{v_1, v_2, \ldots, v_{c-1}\}$ must be internal nodes in T since they have to be parents of their flag nodes. Let z be the number of *type-β* nodes in $V \setminus \{v_c\}$. The other $c - 1 - z$ nodes in $V \setminus \{v_c\}$ are of type γ. If two nodes v_i and v_j in the cycle G_c are such that $i+2 \leq j \leq c-1$ and the shortcut edges bypassing them are not in T, then at most one of them can be a *type-*β node. To see this observe that a path from s to a node v_k, $i < k < j$, must pass through one of the nodes v_i and v_j or through one of their shortcut edges. This means that if neither of the shortcut edges bypassing nodes v_i and v_j is in T, then either v_i or v_j is an ancestor of v_k and therefore cannot be of type β. Thus at least $z-2$ shortcut edges belong to T. Each *type-*β node in $V \setminus \{v_c\}$ contributes 1 to $x_\beta + 2x_\gamma$, each *type-*γ node in $V \setminus \{v_c\}$ contributes 2, and each shortcut edge belonging to T contributes at least 1 (at least one node of such an edge is an internal node in T). Therefore we have

$$x_\beta + 2x_\gamma \geq z + 2(c-1-z) + z - 2 = 2c - 4. \qquad \square$$

Lemmas 19 and 20 imply that the cost of any exploration scheme for any spanning tree of G'_c is at least $6c-2$.

6 Conclusion

We proved that designing an optimal BHS for an arbitrary planar graph is NP-hard, thus solving an open problem stated in [1]. We also gave a polynomial time $3\frac{3}{8}$-approximation algorithm for the BHS problem, showing the first non-trivial upper bound on the approximation ratio for this problem. Finally, we showed that any exploration scheme that visits the given input graph via some spanning tree, as our algorithm does, cannot have an approximation ratio better than 3/2.

We believe that one could show a better upper bound for the approximation ratio of our algorithm than $3\frac{3}{8}$ by further refining the analysis, but we do not expect a bound anywhere near the currently best lower bound 3/2. Similarly, one could probably somewhat improve the bounds on the approximation ratios of the methods mentioned in Section 4.4, but we believe that these bounds will remain higher than the bound for our main algorithm. It seems that to obtain a more substantial improvement of the approximation ratio one would need to abandon the spanning-tree approach, or at least to augment it with some considerably new ideas.

For other complexity issues regarding the black hole search with two agents, we particularly would like to see answers to the following two questions. Is there a constant $c > 1$ such that the approximate BHS problem with ratio c is NP-hard? Is the BHS problem for arbitrary trees NP-hard? We expect the positive answer to the first question and the negative answer to the second one.

It would be also interesting to see non-trivial results regarding the complexity of computing fast black hole search schemes for many agents and possibly many black holes. If there are $k+1$ agents, where k is a parameter (not a constant) and at most k black holes, then it is not even clear how one should formalize the problem. If there are more than two agents and possibly more than one black hole, then the "oblivious" approach of giving each agent one predetermined sequence of nodes to visit is no longer sufficient to cover all exploration algorithms.

Acknowledgements

We would like to thank the two 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 *Proc. 8th Int. Conf. on Principles of Distributed Systems (OPODIS 2004)*, pages 34–35, 2004. Also: Springer LNCS vol. 3544, pages 67-80.
2. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 72:1–14, 2006.
3. S. Dobrev, P. Flocchini, R. Kralovic, 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. Multiple agents rendezvous in a ring in spite of a black hole. In *Proc. 7th Int. Conf. on Principles of Distributed Systems (OPODIS 2003)*, Springer LNCS vol. 3144, pages 34–46, 2003.
5. 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.
6. 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.
7. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., 1979.
8. 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.
9. F. Hohl. Time limited black box security: Protecting mobile agents from malicious hosts. In *Proc. Conf. on Mobile Agent Security*, Springer LNCS vol. 1419, pages 92–113, 1998.
10. 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.
11. R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary graphs. Technical Report 5659, INRIA, August 2005.
12. 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.
13. D.J. Kleitman and D.B. West. Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics*, 4(1):99–106, 1991.
14. 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.

15. 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.
16. K. Schelderup and J. Ines. Mobile agent security – issues and directions. In *Proc. 6th Int. Conf. on Intelligence and Services in Networks*, Springer LNCS vol. 1597, pages 155–167, 1999.
17. J. Vitek and G. Castagna. Mobile computations and hostile hosts. In D. Tsichritzis, editor, *Mobile Objects*, pages 241–261. University of Geneva, 1999.