

Ping Pong in Dangerous Graphs: Optimal Black Hole Search with Pure Tokens

Paola Flocchini ^{*} David Ilcinkas [†] Nicola Santoro [‡]

Abstract

We prove that, for the black hole search problem, the pure token model is computationally as powerful as the whiteboard model; furthermore the complexity is exactly the same. More precisely, we prove that a team of *two* asynchronous agents, each endowed with a single identical pebble (that can be placed only on nodes, and with no more than one pebble per node) can locate the black hole in an arbitrary network of known topology; this can be done with $\Theta(n \log n)$ moves, where n is the number of nodes, even when the links are not FIFO.

Keywords: distributed computing, graph exploration, mobile agents, autonomous robots, dangerous graphs.

1 Introduction

1.1 The Framework

Black Hole Search (BHS) is the distributed problem in a networked system (modeled as a simple edge-labelled graph G) of determining the location of a *black hole* (BH): a site where any incoming agent is destroyed without leaving any detectable trace. The problem has to be solved by a team of identical system agents injected into G from a safe site (the homebase). The team operates in presence of an adversary that chooses e.g., the edge labels, the location of the black hole, the delays, etc. The problem is solved if at least one agent survives and all surviving agents know the location of the black hole (e.g., see [15]).

The practical interest of BHS derives from the fact that a black hole can model several types of faults, both hardware and software, and security threats arising in networked systems supporting code mobility. For example, the crash failure of a site in an asynchronous network turns such a site into a black hole; similarly, the presence at a site of a malicious process (e.g.,

^{*}SITE, University of Ottawa, Ottawa, ON K1N 6N5, Canada. E-mail: flocchin@site.uottawa.ca

[†]CNRS, LaBRI, Université Bordeaux I, France. E-mail: david.ilcinkas@labri.fr.

[‡]School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada.

E-mail: santoro@scs.carleton.ca

a virus) that thrashes any incoming message (e.g., by classifying it as spam) also renders that site a black hole. Clearly, in presence of such a harmful host, the first step must be to determine and report its location.

From a theoretical point of view, the natural interest in the computational and complexity aspects of this distributed problem is amplified by the fact that it opens a new dimension in the classical *graph exploration* problem. In fact, the black hole can be located only after all the nodes of the network but one have been visited and are found to be safe; in this exploration process some agents may disappear in the black hole. In other words, while the existing wide body of literature on *graph exploration* (e.g., see [1, 2, 8, 9, 16, 17]) assumes that the graph is *safe*, BHS opens the research problems of the exploration of *dangerous graphs*.

Indeed BHS has been studied in several settings, under a variety of assumptions on the power of the adversary and on the capabilities of the agents; e.g., on the level of synchronization of the agents; on whether or not the links are FIFO; on the type of mechanisms available for inter agent communication and coordination; on whether or not the agents have a map of the graph. In these investigations, the research concern has been to determine under what conditions and at what cost mobile agents can successfully accomplish this task. The main complexity measures are the size of the team (i.e., the number of agents employed) and the number of moves performed by the agents; sometimes also time complexity is considered.

In this paper we are interested in the weakest settings that still make the problem solvable. Thus we will make no assumptions on timing or delays, and focus on the *asynchronous* setting. Indeed, while the research has also focused on the *synchronous* case [5, 6, 7, 18, 19] where all agents are synchronized and delays are unitary, the main body of the investigations has concentrated on the asynchronous one (e.g., [4, 10, 11, 12, 13]).

1.2 The Quest and its Difficulties

In the asynchronous setting, the majority of the investigations operate in the *whiteboard* model: every node provides a shared space for the arriving agents to read and write (in fair mutual exclusion). The whiteboard model is very powerful: it endows the agents not only with direct and explicit communication capabilities, but also with the means to overcome severe network limitations; in particular, it allows the software designer to assume FIFO links (even when not supported by the system). Additionally, whiteboards allow to break symmetry among identical agents. Indeed, whiteboards (and even stronger inter-agent coordination mechanisms) are supported by most existing mobile agent platforms [3]. The theoretical quest, on the contrary, has been for the weakest interaction mechanism allowing the problem to be solved.

A weaker and less demanding interaction mechanism is the one assumed by the *token* model, used in the early investigations on (safe) graph exploration; it is provided by identical *pebbles* (that can be placed on nodes, picked up and carried by the agents) without any other form of marking or communication (e.g., [2]).

The research quest is to determine if pebbles are computationally as powerful as whiteboards with regards to BHS. The importance of this quest goes beyond the specific problem,

as it would shed some light on the relative computational power of these two interaction mechanisms.

Two results have been established so far in this quest. In [10] it has been shown that $\Delta + 1$ agents* without a map (the minimum team size under these conditions), each endowed with an identical pebble, can locate the black hole with a (very high but) polynomial number of moves. In [13] it has been shown that two agents with a map (the minimum team size under these conditions), each endowed with a constant number of pebbles, can locate the black hole in a ring network with $\Theta(n \log n)$ moves, where n denotes the number of nodes in the network.

Although they indicate that BHS can be solved using pebbles instead of whiteboards, these results do not prove yet the computational equivalence for BHS of these two inter-agent coordination mechanisms. There are two main reasons for this. The first main reason is that both results assume *FIFO links*; note that the whiteboard model allows to work assuming FIFO links, but does not require them. Hence, the class of networks for which the results of [10, 13] apply is smaller than that covered with whiteboards; also such an assumption is a powerful computational help to any solution protocol. The second and equally important reason is that these results are *not* established within the “pure” token model used in the traditional exploration problem. In fact, in [10, 13] the agents are allowed to place pebbles not only on nodes but also on links (e.g., to indicate on which link it is departing); this gives immediately to a single token the computational power of $O(\log \Delta)$ bits of information. In [13], where the network considered is only a ring, each agent has available several tokens, and multiple tokens can be placed at the exact same place (node or link) to store more than one bit of information.

1.3 Our Results

In this paper, we provide the first proof that indeed the pure token model is computationally as powerful as the whiteboard model for BHS.

The context we examine is the one of agents with a map in an arbitrary graph. For this context we prove that: A team of *two* asynchronous agents, each endowed with a single identical pebble (that can be placed only on nodes, and at no more than one pebble per node) and a map of the graph can locate the black hole with $\Theta(n \log n)$ moves, even if the links are not FIFO.

In other words, for networks of known topology, using pure tokens it is possible to obtain exactly the same optimal bounds for team size and number of moves as using whiteboards.

Note that our result implies as a corollary an optimal solution for the whiteboard model using only a single bit of shared memory per node; the existing solution [11] requires a whiteboard of $O(\log n)$ bits at each node.

Our results are obtained using a new and (surprisingly) simple technique called *ping pong*. In its bare form, this technique solves the problem but with $O(n^2)$ moves. To obtain the optimal bound, the technique is enhanced by integrating it with additional mechanisms,

* Δ denotes the maximum node degree in G

exploiting two ideas developed in previous investigations: “split work” [12], and “distance counting” [13]. The mechanisms that we have developed use a variety of novel not-trivial techniques, and are the first to overcome the severe limitation imposed by the lack of the FIFO assumption (available instead in all previous investigations with whiteboards or tokens).

The paper is organized as follows. We first present our techniques, prove their properties and analyze their complexity in the case of ring networks (Section 3). Then, in Section 4, we show how to modify and enhance those techniques so to obtain the same bounds also in the case of arbitrary graphs.

2 Terminology and Definitions

Let $G = (V, E)$ be a simple biconnected[†] graph with $n = |V|$ nodes. At each node x , there is a distinct label from a totally ordered set associated to each of its incident links. We shall denote by (G, λ) the resulting edge-labelled graph.

Operating in (G, λ) is a team of identical autonomous mobile agents (or robots). All agents enter the system from the same node, called homebase. The agents have computing capabilities, computational storage (polynomially bounded by the size of the graph), and a map of (G, λ) with the indication of the homebase; they can move from node to neighbouring node, and obey the same set of behavioral rules (the *algorithm*). Every agent has a *pebble*; all pebbles are identical. A pebble can be carried, put down at a node if no other pebble is already there, and picked up from a node by an agent without pebbles.

When an agent enters a node, it can see if there is a pebble dropped there; it might be however unable to see other agents there or to determine whether they are carrying a pebble with them.

The system is *asynchronous* in the sense that (i) each agent can enter the system at an arbitrary time; (ii) traveling to a node other than the black hole takes a finite but otherwise unpredictable amount of time; and (iii) an agent might be idle at a node for a finite but unpredictable amount of time. The basic computational step of an agent (executed either when the agent arrives to a node, or upon wake-up) is to look for the presence of a pebble, drop or pick up the pebble if wanted, and leave the node through some chosen port (or terminate). The whole computational step is performed in local mutual exclusion as an atomic action, i.e. as if it took no time to execute it. Links are *not* FIFO: two agents moving on the same link in the same direction at the same time might arrive at destination in an arbitrary order.

To simplify the model, we can assume without loss of generality that the transition between two states of the agent at a node plus the corresponding move are instantaneous. In other words, the waiting due to asynchrony only occurs after the move of the agent. Furthermore we can assume that also the actions of agents at different nodes occur at different instants.

[†]Note that biconnectivity is necessary for BHS to be solvable [11].

A *black hole* is a node that destroys any incoming agent; no observable trace of such a destruction will be evident to the other agents. The location of the black hole is unknown to the agents. The *Black Hole Search* problem is to find the location of the black hole. More precisely, the problem is solved if at least one agent survives, and all surviving agents know the location of the black hole.

The two measures of complexity of a solution protocol are the number of agents used to locate the black hole and the total number of moves performed by the agents.

3 Black hole search in rings

3.1 Preliminaries

Without loss of generality, we can assume that the clockwise direction is the same for both agents: for example, the direction implied by the link with the smallest label at the homebase. In the following, going right (resp. left) means going in the clockwise (resp. counterclockwise) direction. An agent exploring to the right (resp. left) is said to be a *right* (resp. *left*) agent. Using this definition, an agent *changes role* if it was a left agent and becomes a right agent or vice versa. For $i \geq 0$, the node at distance i to the right, resp. to the left, of the home base will be called node i , resp. node $-i$. Hence node i and $i - n$ represent the same node, for $0 \leq i \leq n$.

In the algorithm the agents obey the two following metarules:

1. An agent always ensures that a pebble is lying at u before traversing an unknown edge $\{u, v\}$ from u to v (i.e. an edge that it does not know to be safe).
2. An agent never traverses an unknown edge $\{u, v\}$ from u to v if a pebble lies at u and the pebble was not dropped there by this agent.

These metarules imply that the two agents never enter the black hole from the same edge. Moreover, each agent keeps track of its progress by storing the number of the most-right, resp. most-left, node in a variable `Last_Right`, resp. `Last_Left`, used to detect termination: when only one node remains unexplored, this node is the black hole and the agent can stop.

A (right) agent is said to traverse an edge $\{u, v\}$ from u to v using *cautious walk* if it has one pebble, it drops it at u , traverses the edge (in state `Explore-Right`), comes back to u (in state `Pick-Up-Right`), retrieves the pebble and goes again to v (in state `Ping-Right`). A (left) agent is said to traverse an edge $\{u, v\}$ from u to v using *double cautious walk* if it has one pebble and the other is at u , it goes to v (in state `Explore-Left`) carrying one pebble, the other pebble staying at u , drops the pebble at node v , comes back to u (in state `Pick-Up-Left`), retrieves the other pebble and goes again to v (in state `Ping-Left`). We will see later that double cautious walk is employed only by left agents. Note that these two cautious explorations obey the first metarule.

3.2 The Algorithm

Our algorithm is based on a novel coordination and interaction technique for agents using simple tokens, *Ping-Pong*. The idea at the basis of this technique is the following: one agent explores the “right” side and one the “left” side (the side assigned to an agent changes dynamically, due to the non-FIFO nature of the links). However, only one agent at a time is allowed to explore; the agent willing to do so must first “steal” the pebble of the other, and then can proceed to explore its allowed side. When an agent discovers that its pebble has been stolen, it goes to find it and steal the other pebble as well. This generate a “ping-pong” movements of the agents on the ring. The actual Ping-Pong technique based on this idea must however take into account the non-FIFO nature of the links, which creates a large variety of additional situations and scenarios (e.g., an agent moving to steal the pebble of the other, might “jump over” the other agent).

Algorithm `EnhancedPingPong` is divided in two phases, each one further divided into stages. The first phase is the Ping-Pong technique. The second phase, whose function is to ensure that the costs are kept low, in some cases may not be executed at all. Inside a phase, a *stage* is a maximal period during which no agent changes role.

In the first phase, exploration to the right is always done using cautious walk, while exploration to the left is always done using double cautious walk (i.e., after stealing a pebble). Note that, since an agent exploring to the right uses one pebble and an agent exploring to the left uses two pebbles, the agents cannot make progress simultaneously in two different directions because there are only two pebbles in total. This also implies that while an agent is exploring new nodes it knows all the nodes that have already been explored, as well as the position of the only unexplored node where the other agent possibly died. This prevents the agents from exploring the same node and thus from dying in the black hole from two different directions.

Phase 1. Initially both agents explore to the right. Since links are not FIFO, an agent may pass the other and take the lead without any of the two noticing it. Nevertheless, it eventually happens that one agent L finds the pebble of the other agent R , say at node p (at the latest it happens when one agent locates or dies in the black hole). When this happens L drops its pebble at node $p - 1$ (if its pebble is not already there) and steals R 's pebble. Having control on the two pebbles, L starts to explore left using double cautious walk. The stage has now an even number. When/if R comes back to p to retrieve its pebble, it does not find it. It then goes left in state `Pong-Right` until it finds a pebble. Agent R does eventually find a pebble because at the beginning of the stage there is a pebble at its left (at node $p - 1$), and Agent L never removes a pebble before putting the other pebble further to the left. At this point R retrieves the pebble and goes right again in state `Ping-Right` and explores to the right. When/if L realizes that one of its pebble has been stolen, it changes role (and the stage changes) and explores to the right using its remaining pebble. At this point, both agents explore to the right. Again, one agent will find and steal the pebble of the other. To ensure progress in exploration, a right agent puts down its pebble only when it reaches the last visited node to the right it knows (using its variable `Last_Right`). Consequently the stealing at the end of an odd stage always occurs at least one node further to the right from

two stages before. Hence the algorithm of Phase 1 is in fact correct by itself but the number of moves can be $\Theta(n^2)$ in the worst case (one explored node every $O(n)$ moves). To decrease the worst case number of moves to $O(n \log n)$, the agents switch to Phase 2 as soon as at least two nodes have been explored to the right.

Phase 2. Phase 2 uses the *halving* technique, based on an idea of [12], but highly complicated by the absence of whiteboards and by the lack of FIFO. The idea is to regularly divide the workload (the unexplored part) in two. One agent has the left half to explore (using variable `Goal_Left`), while the second agent explores the right half (using variable `Goal_Right`). These explorations are performed concurrently by using (simple) cautious walk (for a right agent, in states `Halving-Explore-Right`, `Halving-Pick-Up-Right` and `Halving-Ping-Right`). After finite time, exactly one agent finishes its part and joins the other in exploring the other part, changing role and thus changing the stage number. At some point, one agent A will see the other agent's pebble. A steals the pebble and moves it by one position to indicate a change of stage to the second agent B . It then computes the new workload, divide it into two parts (using the function `Update_Goal_Left` or `Update_Goal_Right`), and goes and explores its newly assigned part, changing role again by switching to state `Halving-From-Left-To-Right` or `Halving-From-Right-To-Left`. This can happen several times (if B remains blocked by the asynchronous adversary or if it is dead in the black hole). When/if agent B comes back to retrieve its pebble, it does not find it. It further goes back to retrieve its pebble in state `Halving-Pong-Right` (if it is a right agent). The number of moves it has to perform to find the pebble indicates how many halvings (pair of stages) it misses. Knowing that, it can compute what is the current unexplored part and what is its current workload. It then starts to explore its part. Since there are at most $O(\log n)$ stages of $O(n)$ moves each, this leads to a total number of moves of $O(n \log n)$.

The algorithm starts with a few stages of Phase 1 because Phase 2 needs some safe nodes to put the pebble that is used as a message to indicate the current partition of the workload.

Several other technical details and precautions have to be taken because of asynchrony and lack of FIFO. The code describing all the details of the state transitions can be found below.

Procedure CHECK_TERMINATION

```

1: if Last_Right - Last_Left = n - 2 then
2:   Black_Hole := Last_Right + 1
3:   Terminate
4: end if

```

Functions `Update_Goal_Right` and `Update_Goal_Left`:

```

Update_Goal_Right(p, q, 0) = p
Update_Goal_Right(p, q, k) =  $\lfloor \frac{p'+q+n}{2} \rfloor$  where  $p' = \text{Update\_Goal\_Right}(p, q, k - 1)$ 

Update_Goal_Left(p, q, 0) = q
Update_Goal_Left(p, q, k) =  $\lfloor \frac{p+q'-n}{2} \rfloor$  where  $q' = \text{Update\_Goal\_Left}(p, q, k - 1)$ 

```

for $q \leq 0 \leq p$ and $k \geq 1$.

Initial state of Algorithm EnhancedPingPong:

State: Ping-Right

Variables: Last_Right = Last_Left = Counter = 0

Phase-1 states of Algorithm EnhancedPingPong

Ping-Right

Input invariant: The robot has a pebble with him.

```
1: CHECK_TERMINATION
2: if current  $\neq$  Last_Right then // Do not use cautious walk if not necessary
3:   Move Right
4:   (Stay Ping-Right)
5: else
6:   if empty then // Everything seems normal
7:     Put_Down_Pebble
8:     Move Right
9:     Become Explore-Right
10:  else // An other right robot is preceding me
11:    Move Left
12:    Become Put-Pebble-Right
13:  end if
14: end if
```

Explore-Right

Input invariant: The robot is empty-handed.

```
1: Last_Right := max{current, Last_Right} // Everything seems normal
2: if empty then
3:   Move Left
4:   Become Pick-Up-Right
5: else
6:   CHECK_TERMINATION
7:   Pick_Up_Pebble
8:   if Last_Right < 2 then // We are still in Phase 1
9:     Move Left
10:    Become Ping-Left
11:  else // Start Phase 2
12:    Goal_Left := Update_Goal_Left(Last_Right, Last_Left, 1)
13:    Goal_Right := Goal_Left + n - 1
14:    Move Left
15:    Become Halving-From-Right-To-Left
16:  end if
17: end if
```

Pick-Up-Right

Input invariant: The robot is empty-handed.


```

1: if not empty then                                     // Everything seems normal
2:   Pick_Up_Pebble
3:   Move Right
4:   Become Ping-Right
5: else
6:   if Last_Right < 3 then                                 // We are still in Phase 1
7:     Move Left
8:     Become Pong-Right
9:   else                                                   // Start Phase 2
10:    Last_Right := Last_Right - 1
11:    Goal_Left := Last_Left
12:    Counter := 0
13:    Move Left
14:    Become Halving-Pong-Right
15:  end if
16: end if

```

Put-Pebble-Right

Input invariant: The robot has a pebble with him.

```

1: Put_Down_Pebble
2: Move Right
3: Become Explore-Right

```

Pong-Right

Input invariant: The robot is empty-handed.

```

1: if empty then                                           // Pebble not yet found
2:   Move Left
3:   (Stay Pong-Right)
4: else                                                     // Pebble found
5:   Last_Left := min{current, Last_Left}
6:   CHECK_TERMINATION
7:   Pick_Up_Pebble
8:   Move Right
9:   Become Ping-Right
10: end if

```

Ping-Left

Input invariant: The robot has a pebble with him.

```

1: Last_Left := min{current, Last_Left}
2: CHECK_TERMINATION
3: if not empty then                                       // Everything seems normal
4:   Move Left
5:   Become Explore-Left
6: else                                                     // One of my pebbles has been stolen
7:   Move Right
8:   Become Ping-Right
9: end if

```

Explore-Left

Input invariant: The robot has a pebble with him.

- 1: Put_Down_Pebble
- 2: Move Right
- 3: Become Pick-Up-Left

Pick-Up-Left

Input invariant: The robot is empty-handed.

- 1: **if** not empty **then** *// Everything seems normal*
- 2: Pick_Up_Pebble
- 3: Move Left
- 4: Become Ping-Left
- 5: **else** *// One of my pebbles has been stolen*
- 6: Move Left
- 7: Become Pick-Up-Right
- 8: **end if**

Phase-2 states of Algorithm EnhancedPingPong

In the following we describe only the states for a right agent. The states for a left agent are symmetric.

Halving-From-Left-To-Right

Input invariant: The robot has a pebble with him.

- 1: Move Right
- 2: Become Halving-Ping-Right

Halving-Ping-Right

Input invariant: The robot has a pebble with him.

- 1: **if** not empty **then** *// An other right robot is preceding me*
- 2: Move Left
- 3: Become Halving-Put-Pebble-Right
- 4: **else**
- 5: **if** current \neq Last_Right **then** *// Do not use cautious walk if not necessary*
- 6: Move Right
- 7: (Stay Halving-Ping-Right)
- 8: **else** *// Everything seems normal*
- 9: Put_Down_Pebble
- 10: Move Right
- 11: Become Halving-Explore-Right
- 12: **end if**
- 13: **end if**

Halving-Explore-Right

Input invariant: The robot is empty-handed.

```

1: Last_Right := max{current, Last_Right}
2: CHECK_TERMINATION
3: if empty then // Everything seems normal
4:   Move Left
5:   Become Halving-Pick-Up-Right
6: else // An other right robot is preceding me
7:   Last_Left := Goal_Left
8:   Goal_Left := Update_Goal_Left(Last_Right, Last_Left, 1)
9:   Goal_Right := Goal_Left + n - 1
10:  Pick_Up_Pebble
11:  Move Left
12:  Become Halving-From-Right-To-Left
13: end if

```

Halving-Pick-Up-Right

Input invariant: The robot is empty-handed.

```

1: if not empty then // Everything seems normal
2:   Pick_Up_Pebble
3:   if Last_Right  $\neq$  Goal_Right then // The current objective is fulfilled
4:     Move Right
5:     Become Halving-Ping-Right
6:   else // The current objective is not fulfilled yet
7:     Move Left
8:     Become Halving-Ping-Left
9:   end if
10: else // My pebble has been stolen
11:   Last_Right := Last_Right - 1
12:   Last_Left := Goal_Left
13:   Counter := 0
14:   Move Left
15:   Become Halving-Pong-Right
16: end if

```

Halving-Put-Pebble-Right

Input invariant: The robot has a pebble with him.

```

1: Put_Down_Pebble
2: Move Right
3: Become Halving-Explore-Right

```

Halving-Pong-Right

Input invariant: The robot is empty-handed.

```

1: Counter++
2: Last_Left := Goal_Left
3: Goal_Left := Update_Goal_Left(Last_Right, Last_Left, 1)
4: Goal_Right := Goal_Left + n - 1
5: if empty then // Pebble not yet found
6:   Move Left
7:   (Stay Halving-Pong-Right)

```

```

8: else // Pebble found
9:   Counter := 0
10:  Pick_Up_Pebble
11:  Move Right
12:  Become Halving-Ping-Right
13: end if

```

3.3 Correctness and complexity

As explained before the algorithm consists of up to two *phases*. The first one corresponds to the case where both agents are in one of the eight states `Ping-Right`, `Ping-Left`, `Explore-Right`, `Explore-Left`, `Pick-Up-Right`, `Pick-Up-Left`, `Pong-Right`, `Put-Pebble-Right`. If this is not the case, we say that the algorithm is in its second phase. (Note that this phase may not exist in all possible executions.) An agent is said to be a *right*, resp. *left*, agent if its state ends with `-Right`, resp. `-Left`. Using this definition, an agent *changes role* if it was a left agent and becomes a right agent or vice versa. Finally, inside a phase, a *stage* is a maximal period during which no agent changes role.

For the purpose of the proofs of the main theorems, we will use the three following properties.

Property $\mathcal{P}(p)$, with $p \in \{0, 1\}$: There is a left agent L and a right agent R . The agent L is waiting at node $p - 1$, where one pebble is located. Agent L is carrying the other pebble and is in state `Ping-Left`. Moreover, its variable `Last_Right` has value p . Agent R , empty-handed, is in one of the following situations:

- it is dead in the black hole located at node $p + 1$;
- it is at node $p + 1$ in state `Explore-Right` and its variable `Last_Right` has value p ;
- it is already back from node $p + 1$ at node p in state `Pick-Up-Right` and its variable `Last_Right` has value $p + 1$.

Moreover, the termination condition of agent L is not satisfied, and in the last two cases, the value `Last_Left` is the same for each agent.

Property $\mathcal{P}'_L(p, q)$, with $p \geq 2$, $q \leq 0$ and $p - q < n - 2$: There is a left agent L and a right agent R . There exists some $k \geq 0$, with $p - k - 1 > q$, such that L is waiting at node $p - k - 1$ where one pebble is located. Agent L is carrying the other pebble and is in state `Halving-From-Right-To-Left`. Moreover, its variable `Last_Right`, resp. `Last_Left`, has value p , resp. q . Its variable `Goal_Left` has value `Update_Goal_Left(p, q, 1)`. (Its variable `Goal_Right` has value `Goal_Left + n - 1`.) Agent R , empty-handed, is in one of the following situations:

- it is dead in the black hole located at node $p+1$;
- it is waiting at $p+1$ in state `Explore-Right` and its variable `Last_Right` has value p ;
- it is already back from node $p + 1$ at node p in state `Pick-Up-Right` and its variable `Last_Right` has value $p + 1$;
- it is waiting at node $p + 1$ in state `Halving-Explore-Right` and its variable `Last_Right` has value p ;

- it is already back from node $p + 1$ at node p in state `Halving-Pick-Up-Right` and its variable `Last_Right` has value $p + 1$;
- it is waiting at node $p - i$, $1 \leq i \leq k$, in state `Halving-Pong-Right`, its variable `Last_Right` has value $p + 1$ and its variable `Counter` has value $i - 1$.

Moreover, in the second and third cases, the value `Goal_Left` of Agent L is equal to `Update_Goal_Left` ($p, q', k + 1$), where q' is the value `Last_Left` of Agent R . In the last three cases, the value `Goal_Left` of Agent L is equal to `Update_Goal_Left`($p, q', k + 1 - \text{Counter}$), where q' equals `Goal_Left` of Agent R .

Property $\mathcal{P}'_R(p, q)$, with $p \geq 2$, $q \leq 0$ and $p - q < n - 2$: There is a left agent L and a right agent R . There exists some $k \geq 0$, with $q + k + 1 < p$, such that R is waiting at node $q + k + 1$ where one pebble is located. Agent R is carrying the other pebble and is in state `Halving-From-Left-To-Right`. Moreover, its variable `Last_Left`, resp. `Last_Right`, has value q , resp. p . Its variable `Goal_Right` has value `Update_Goal_Right`($p, q, 1$). (Its variable `Goal_Left` has value `Goal_Right` $- n + 1$.) Agent L , empty-handed, is in one of the following situations:

- it is dead in the black hole located at node $q - 1$;
- it is waiting at $q - 1$ in state `Halving-Explore-Left` and its variable `Last_Left` has value q ;
- it is already back from node $q - 1$ at node q in state `Halving-Pick-Up-Left` and its variable `Last_Left` has value $q - 1$;
- it is waiting at node $q + i$, for some $1 \leq i \leq k$ in state `Halving-Pong-Left`, its variable `Last_Left` has value $q - 1$ and its variable `Counter` has value $i - 1$.

Moreover, in the last three cases, the value `Goal_Right` of Agent R is equal to `Update_Goal_Right` ($p', q, k + 1 - \text{Counter}$), where p' is `Goal_Right` of Agent L .

Lemma 1 *Consider a n -node ring containing a homebase and a black hole, and two agents running Algorithm `EnhancedPingPong` from the homebase. After finite time, one of the following situations occurs:*

- Stage 2 of Phase 1 begins and Property $\mathcal{P}(p)$ holds for some $p \in \{0, 1\}$;
- Phase 2 begins and Property $\mathcal{P}'_L(p, 0)$ holds for some integer p such that $2 \leq p \leq n - 2$;
- all agents of the non-empty set of surviving agents have terminated and located the black hole.

Moreover, at that time, each edge has been traversed at most a constant number of times since the beginning of the algorithm.

Proof: Let R_1 be the first agent to act and let R_2 be the other agent.

First assume that R_2 starts immediately after R_1 and finds R_1 's pebble at node 0 at the beginning. Thus R_2 moves left to node -1 in state `Put-Pebble-Right`. If node -1 is the black hole, then R_2 dies and R_1 eventually visits all nodes from 0 to $n - 2$ and terminates by correctly locating the black hole at node $n - 1$ (or equivalently node -1). If node -1 is not the black hole, then R_2 drops its pebble at node -1 and comes back to node 0 in state `Explore-Right`. If R_1 's pebble is still there, then R_2 steals it and goes to the left in state `Ping-Left`, and the stage changes. Moreover property $P(0)$ is satisfied, which proves the

lemma in this case. If R_1 's pebble is not there anymore, then the situation is exactly the same as if R_2 would have not find R_1 's pebble at the beginning because in this case R_2 eventually goes back to node -1 to retrieve its pebble and comes back to node 0, holding the pebble, in state `Ping-Right`, with `Last_Left` = 0.

We now assume that R_2 does not find R_1 's pebble at node 0 at the beginning. However, at some point, one agent will eventually find the pebble of the other agent. More precisely, one agent R_L in state `Explore-Right` eventually finds the pebble of the other agent R_R and executes its transition function while R_R 's pebble still lies at the current node p (the other pebble being at node $p-1$). This is always true because such a situation occurs, at the latest, when one of the agent locates the black hole or dies in it. Indeed, in both cases, its pebble stays forever at the node u preceding the black hole, giving the other agent enough time to drop its own pebble at the previous node on the left thanks to the state `Put-Pebble-Right` (if this latter pebble is not already there), to go back to u and to execute its transition function. Let p be the number of the node where R_R 's pebble is found by R_L in state `Explore-Right`. Note that $p \geq 1$.

If the termination condition of R_L is satisfied after the update of the variable `Last_Right`, then R_L terminates by locating the black hole at its actual position $p+1$ (the only node not explored by R_L). Concerning R_R , it has already terminated by locating the black hole at $p+1$ (both agents have the same `Last_Left` value). Otherwise (the termination condition of R_L is not satisfied) R_L changes role. At this time, if $p < 2$, then Property $\mathcal{P}(1)$ is satisfied, and if $p \geq 2$, then Property $\mathcal{P}'_L(p, 0)$ is satisfied.

Since the beginning of the algorithm, each edge has been traversed at most a constant number of times. Indeed, the edge between the nodes -1 and 0 is traversed at most twice and any other edge $\{x, x+1\}$, for $x \geq 0$, is traversed at most 7 times by each agent in the worst case: the first time from left to right (i.e. from x to $x+1$) in state `Explore-Right`, then twice for retrieving the pebble (if no pebble is found at $x+1$), then twice to put the pebble back (if a pebble appeared meanwhile at $x+1$), and finally twice to retrieve again the pebble (if the pebble disappeared from $x+1$). \square

Lemma 2 *Consider a n -node ring containing a homebase and a black hole, and two agents running Algorithm **EnhancedPingPong** from the homebase. Assume that at some time t a Phase-1 stage of even number i begins and that Property $\mathcal{P}(p)$ holds for some $p \in \{0, 1\}$. Then at some time $t' > t$ one of the following situations occurs:*

- Stage $i+2$ of Phase 1 begins and Property $\mathcal{P}(p')$ holds for some integer p' such that $p < p' \leq 1$ (thus $p' = 1$);
- Phase 2 begins and Property $\mathcal{P}'_L(p', q)$ holds for some integers p' and q such that $p' \geq 2$, $q \leq 0$ and $p' - q < n - 2$;
- all agents of the non-empty set of surviving agents have terminated and located the black hole.

Moreover, each edge has been traversed at most a constant number of times between times t and t' .

Proof: From the hypothesis of the lemma, Property $\mathcal{P}(p)$ holds, with $p \in \{0, 1\}$. Let L and

R be respectively the left and right agent. The integer p is the number of the node where L stole R 's pebble at the end of the previous stage.

In the first case of Property $\mathcal{P}(p)$, the node $p + 1$ is the black hole and Agent R never acts again, as it is dead in it. In this case, Agent L explores the ring from right to left, using both pebbles, until it reaches node $p - n + 2$ (i.e. node $p + 2$) in state **Ping-Left**. There, the termination condition is satisfied for Agent L and it locates the black hole and terminates. Note that L does explore node $p - n + 2$ in state **Ping-Left** and thus terminates because $p - n + 2$ is at most $p - 1$ (a ring has at least three nodes).

In the remaining of the proof we assume that we are not in the first case of Property $\mathcal{P}(p)$ and thus that the node $p + 1$ is not the black hole. It follows that Agent R eventually comes back from node $p + 1$ to node p with its variable **Last_Right** set to $p + 1$. It then switches to state **Pong-Right** and goes left until it finds a pebble. Agent R does eventually find a pebble because at the beginning of the stage there is a pebble at its left (at node $p - 1$), and Agent L never removes a pebble before putting the other pebble further to the left. Let q be the number of the node where R eventually finds and takes one of L 's pebbles. Note that $q \leq 0$.

At this point, if the termination condition of R is satisfied (i.e. $p + 1$ minus its new value **Last_Left** equals $n - 2$), then R terminates by locating the black hole correctly at the only node it never visited (node $p + 2$). Moreover R terminates without taking the found pebble. Therefore, L continues its exploration to the left, using both pebbles, until it dies in the black hole. Hence, the lemma holds in this case. Note that L may die in the black hole before R terminates.

Otherwise (R does not terminate), Agent R comes back to the right in state **Ping-Right**, holding the found pebble. If, at this time, L has already moved to node $q - 1$ in state **Explore-Left** and if this node is the black hole, then L is dead and the only surviving agent R eventually visits node $n + q - 2$ while going right and terminates by locating the black hole at node $n + q - 1$, that is $q - 1$. Otherwise, L eventually retrieves the second pebble (if it does not hold it already) and goes right in state **Ping-Right**. As a consequence, the stage number is now $i + 1$.

To summarize, both agents eventually go right in state **Ping-Right** holding a pebble. Moreover one can show that the value **Last_Left** is the same for each agent. Indeed, if $q - 1$ is not smaller than their value of **Last_Left** at time t , then **Last_Left** still has this value, and otherwise **Last_Left** has now value q for both agents, even if L went to node $q - 1$ (L updates its value **Last_Left** only in state **Ping-Left**). The remaining of the proof is very similar to the last two paragraphs of the proof of Lemma 1. Indeed, at some point, one agent R_L in state **Explore-Right** will eventually find the pebble of the other agent R_R and execute its transition function while R_R 's pebble still lies at the current node p' . (At the latest, this occurs when one of the agent locates the black hole or dies in it. In both cases, its pebble stays forever at the node preceding the black hole.) From the description of the algorithm in state **Ping-Right** and the fact that one of the agents has visited node $p + 1$, we know that this agent will neither drop its pebble nor look at a pebble at node p . Therefore, we have $p' > p$.

If the termination condition of R_L is satisfied after the update of the variable `Last_Right`, then R_L terminates by locating the black hole at its actual position $p' + 1$. Concerning R_R it has already terminated by locating the black hole at $p' + 1$ (both agents have the same `Last_Left` value). Otherwise (the termination condition of R_L is not satisfied) R_L changes role. At this time, if $p' < 2$, then $p' = 1$ and Property $\mathcal{P}(1)$ is satisfied, and if $p' \geq 2$, then Property $\mathcal{P}'(p', q')$ is satisfied, where $q' = \text{Last_Left}$.

Since the beginning of stage i (time t), each edge has been traversed at most a constant number of times. Indeed, during stage i , an edge is traversed at most three times by L and at most twice by R (in fact at most once except possibly for edge $\{p, p + 1\}$). In stage $i + 1$, if it exists, each edge is traversed at most 7 times by each agent for the reasons detailed in the proof of the previous lemma. \square

Lemma 3 *Consider a n -node ring containing a homebase and a black hole, and two agents running Algorithm **EnhancedPingPong** from the homebase. Assume that at some time t a Phase-2 stage of odd number i begins and that either Property $\mathcal{P}'_L(p, q)$ or Property $\mathcal{P}'_R(p, q)$ holds for some integers p and q such that $p \geq 2$, $q \leq 0$ and $p - q < n - 2$. Then at some time $t' > t$ one of the following situations occurs:*

- Stage $i + 2$ of Phase 2 begins and either Property $\mathcal{P}'_L(p', q')$ or Property $\mathcal{P}'_R(p', q')$ holds for some integers p' and q' such that $p' \geq p$, $q' \leq q$ and $n - (p' - q' + 1) \leq \lceil \frac{n(p-q+1)}{2} \rceil$;
- all agents of the non-empty set of surviving agents have terminated and located the black hole.

Moreover, each edge has been traversed at most a constant number of times between times t and t' .

Proof: We assume that at some time t a Phase-2 stage of odd number i begins and that Property $\mathcal{P}'_L(p, q)$ holds for some integers p and q . The case when Property $\mathcal{P}'_R(p, q)$ holds is simpler, as there are less cases to consider, and can be treated similarly. Let $p - k - 1$ be the node where L lies at time t . Note that the node $p - k - 2$ is not the black hole by definition of Property $\mathcal{P}'_L(p, q)$.

In the first case of Property $\mathcal{P}'_L(p, q)$, the node $p + 1$ is the black hole and Agent R never acts again, as it is dead in it. In this case, Agent L explores the ring from right to left, using the pebble it carried at the beginning of the stage, until it reaches node `Goal_Left` in state `Halving-Explore-Left`. Note that since $p - q < n - 2$, we have $q > \text{Goal_Left} \leq p + 1 - n$ and thus the node `Goal_Left` is reached safely without dying in the black hole. There, either the termination condition is satisfied for Agent L (`Goal_Left` = $p + 2 - n$) and it locates the black hole and terminates, or the termination condition is not satisfied. In this latter case, it means that $p - q' < n - 2$, where $q' = \text{Goal_Left}$. Therefore L comes back to node $q' + 1$ to retrieve its pebble and heads back toward the node `Last_Right` in state `Halving-Ping-Right`. The stage number is now $i + 1$. Since Agent R does not act anymore, the pebble at node $p - k - 1$ is still there. Hence Agent L will eventually find it in state `Halving-Ping-Right` and then in state `Halving-Explore-Right`. There L steals the pebble, changes role again, and go to node $p - k - 2$ (where the other pebble lies) in state `Halving-From-Right-To-Left`. The stage $i + 2$ begins. Furthermore one can easily check

that Property $\mathcal{P}'_L(p, q')$ holds. In particular, we do have that $p - k - 2 > q - 1 \geq q'$. Since $q' = \text{Update_Goal_Left}(p, q, 1)$, we have $n - (p - q' + 1) \leq (n(p - q + 1))/2$.

We assume now that we are not in the first case of Property $\mathcal{P}'_L(p, q)$, that is, $p + 1$ is not the black hole. Let us consider two cases, depending in which half lies the black hole.

Case 1: The black hole is in the left half, between node q and node **Goal_Left** (inclusive). In this case, the left agent L will never come back to the right part because it would first visit the black hole and thus die. Hence, when the right agent eventually wakes up, it goes left in state **Halving-Pong-Right** and finds the pebble at node $p - k - 1$. There it updates its values **Goal_Left** and **Goal_Right**. Note that, by definition of the algorithm and by hypothesis of Property $\mathcal{P}'_L(p, q)$, both agents now agree on the values **Goal_Left** and **Goal_Right**. At this point, agent R goes right and explore its half, from node p to node $p' = \text{Goal_Right}$, which is at least $p + 1$. If the termination condition of R is satisfied, then R locates the black hole at its correct position and terminates. Otherwise it changes role and the stage switches to $i + 1$. Similarly as in the proofs of the two previous lemmas one agent R_R in state **Halving-Explore-Left** will eventually find the pebble of the other agent R_L and execute its transition function while R_L 's pebble still lies at the current node q' . (At the latest, this occurs when one of the agent dies in it because in this case, its pebble stays forever at the node preceding the black hole.) Agent R_R steals the pebble and goes to node $q' + 1$ in state **Halving-From-Left-To-Right**. The stage $i + 2$ begins. Furthermore one can easily check that Property $\mathcal{P}'_R(p', q')$ holds. Since $q' \leq q$ and $p' = \text{Update_Goal_Left}(p, q, 1) + n - 1$, we have $n - (p' - q' + 1) \leq \lceil (n(p - q + 1))/2 \rceil$;

Case 2: The black hole is in the right half, between node p and node **Goal_Right** (inclusive).

In this case, the left agent L will eventually explore its half, from node q to node $q' = \text{Goal_Left}$, which is at most $q - 1$. The termination condition of L is not satisfied because we assumed that $p + 1$ is not the black hole. Hence it changes role and the stage switches to $i + 1$. Agent L heads back toward the node **Last_Right** in state **Halving-Ping-Right** while looking at R 's pebble. If meanwhile R did not retrieve the pebble at node $p - k - 1$, then L finds this pebble in state **Halving-Explore-Right**. There L steals the pebble, changes role again, and go to node $p - k - 2$ (where the other pebble lies) in state **Halving-From-Right-To-Left**. The stage $i + 2$ begins. Furthermore Property $\mathcal{P}'_L(p, q')$ holds. Otherwise R did come back from node $p + 1$ to node $p - k - 1$ and got back the pebble. There it updates its values **Goal_Left** and **Goal_Right**. Note that, by definition of the algorithm and by hypothesis of Property $\mathcal{P}'_L(p, q)$, both agents now agree on the values **Goal_Left** and **Goal_Right**. Similarly as before, one agent R_L in state **Halving-Explore-Right** will eventually find the pebble of the other agent R_R and execute its transition function while R_R 's pebble still lies at the current node p' . Agent R_L steals the pebble and goes to node $p' - 1$ in state **Halving-From-Left-To-Right**. The stage $i + 2$ begins. Furthermore one can easily check that Property $\mathcal{P}'_R(p', q')$ holds. Since $p' \geq p$ and $q' = \text{Update_Goal_Left}(p, q, 1)$, we have $n - (p' - q' + 1) \leq (n(p - q + 1))/2$.

It remains to bound the number of edge traversals since time t . During stage i each edge is traversed at most three times by each agent. During stage $i + 1$, if it exists, each edge

is traversed at most 7 times by each agent, for the same reasons as detailed in the proof of Lemma 1. \square

Theorem 1 *Algorithm `EnhancedPingPong` is correct.*

More precisely, consider a n -node ring containing a homebase and a black hole, and two agents running Algorithm `EnhancedPingPong` from the home base. After finite time, there remains at least one surviving agent and all surviving agents have terminated and located the black hole.

Proof: From Lemmas 1 and 2, we know that the first phase contains at most five stages, each one ending after finite time. Furthermore we know that after finite time, either the algorithm terminates correctly, or Property $\mathcal{P}'_L(p, q)$ or $\mathcal{P}'_R(p, q)$ holds, for some integers p and q such that $q \leq 0 < p$ and $0 < p - q < n - 2$. From Lemma 3, we know that a stage of Phase 2 ends after finite time. We also know that if the algorithm does not terminate after two stages $i, i + 1$ in Phase 2, then Property $\mathcal{P}'_L(p', q')$ or $\mathcal{P}'_R(p', q')$ holds, for some integers p' and q' such that the positive value $p' - q'$ is strictly less than $p - q$. Hence, after finite time, neither $\mathcal{P}_L(p, q)$ nor $\mathcal{P}'_R(p, q)$ can be satisfied and the algorithm terminates correctly. \square

Theorem 2 *The total number of moves performed by two agents running Algorithm `EnhancedPingPong` in a n -node ring is at most $O(n \log n)$.*

Proof: From Lemmas 1 and 2, there are at most five stages in Phase 1 and for each of them the number of edge traversals performed by each agent is at most $O(n)$. From Lemma 3, there are at most $O(\log n)$ stages in Phase 2 because the unexplored part is basically halved every two stages. From the same lemma, we have that for each Phase-2 stage the number of edge traversals performed by each agent is at most $O(n)$. Hence, overall, the total number of moves performed by two agents running Algorithm `EnhancedPingPong` in a n -node ring is at most $O(n \log n)$. \square

The optimality of the algorithm follows from the fact that, in a ring, the problem cannot be solved with less agents or (asymptotically) less moves [12], and clearly not with less pebbles.

4 Black hole search in arbitrary graphs

4.1 Preliminaries

In this section, both agents are provided with a map of the network containing all edge labels and a mark showing the position of the homebase in this network. Thus, each node of the map can be uniquely identified (for example by a list of edge labels leading to it from the homebase). Therefore, each agent is able to know where it lies at any point of the execution of the algorithm. It also knows where each edge incident to its position leads.

The algorithm `GeneralizedEnhancedPingPong` we propose for arbitrary networks is an adaptation of the algorithm `EnhancedPingPong` that we described for rings. To be able to

apply **EnhancedPingPong** in a general graph, each agent will maintain a partial mapping between the node numbers used in the algorithm and the actual nodes in the network (or its map), such that at any point in time an agent knows what means “go left” and “go right”.

During the execution of the algorithm, each agent maintains two walks W_R and W_L , defined as two sequences (r_0, r_1, \dots, r_P) and (l_0, l_1, \dots, l_Q) of nodes of the network. The nodes r_0 and l_0 correspond to the homebase. Since W_R and W_L are walks, we have that $\{r_i, r_{i+1}\}$ and $\{l_j, l_{j+1}\}$ are edges of the graph, for all $0 \leq i < P$ and $0 \leq j < Q$.

From these two walks, we define recursively function σ as follows. First $\sigma(0) = 0$. Assume that σ is defined for all j such that $0 \leq j \leq i$, for some $i \geq 0$. Then if there exists an element r_K in W_R such that $r_K \notin \{r_{\sigma(0)}, r_{\sigma(1)}, \dots, r_{\sigma(i)}\}$ but for all $k < K$, $r_k \in \{r_{\sigma(0)}, r_{\sigma(1)}, \dots, r_{\sigma(i)}\}$, then $\sigma(i+1) = K$, otherwise $\sigma(i+1)$ is not defined. Similarly, assume that σ is defined for all j such that $i \leq j \leq 0$, for some $i \leq 0$. Then if there exists an element l_K in W_L such that $l_K \notin \{l_{\sigma(0)}, l_{\sigma(1)}, \dots, l_{\sigma(i)}\}$ but for all $k < K$, $l_k \in \{l_{\sigma(0)}, l_{\sigma(1)}, \dots, l_{\sigma(i)}\}$, then $\sigma(i-1) = K$, otherwise $\sigma(i-1)$ is not defined.

Let us assume that an agent lies at some node i . If $i \geq 0$ (i.e., the agent is at the homebase or somewhere in the explored part to the right) going one step right from node i means following the sub-walk $(r_{\sigma(i)}, r_{\sigma(i)+1}, \dots, r_{\sigma(i+1)})$ of W_R . Going left from node $i+1$ to node i means following this sub-walk in reverse order. Similarly, if $i \leq 0$ (i.e., the agent is at the homebase or in the explored part to the left) going one step left from node i means following the sub-walk $(l_{\sigma(i)}, l_{\sigma(i)+1}, \dots, l_{\sigma(i-1)})$ of W_L . Going right from node $i-1$ to node i means following this sub-walk in reverse order.

4.2 The Algorithm

We now describe the definitions of the walks W_R and W_L throughout the algorithm. First of all, node v_0 denotes the homebase. Node v_1 is the neighbor of node v_0 reachable by the smallest edge label while node v_{-1} is the neighbor of node v_0 reachable by the largest edge label.

At the beginning of the algorithm, let T_R be a tree spanning all nodes except for node v_{-1} and containing the edge $\{v_0, v_1\}$. Let W_R be a DFS traversal of T_R starting from node v_0 by the edge $\{v_0, v_1\}$. Let W_L be (v_0, v_{-1}) . Clearly, nodes v_{-1}, v_0 and v_1 are the nodes -1, 0 and 1.

Assume that the stage changes from an odd number to an even number in Phase 1. Let p and q be the values **Last_Right** and **Last_Left** of the left agent. Then the new walk W_R consists of the first $\sigma(p+1) + 1$ elements of the old W_R , that is the sequence $(r_0, r_1, \dots, r_{\sigma(p+1)})$. In addition, let T_L be a tree spanning all nodes except for node $p+1$. Let S_L be a DFS traversal of T_L starting from node $q-1$. Then the new walk W_L is the concatenation of the old W_L and of the sequence S_L . The left agent does these updates of the walks when changing role. The other agent does these updates when it finds out that its pebble has been stolen. More precisely, it updates its walks just before switching to state **Pong-Right**. Note that both agents agree on the new definition of the walks because they use the same values for p and for q (cf. Property $\mathcal{P}(p)$).

Similarly assume that the stage changes from an even number to an odd number in Phase 1. Let p and q be the values `Last_Right` $- 1$ and `Last_Left` of the right agent. Then the new walk W_L consists of the first $\sigma(q - 1) + 1$ elements of the old W_L , that is the sequence $(l_0, l_1, \dots, l_{\sigma(q-1)})$. In addition, let T_R be a tree spanning all nodes except node $q - 1$. Let S_R be a DFS traversal of T_R starting at node $p + 1$. Then the new walk W_R is the concatenation of the old W_R and of the sequence S_R . The right agent does these updates of the walks when it retrieves a pebble, just before switching from state `Pong-Right` to state `Ping-Right`. The other agent does these updates when it finds out that its pebble has been stolen. More precisely, it updates its walks just before changing role. Note that again both agents agree on the new definition of the walks because they use the same values for p and for q .

The walks are also updated at the beginning of each Phase-2 stage of odd number i . More precisely this is done by an agent each time and just after it updates its knowledge of the unexplored part and its goals. Assume w.l.o.g. that the stage is now i because a right agent became a left agent. Let p , q and g be the values, respectively, of `Last_Right`, `Last_Left` and `Goal_Right` just after the update of the goals. Let $\{V_{ex}, V_{uex}\}$ be a partition of the nodes of the graph such that V_{ex} is the set of nodes $\{q, q + 1, \dots, p - 1, p\}$. From Lemma 5.2 in [11], V_{uex} can be partitioned into V_R and V_L such that $|V_R| = p - g$, the node $p + 1$ is in V_R , and the graphs G_R and G_L induced by, respectively, $V_{ex} \cup V_R$ and $V_{ex} \cup V_L$ are connected. Let T_R and T_L be spanning trees of G_R and G_L . Let S_R be a DFS traversal of T_R starting at node $p + 1$. Similarly let S_L be a DFS traversal of T_L starting at node q . Finally, let W'_R , resp. W'_L , consists of the first $\sigma(p + 1)$, resp. $\sigma(q)$, elements of W_R , resp. W_L . Then the new walks W_R and W_L are respectively the concatenation of W'_R and S_R and the concatenation of W'_L and S_L . Note that both agents agree on the new definition of the walks because they use the same values for p , q and g (cf. Properties $\mathcal{P}'_R(p, q)$ and $\mathcal{P}'_L(p, q)$).

In some cases, it is possible to use (safe) shortcuts to decrease the number of moves. Indeed, always following the walks W_R and W_L to go right and left may lead to a total of $n \log^2 n$ moves. The algorithm is modified as follows. During Phase 2, each agent maintains an additional variable `Last_Seen_Pebble` that basically memorizes the last place where the agent has seen the other pebble. When an agent finishes a half and switches to state `Halving-Ping-Left`, resp. `Halving-Ping-Right`, it goes directly to node `Last_Seen_Pebble` and if there are no pebbles at this node, it then goes directly to node `Last_Left`, resp. `Last_Right`. This is done by traversing only nodes that are known to be safe.

4.3 Correctness and complexity

Theorem 3 *Algorithm `GeneralizedEnhancedPingPong` is correct.*

More precisely, consider a n -node graph containing a homebase and a black hole, and two agents running Algorithm `GeneralizedEnhancedPingPong` from the homebase. After finite time, at least one agent survives and all surviving agents have terminated and located the black hole.

Proof: As noticed in the description of the algorithm in the previous subsection, the two agents agree on the definition of the walks and thus of the node numbers. Moreover, one can

easily check that the function σ defining the node numbers always gives the same number to the same node as soon as this node has been explored by at least one agent. Indeed, if a node $i \geq 0$ is explored, then the initial part $(r_0, r_1, \dots, r_{\sigma(i)})$ of W_R is kept unchanged forever. A similar property holds for $i \leq 0$. Finally note that a node of the graph has at most one pre-image by σ .

To summarize, Algorithm `GeneralizedEnhancedPingPong` behaves exactly the same as Algorithm `EnhancedPingPong`. The only difference is that traversing an edge in the ring may correspond to the traversals of (finitely) many edges in an arbitrary graph. Nevertheless, since Algorithm `EnhancedPingPong` is correct, Algorithm `GeneralizedEnhancedPingPong` is correct as well. \square

Theorem 4 *The total number of moves performed by two agents running Algorithm `GeneralizedEnhancedPingPong` in a n -node graph is at most $O(n \log n)$.*

Proof: In this proof we call the number of edge traversals performed by an agent going from node i to node $i + 1$ ($-n < i < n - 1$) the length of the virtual edge $\{i, i + 1\}$. We now bound the total number of moves performed by each agent in each phase.

As in the case of the ring, the first phase consists of at most five stages. Moreover, each update of the walks increases their length by at most $2n$ because the path appended to a walk is a DFS traversal of a tree. Hence, the sum of all the lengths of the virtual edges traversed in the first phase is at most $10n$. From Lemmas 1 and 2, each edge of the network has been traversed at most a constant number of times during Phase 1. Hence, the total number of moves performed by two agents running Algorithm `GeneralizedEnhancedPingPong` is at most $O(n \log n)$ in the first phase.

From the lemmas 1, 2 and 3, either Property $\mathcal{P}'_R(p, q)$ or Property $\mathcal{P}'_L(p, q)$ holds, for some integers p and q , at the beginning of a Phase-2 stage of odd number. Let p_i and q_i be the two integers corresponding to the stage $2i + 1$ of the second phase, for $0 \leq i \leq s$. Note that s is at most $O(\log n)$. Let p_{s+1} , resp. q_{s+1} , be the right, resp. left, neighbor of the black hole. By definition of the properties and from the lemmas, we have that $q_{s+1} \leq \dots \leq q_0 \leq 0 \leq p_0 \leq \dots \leq p_{s+1}$. Since the walks W_R and W_L are updated when and only when the goals are updated, and since a walk is always extended by a DFS traversal of a tree, we obtain that the sum of all the lengths of the virtual edges from node p_i to p_{i+1} , and from node q_i to q_{i+1} , is at most $O(n)$, for $0 \leq i \leq s$. Moreover, from the previous paragraph, the sum of all the lengths of the virtual edges from node q_0 to p_0 is at most $O(n)$.

Consider a stage $2i + 1$ of the second phase, for $0 \leq i \leq s$. Let A be the agent that started the stage by changing role and let B the other agent. Without loss of generality, assume that A is a left agent. The total number of moves performed by A in this stage is at most $O(n)$ because A first goes directly (by a shortest safe path) to the beginning q_i of its workload, thus in at most n moves, and then stays in the final part of W_L that corresponds to the DFS traversal of a tree to explore its assigned workload, which incurs at most $O(n)$ moves (from Lemma 3). If it succeeds to explore its half, then it goes directly to the node where it left the other pebble (thanks to the variable `Last.Seen.Pebble`). If the pebble is not there anymore, it further goes directly to node p_i and starts to explore the right half. Hence, in

any case, for the same reasons as before, A performs at most $O(n)$ moves in stage $2i + 2$. Concerning B , if it retrieves its pebble in stage $2i + 1$ or $2i + 2$, it will perform at most $O(n)$ moves in these two stages, without counting the moves done in state **Halving-Pong-Right**. Indeed, again, exploring a half or going directly to the beginning of it costs at most a linear number of moves.

It remains to bound the number of moves done while in one of the states **Halving-Pong-Right** or **Halving-Pong-Left**. This is done globally over the whole second phase. Each edge traversed in one of these two states may cost up to $O(n)$ moves. However, there are at most $O(\log n)$ such traversals because any of them corresponds to an update of the workloads, which happens only a logarithmic number of times in the entire algorithm.

One can now conclude that the total number of moves performed by two agents running Algorithm **GeneralizedEnhancedPingPong** in a n -node graph is at most $O(n \log n)$. \square

The optimality of the algorithm follows from the fact that, in an arbitrary graph, BHS cannot be solved with less agents or (asymptotically) less moves [11], and clearly not with less pebbles.

Acknowledgment This work was done during the stay of David Ilcinkas at the University of Ottawa, as a postdoctoral fellow. Paola Flocchini was partially supported by the University Research Chair of the University of Ottawa. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Discovery grants.

References

- [1] S. Albers and M. Henzinger. Exploring unknown environments. *29th ACM Symposium on Theory of Computing (STOC)*, 416–425, 1997.
- [2] M. A. Bender, A. Fernández, D. Ron, A. Sahai, and S. P. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation*, 176(1):1–21, 2002.
- [3] J. Cao and S. Das (Eds), *Mobile Agents in Networking and Distributed Computing*, John Wiley, 2008.
- [4] J. Chalopin, S. Das, N. Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. *21st Conf. on Distributed Comp. (DISC)*, 108-122, 2007.
- [5] C. Cooper, R. Klasing, and T. Radzik. Searching for black-hole faults in a network using multiple agents. *10th Int. Conf. on Principles of Distributed Systems (OPODIS)*, 320-332, 2006.
- [6] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71 (2-3): 229-242, 2006

- [7] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability & Computing* 16: 595-619, 2007.
- [8] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science* 385(1-3): 34-48, 2007.
- [9] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *J. Graph Theory* 32 (3): 265–297, 1999.
- [10] S. Dobrev, P. Flocchini, R. Kralovic, and N. Santoro. Exploring a dangerous unknown graph using tokens. *5th IFIP Int. Conf. on Theoretical Computer Science (TCS)*, 131-150, 2006.
- [11] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocol. *Distributed Computing* 19 (1): 1-19, 2006.
- [12] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica* 48: 67–90, 2007.
- [13] S. Dobrev, R. Kralovic, N. Santoro, and W. Shi. Black hole search in asynchronous rings using tokens. *6th Conf. on Algorithms and Complexity (CIAC)*, 139-150, 2006.
- [14] P. Flocchini and N. Santoro. *Distributed Security Algorithms For Mobile Agents*. Chapter 5 of [3], 2008.
- [15] P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48 (3): 166-177, 2006.
- [16] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345 (2-3): 331-344, 2005.
- [17] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Approximation bounds for black hole search problems. *9th Int. Conf. on Principles of Distributed Systems (OPODIS)* 261-274, 2005.
- [18] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theoretical Computer Science* 384 (2-3): 201-221, 2007.