Ping Pong in Dangerous Graphs: Optimal Black Hole Search with Pebbles

Paola Flocchini, David Ilcinkas, Nicola Santoro

Algorithmica manuscript No.
(will be inserted by the editor)

Ping Pong in Dangerous Graphs: Optimal Black Hole Search with Pebbles

Paola Flocchini ·
David Ilcinkas ·
Nicola Santoro

Received: date / Accepted: date

Abstract We prove that, for the black hole search problem in networks of arbitrary but known topology, the *pebble* model of agent interaction 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. These results are obtained with a novel algorithmic technique, *ping-pong*, for agents using pebbles.

Keywords Distributed computing · Graph exploration · Mobile agents · Autonomous robots · Dangerous graphs

A preliminary version of this paper appeared in the Proceedings of the 22nd International Symposium on Distributed Computing (DISC 2008) [15].

P. Flocchini
SITE, University of Ottawa, Ottawa, ON K1N 6N5, Canada
E-mail: flocchin@site.uottawa.ca

D. Ilcinkas
LaBRI, CNRS & Université de Bordeaux, France
E-mail: david.ilcinkas@labri.fr

N. Santoro
SCS, Carleton University, Ottawa, Ontario, K1S 5B6, Canada
E-mail: santoro@scs.carleton.ca

1 Introduction

1.1 The Framework

Black Hole Search (BHS) is the distributed problem in a networked system (modelled 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 [17]).

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., 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 [1,2,9,10,18,19]) 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 of locating a single black hole 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 [5,7,8,21–23] where all agents are synchronized and delays are unitary, the largest body of the investigations has concentrated on the asynchronous one (e.g., [4,11–14,16,20]).

More specifically, in synchronous networks, tight bounds have been established for some classes of trees [8]. In the case of general networks the decision problem corresponding to the one of finding the optimal strategy is shown to be NP-hard [7, 22] and approximation algorithms are given in [7] and subsequently improved in [21]. The case of multiple black holes have been investigated in [5] where a lower bound on the cost and close upper bounds are given.

In asynchronous networks with whiteboards, a complete characterization has been done for the localization of a single black hole in a ring [13], providing protocols that are optimal in size, number of moves, and time. In [12], arbitrary topologies have been considered and optimal location algorithms have been proposed under a variety of assumptions on the agents' knowledge (knowledge of the topology, presence of sense of direction). In [20], the effects of knowledge of incoming link on the optimal team size is studied and lower bounds are provided. The case of black links in arbitrary networks has been studied in [4, 16], respectively for anonymous and non-anonymous nodes providing non-optimal solutions in the case of multiple black holes and links in [4], and conditions for solvability in [16]. All the previous literature considers black holes in undirected graphs; the case of directed graphs has been recently investigated in [6], where it is shown that the requirements in number of agents change considerably. A variant of dangerous node behavior is studied in [24], where the authors introduce *gray holes* (i.e. black holes with Byzantine behavior, which do not always destroy a passing agent) and consider the periodic ring exploration problem. Finally, variations of the pebble model where a pebble can be placed on a node in correspondence of ports, or multiple pebbles can share the same node, have been investigated in [11, 14, 25]. More details about those approaches are given in the next Section when describing the challenges we face when dealing with pebbles.

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 *pebble* 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 goal 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 [11] it has been shown that $\Delta+1$ agents[1] 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 [14] 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 [11,14,25] 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 single pebble model used in the traditional exploration problem. In fact, in [11,14,25] 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 pebble the computational power of $O(\log \Delta)$ bits of information, while additionally in [14,25] each agent has available several pebbles, and multiple pebbles can be placed at the exact same place to store even more information.

1.3 Our Results

In this paper, we provide the first proof that, indeed, the pebble model is computationally as powerful as the whiteboard model for Bhs in networks of known topology. 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 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 pebbles it is possible to obtain exactly the same optimal bounds for team size and number of moves as using whiteboards.

[1] Δ denotes the maximum node degree in G

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 [12] 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 that we have developed using a variety of novel not-trivial techniques. These mechanisms 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 pebbles).

The paper is organized as follows. We first define the problem and some useful terminology in Section 2. We then present our techniques, prove their properties and analyze their complexity in the case of ring networks (Section 3). 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. Finally, open research questions and problems are discussed in Section 5.

2 Terminology and Definitions

Let $G = (V, E)$ be a simple biconnected[2] 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, λ being the function specifying the link labels.

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 behavioural 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) travelling 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

[2] Note that biconnectivity is necessary for BHS to be solvable in the worst case [12]. It is actually sufficient that the removal of the black hole does not disconnect the graph.

exclusion as an atomic action, i.e. as if it takes 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.

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 network contains exactly one black hole and its location 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. An edge is said to be *safe* if the black hole is not one of its extremities.

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

In this section we present our techniques, prove their properties, and analyze their complexity when the network is a ring. These techniques will be modified and enhanced in the next section to work in the arbitrary graphs with the same time bounds.

3.1 Preliminaries

Let the network be a ring whose size n is known to the two agents. 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 homebase will be called node i, resp. node -i. Hence, nodes 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 visited by this agent 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 its (right) neighbour 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 its (left) neighbour 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 pebbles, *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 generates "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 using cautious walk. This is done by cycling through the states `Explore-Right`, `Pick-Up-Right`, and `Ping-Right`, as explained in the last paragraph of Section 3.1.

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 R_1 finds the pebble of the other agent R_2, say at node p (at the latest it happens when one agent locates or dies in the black hole). Agent R_1 needs to be empty-handed at this point to be able to steal R_2's pebble. Hence, if R_1 is currently carrying its pebble, it goes back to node $p-1$ in state `Put-Pebble-Right`, drops there its pebble, and moves again (without a pebble) to p. By that time, R_2 might have already picked up his pebble and moved on. In this case, R_1 resumes the normal activity of a right agent, that is, it explores to the right using cautious walk.

When it eventually happens that one agent L finds the pebble of the other agent R at some node p, L's pebble being at node $p-1$, the agent L steals R's pebble. (This happens with L being in state `Explore-Right`.) Having control on the two pebbles, L becomes a left agent (by switching to state `Ping-Left`) and starts to explore left using double cautious walk. This is done by cycling through the states `Explore-Left`, `Pick-Up-Left`, and `Ping-Left`, as explained in the last paragraph of Section 3.1. The stage has now an even number.

When/if R comes back to p in state `Pick-Up-Right` 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 L never removes a pebble before putting the other pebble further to the left. Thus R eventually retrieves the pebble and goes right again in state `Ping-Right`, resuming exploration 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`). This is done by staying in state `Ping-Right` until it reaches node `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 $O(n \log n)$, an agent switches to Phase 2 when at least two nodes have been explored to the right.

More precisely, this happens when a right agent R_1 steals the pebble of another right agent R_2 at some node p with $p \geq 2$. Agent R_1 starts Phase 2 by changing role, becoming a left agent. It switches from state `Explore-Right` to `Halving-From-Right-To-Left` (instead of `Ping-Left`). Agent R_2 switches to the Phase-2 state `Halving-Pong-Right` (instead of `Pong-Right`) when/if it realizes in state `Pick-Up-Right` that its own pebble has been stolen.

The state transitions of the algorithm used in Phase 1 are summarized in Figure 1, where dashed arrows correspond to changing role, and dotted edges correspond to transitions to Phase-2 states. The transitions used by a right agent to perform cautious walk and the ones used by a left agent to perform double cautious walk are depicted by bold arrows. The letter on an arrow indicates the direction of the move associated to the corresponding state transition. (L and R stand for Left and Right respectively.)

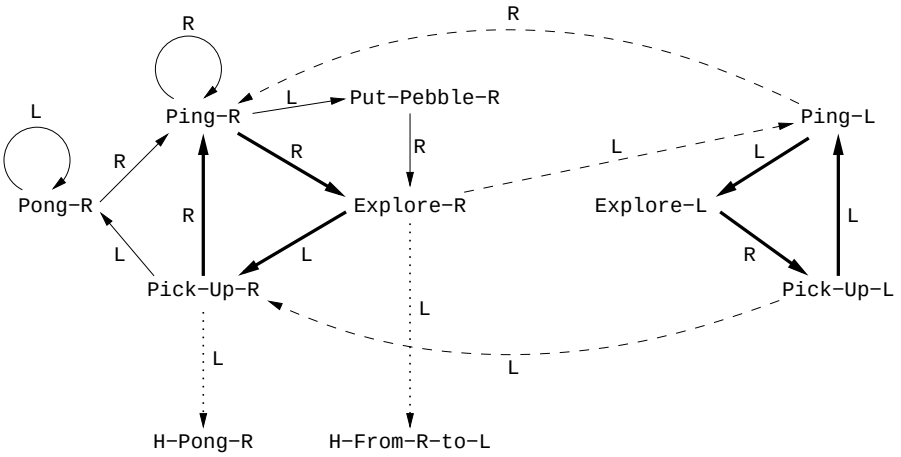

Fig. 1 State transitions of Algorithm `EnhancedPingPong` in Phase 1.

Phase 2. Phase 2 uses the *halving* technique, based on an idea of [13], 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. W.l.o.g., assume that both agents are right agents. Similarly as in Phase 1, it eventually happens that one agent A finds the pebble of the other agent B at some node p, A's pebble being at node $p-1$ (possibly thanks to a move

back in state `Halving-Put-Pebble-Right`). At this point, A steals B's pebble. For B, it looks like its pebble has been moved by one position away from him. This pebble move is used to indicate a change of stage to B. Agent A then computes the new workload, divide it into two parts (using the function `Update_Goal_Left`), and goes and explores its newly assigned part, changing role again by switching to state `Halving-From-Right-To-Left`. (This additional state is used to ignore the presence of a pebble at node $p-1$.) 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`. 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 the current unexplored part is and what its current workload is. (In fact, it updates the goals after every move performed in state `Halving-Pong-Right`.) 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 $O(n \log n)$.

As explained before, the agents use the variables `Last_Right` and `Last_Left` to detect termination. More precisely, when only one node remains unexplored, that is when `Last_Right` $-$ `Last_Left` $= n-2$, this remaining node `Last_Right`$+1$ (or, equivalently, `Last_Left`-1) is the black hole and the agent stops. The procedure used for this purpose is called CHECK_TERMINATION.

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 in the Appendix.

3.3 Correctness and Complexity

Let us now prove the correctness of the proposed algorithm and analyze its complexity. All the proofs refer to the formal pseudocode in the Appendix.

Lemma 1 *The algorithm is well defined and the input invariants are correct.*

Proof First note that if the input invariant of a state is true, then the algorithm of this state is well defined. We now prove that the input invariants are correct, by induction on the step number of any agent in any execution. Clearly, the input invariant is correct at the first step because the agent effectively starts the algorithm with a pebble in state `Ping-Right`. Assume now that the input invariant is correct for the step $i \geq 1$.

The states can be partitioned in two sets: E and W, depending on whether the input invariant ensures that the agent is empty-handed or that the agent has a pebble, respectively.

Assume first that, at step i, the agent is in state $S \in E$. For every possible transition to a state $S' \in E$, by algorithm construction, no pebble is picked up. For every transition to a state $S" \in W$, the algorithm requires the agent to pick up the pebble, after having successfully checked that a pebble is actually present at the current node: line 7 of `Explore-Right`, line 2 of `Pick-Up-Right`, line 7 of `Pong-Right`, line 2 of `Pick-Up-Left`, line 10 of `Halving-Explore-Right` (similarly for `Halving-Explore-Left`), line 2 of `Halving-Pick-Up-Right` (similarly for `Halving-Pick-Up-Left`), line 8 of `Halving-Pong-Right` (similarly for `Halving-Pong-Left`).

Assume now that, at step i, the agent is in state $S \in W$. For every possible transition from S to a state in W, by construction of the algorithm, no pebble is put down. On the other hand, for every transition to a state $S' \in E$, the algorithm specifies that the agent puts down the pebble: line 7 of `Ping-Right`, line 1 of `Put-Pebble-Right`, line 1 of `Explore-Left`, line 9 of `Halving-Ping-Right` (similarly for `Halving-Ping-Left`), line 1 of `Halving-Put-Pebble-Right` (similarly for `Halving-Put-Pebble-Left`).

Thus, in all cases, the input invariant remains correct for step $i + 1$. □

As explained before the algorithm consists of up to two *phases*. The first one corresponds to the cases where both agents are in one of the eight states not beginning by `Halving`: `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 (i.e. at least one agent is in a state beginning by `Halving`), 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; let $i = 0$;
- it is already back from node $p+1$ at node p in state `Halving-Pick-Up-Right` and its variable `Last_Right` has value $p + 1$; let $i = 0$;
- it is waiting at node $p - 1 - i$ in state `Halving-Pong-Right`, for some $0 \leq i \leq k - 1$, and its variable `Last_Right` has value $p + 1$.

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`$(p, q', k - i + 1)$, 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; let $i = 0$;
- it is already back from node $q-1$ at node q in state `Halving-Pick-Up-Left` and its variable `Last_Left` has value $q - 1$; let $i = 0$;
- it is waiting at node $q + 1 + i$ in state `Halving-Pong-Left`, for some $0 \leq i \leq k - 1$, and its variable `Last_Left` has value $q - 1$.

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

Lemma 2 *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 $\mathcal{P}(0)$ 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 (the other pebble being at the previous node on the left). 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 $x+1$). □

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-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 $p+2$, or equivalently $q-1$). Moreover R terminates without taking the found pebble. Therefore, L eventually continues its exploration to the left, and dies immediately in the next node $q-1$, the black hole (its value `Last_Right` is only p). 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 2. 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 $q' =$ `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. □

Lemma 4 *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 fewer 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 $p+1-n <$ `Goal_Left` $< q$ 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 $q' =$ `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 $q' =$ `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 $p-k-1$. While going 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 $p' =$ `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.) If the termination condition of R_R is satisfied, then R_R locates the black hole at its correct position and terminates. Otherwise it 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 $p' = \texttt{Update\_Goal\_Left}(p, q, 1) + n - 1$, 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 $q' =$ `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. While going 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'. If the termination condition of R_L is satisfied, then R_L locates the black hole at its correct position and terminates. Otherwise it 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 $q' = \texttt{Update\_Goal\_Left}(p, q, 1)$, 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 2. □

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 homebase. After finite time, there remains at least one surviving agent and all surviving agents have terminated and located the black hole.*

Proof From Lemmas 2 and 3, 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 4, 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 stricty 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. □

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 2 and 3, 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 4, 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 $O(n \log n)$. □

The optimality of the algorithm follows from the fact that, in a ring, the problem cannot be solved with fewer agents or (asymptotically) fewer moves [13], and clearly not with fewer pebbles.

4 Black hole search in arbitrary graphs

We now show how to modify algorithm `EnhancedPingPong`, designed for ring networks, so to obtain an algorithm `GeneralizedEnhancedPingPong` for arbitrary networks with the same bounds.

Let G be an arbitrary biconnected graph. Both agents are provided with a map of G 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 in 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 cyclic ordering of the network nodes, i.e., a mapping between the node numbers used in the ring 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". Since the network does not necessarily contain an Hamiltonian cycle, a single edge in the "virtual" ring may correspond to a longer path in the actual network. In order not to increase too much the move complexity, the cyclic ordering of the network nodes is constructed by

following a DFS traversal of some spanning tree avoiding the current suspicious node (the node where the other agent is apparently blocked). Since the location of the suspicious node changes throughout the execution of the algorithm, so does the spanning tree and thus the cyclic ordering. (In fact, only the ordering of the unexplored nodes may change.) This is done carefully in such a way that both agents agree on the cyclic ordering when it is necessary.

4.1 The Algorithm `GeneralizedEnhancedPingPong`

This subsection is devoted to Algorithm `GeneralizedEnhancedPingPong`, that is, to all the modifications and adaptations performed on `EnhancedPingPong` to obtain Algorithm `GeneralizedEnhancedPingPong`.

The mapping σ. To be able to apply `EnhancedPingPong` in a general graph, each agent maintains a bijective mapping σ between the ring node numbers used in the algorithm `EnhancedPingPong` and the actual nodes in the network (or its map). The bijection is constructed as follows.

Initialization
: First of all, node $\sigma(0)$ is defined as the homebase. Node $\sigma(1)$ is the neighbour of node $\sigma(0)$ reachable by the smallest edge label, while node $\sigma(-1)$ is the neighbour of node $\sigma(0)$ reachable by the largest edge label.
Then, let T be any tree spanning all nodes except for node $\sigma(-1)$. Nodes $\sigma(2)$ to $\sigma(n-2)$ are defined as all the nodes of G except $\sigma(-1), \sigma(0), \sigma(1)$ taken in the order of their first visit by a depth-first traversal of T starting at $\sigma(1)$.

Phase 1: the stage changes from an odd number to an even number
: For the agent changing role (the left one): The update is performed when it changes role, that is just before switching to state `Ping-Left`. Let p and q be the values `Last_Right` and `Last_Left` of the agent at this moment.
For the other agent (the right one): The update is performed when it finds out that its pebble has been stolen, that is just before switching to state `Pong-Right`. Let p and q be the values `Last_Right` -1 and `Last_Left` of the agent at this moment.
The update: Let T be any tree spanning all nodes except for node $\sigma(p+1)$. Nodes $\sigma(q-2), \sigma(q-3), \ldots, \sigma(p+2-n)$ are defined as all the nodes of G except $\sigma(q-1), \ldots, \sigma(0), \ldots, \sigma(p+1)$ taken in the order of their first visit by a depth-first traversal of T starting at $\sigma(q-1)$.

Phase 1: the stage changes from an even number to an odd number
: For the agent changing role: The update is performed when it finds out that its pebble has been stolen, just before changing role, by switching either from state `Ping-Left` to state `Ping-Right` or from state `Pick-Up-Left` to state `Pick-Up-Right`. Let p and q be the values of `Last_Right` and `Last_Left` of the agent at this moment.
For the other agent: The update is performed when it retrieves a pebble,

that is just before switching to state `Ping-Right`. Let p and q be the values `Last_Right` -1 and `Last_Left` of the agent at this moment.
The update: Let T be any tree spanning all nodes except for node $\sigma(q-1)$. Nodes $\sigma(p+2), \sigma(p+3), \ldots, \sigma(q-2+n)$ are defined as all the nodes of G except $\sigma(q-1), \ldots, \sigma(0), \ldots, \sigma(p+1)$ taken in the order of their first visit by a depth-first traversal of T starting at $\sigma(p+1)$.

Phase 2: at the beginning of each stage of odd number
For each agent: The update is performed after each application of the function `Update_Goal_Right` or `Update_Goal_Left`. Let us consider w.l.o.g. the case when `Update_Goal_Left` is applied. Let p, q and g be the values, respectively, of `Last_Right`, `Last_Left` and `Goal_Left` just after the application of the function `Update_Goal_Left`.
The update: Let $\{V_{ex}, V_{uex}\}$ be a partition of the nodes of the graph such that V_{ex} is the set of nodes $\{\sigma(q), \sigma(q+1), \ldots, \sigma(p-1), \sigma(p)\}$. From Lemma 5.2 in [12], V_{uex} can be partitioned into V_R and V_L such that $|V_L| = q - g$, the node $\sigma(p+1)$ 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.
Nodes $\sigma(p+2), \sigma(p+3), \ldots, \sigma(g+n-1)$ are defined as all the nodes of G_R except for nodes $\sigma(q), \ldots, \sigma(0), \ldots, \sigma(p+1)$ taken in the order of their first visit by a depth-first traversal of T_R starting at $\sigma(p+1)$. Nodes $\sigma(q-1), \sigma(q-2), \ldots, \sigma(g)$ are defined as all the nodes of G_L except $\sigma(q), \ldots, \sigma(0), \ldots, \sigma(p)$ taken in the order of their first visit by a depth-first traversal of T_L starting at $\sigma(q)$.

Moving in the graph. Assume that the agent is currently lying at node $\sigma(i)$. Then moving left, resp. right, simply means moving to node $\sigma(i-1)$, resp. $\sigma(i+1)$, by a shortest safe path, that is it means going to that node by a shortest path in the graph induced by the already visited nodes (i.e. nodes $\sigma($`Last_Left`$)$ to $\sigma($`Last_Right`$)$).

Shortcuts. In some cases, it is possible to use (safe) shortcuts to decrease the number of moves. 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 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`. As usual, this is done by traversing only nodes that are known to be safe.

4.2 Correctness and Complexity

We will now examine the correctness and complexity of the proposed algorithm.

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 Let us first note that, for each agent and at any time, the mapping σ is well defined and bijective. Moreover, the two agents agree on the definition of the mapping σ in the following sense.

- Initially, both agents agree on the definition of σ.
- When the stage changes from an odd number to an even number in Phase 1, each agent updates σ as soon as it notices the stage number modification (when changing role for one, and when finding out that its pebble has been stolen for the other). Furthermore, both agents use the same values p and q for updating σ, by Lemmas 2 and 3 and by definition of the algorithm.
- Similarly, when the stage changes from an even number to an odd number in Phase 1, each agent updates σ as soon as it notices the stage number modification, and both agents use the same values p and q for updating σ.
- Finally, at the beginning of each stage of odd number in Phase 2, each agent updates σ as soon as it notices the stage number modification. Note however that in this case a slow agent may not be aware that the stage changed many times. It will nevertheless discover it progressively while moving back in State `Halving-Pong-Left` or `Halving-Pong-Right` (if it is not dead in the black hole). Moreover, by Lemma 4 and by definition of the algorithm, both agents use the same values for updating σ for the same stage number modification.

One can additionally note that the function σ 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 $\sigma(i)$ is explored, then any subsequent modification of σ will not redefine $\sigma(i)$.

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. □

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 $\sigma(i)$ to node $\sigma(i+1)$ (for $-n < i < n-1$) the length of the virtual edge $\{\sigma(i), \sigma(i+1)\}$. 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, the walk resulting from the concatenation of all virtual edges is at

most $10n$ (real) edges long since it results in the worst case from the concatenation of at most five DFS traversals of some trees. From Lemmas 2 and 3, 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 $O(n)$ in the first phase.

From Lemmas 2, 3 and 4, 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, neighbour of the black hole. By definition of the properties and from the lemmas, we have that $q_{s+1} \leq \cdots \leq q_0 \leq 0 \leq p_0 \leq \cdots \leq p_{s+1}$. Since the mapping σ is updated when and only when the goals are updated, and since the redefined elements are taken in the order of their first visit by a DFS traversal of some tree, we obtain that the sum of all the lengths of the virtual edges from node $\sigma(p_i)$ to $\sigma(p_{i+1})$, and from node $\sigma(q_i)$ to $\sigma(q_{i+1})$, 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 $\sigma(q_0)$ to $\sigma(p_0)$ 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 $\sigma(q_i)$ of its workload, thus in at most n moves, and then stays between nodes $\sigma(q_i)$ and $\sigma(q_{i+1})$, which incurs at most $O(n)$ moves. 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 $\sigma(p_i)$ and starts exploring 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 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 $O(n \log n)$. □

The optimality of the algorithm follows from the fact that, in an arbitrary graph, BHS cannot be solved with fewer agents or (asymptotically) fewer moves [12], and clearly not with fewer pebbles.

5 Open Problems

In this paper, we have proved that the pebble model is computationally as powerful as the whiteboard model for BHS in networks of known topology. There are several research questions and problems that now need to be asked and studied.

Ftoremost, it is still unknown whether BHS can be solved with pebbles if the agents do *not* have a map of the network. A positive answer to this question would prove that the computational equivalence, between the two models of inter-agents communication and interaction, does not depend on the availability of a priori topological knowledge to the agents. Conversely, a negative answer would prove that topological knowledge has a crucial impact on the computability of BHS.

Another class of open research problems refers to the impact that the topological structure has on the complexity of BHS with pebbles. In fact, in special classes of graphs, the number of moves to locate the black hole can be reduced from $\Theta(n \log n)$ to $\Theta(n)$ in the whiteboard model; the question arises of whether similar improvements can be obtained in the pebble model.

Acknowledgements The authors would like to thank the anonymous referees for their helpful comments.

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

References

1. S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.
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, 2009.
4. J. Chalopin, S. Das, N. Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. *21st Conference on Distributed Computing* (DISC), 108-122, 2007.
5. C. Cooper, R. Klasing, and T. Radzik. Searching for black-hole faults in a network using multiple agents. *10th International Conference on Principles of Distributed Systems* (OPODIS), 320-332, 2006.
6. J. Czyzowicz, S. Dobrev, R. Královic, S. Miklík, and D. Pardubská. Black hole search in directed graphs. *16th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), 182–194, 2009.
7. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71 (2-3): 229-242, 2006

8. 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.
9. 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.
10. X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *J. Graph Theory* 32 (3): 265–297, 1999.
11. S. Dobrev, P. Flocchini, R. Kralovic, and N. Santoro. Exploring a dangerous unknown graph using tokens. *5th IFIP International Conference on Theoretical Computer Science* (TCS), 131-150, 2006.
12. 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.
13. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica* 48: 67–90, 2007.
14. S. Dobrev, R. Kralovic, N. Santoro, and W. Shi. Black hole search in asynchronous rings using tokens. *6th International Conference on Algorithms and Complexity* (CIAC), 139-150, 2006.
15. P. Flocchini, D. Ilcinkas, and N. Santoro. Ping Pong in dangerous graphs: Optimal black hole search with pure tokens. *22nd International Symposium on Distributed Computing* (DISC), LNCS 5218, pages 227-241, 2008.
16. P. Flocchini, M. Kellett, P. Mason, N. Santoro. Map construction and exploration by mobile agents scattered in a dangerous network. *24th IEEE International Parallel and Distributed Processing Symposium* (IPDPS), 1-10, 2009
17. P. Flocchini and N. Santoro. *Distributed Security Algorithms For Mobile Agents*. Chapter 5 of [3], 2009.
18. P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48 (3): 166-177, 2006.
19. 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.
20. P. Glaus. Locating a black hole without the knowledge of incoming link. *5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks* (ALGOSENSORS), LNCS 5804, pages 128-138, 2009.
21. R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Approximation bounds for black hole search problems. *Networks*, 52(4): 216-226, 2008.
22. 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.
23. A. Kosowski, A. Navarra, and C. M. Pinotti. Synchronization helps robots to detect black holes in directed graphs. *13th International Conference on Principles of Distributed Systems* (OPODIS), LNCS 5923, pages 86-98, 2009.
24. R. Královic, S. Miklík. Periodic data retrieval problem in rings containing a malicious host. *17th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), to appear, 2010.
25. W.Shi. Black hole search with tokens in interconnected networks. *The 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems* (SSS), pages 670-682, 2009.

APPENDIX:
Detailed description of the algorithm

Procedure CHECK_TERMINATION

```
1: if Last_Right − Last_Left = n − 2 then
2:     Black_Hole := Last_Right + 1                    // Or, equivalently, Last_Left − 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 $p' =$ Update_Goal_Right$(p, q, k-1)$

Update_Goal_Left$(p, q, 0) = q$
Update_Goal_Left$(p, q, k) = \lceil \frac{p+q'-n}{2} \rceil$ where $q' =$ 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 = 0

A Phase-1 states of Algorithm EnhancedPingPong

A.1 Phase-1 states of a right agent for cautious walk

Ping-Right

Input invariant: The robot has a pebble with him.

```
 1: if current ≠ Last_Right then          // Do not use cautious walk if not necessary
 2:     Move Right
 3:     (Stay Ping-Right)
 4: else
 5:     if empty then                                  // Everything seems normal
 6:         CHECK_TERMINATION
 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}
 2: if empty then                                   // Everything seems normal
 3:     Move Left
 4:     Become Pick-Up-Right
 5: else                                            // An other right robot is preceding me
 6:     CHECK_TERMINATION
 7:     Pick_Up_Pebble                                          // Steal its 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                                            // My pebble has been stolen
 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:         Move Left
13:         Become Halving-Pong-Right
14:     end if
15: end if
```

A.2 Other Phase-1 states of a right agent

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                              // Steal back the pebble
 8:     Move Right
 9:     Become Ping-Right
10: end if
```

A.3 Phase-1 states of a left agent for double cautious walk

`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
```

B 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.

B.1 Phase-2 states of a right agent for cautious walk

`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 ≠ 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: if empty then                                      // Everything seems normal
 3:     Move Left
 4:     Become Halving-Pick-Up-Right
 5: else                                               // An other right robot is preceding me
 6:     CHECK_TERMINATION
 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                                // Steal its 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 ≠ Goal_Right then                // The current objective is not fulfilled yet
 4:         Move Right
 5:         Become Halving-Ping-Right
 6:     else                                           // The current objective is fulfilled
 7:         CHECK_TERMINATION
 8:         Move Left
 9:         Become Halving-Ping-Left
10:     end if
11: else                                               // My pebble has been stolen
12:     Last_Right := Last_Right − 1
13:     Move Left
14:     Become Halving-Pong-Right
15: end if
```

B.2 Other Phase-2 states of a right agent

`Halving-From-Left-To-Right`

Input invariant: The robot has a pebble with him.

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

`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: Last_Left := Goal_Left
2: Goal_Left := Update_Goal_Left(Last_Right, Last_Left, 1)
3: Goal_Right := Goal_Left + n − 1
4: if empty then                                   // Pebble not yet found
5:     Move Left
6:     (Stay Halving-Pong-Right)
7: else                                            // Pebble found
8:     Pick_Up_Pebble                              // Steal back the pebble
9:     Move Right
10:    Become Halving-Ping-Right
11: end if
```