Perpetual Exploration in Anonymous Synchronous Networks with a Byzantine Black Hole

Adri Bhattacharya
Indian Institute of Technology Guwahati, Assam, India

Pritam Goswami
Sister Nivedita University, Kolkata, India

Evangelos Bampas
Université Paris-Saclay, CNRS, Laboratoire Interdisciplinaire des Sciences du Numérique, 91400 Orsay, France

Partha Sarathi Mandal
Indian Institute of Technology Guwahati, Assam, India

Abstract

In this paper, we investigate the following question:

"How can a group of initially co-located mobile agents perpetually explore an unknown graph, when one stationary node occasionally behaves maliciously, under the control of an adversary?"

This malicious node is termed as "Byzantine black hole (BBH)" and at any given round it may choose to destroy all visiting agents, or none of them. While investigating this question, we found out that this subtle power turns out to drastically undermine even basic exploration strategies which have been proposed in the context of a classical, always active, black hole.

We study this perpetual exploration problem in the presence of at most one BBH, without initial knowledge of the network size. Since the underlying graph may be 1-connected, perpetual exploration of the entire graph may be infeasible. Accordingly, we define two variants of the problem, termed as PerpExploration-BBH and PerpExploration-BBH-Home. In the former, the agents are tasked to perform perpetual exploration of at least one component, obtained after the exclusion of the BBH. In the latter, the agents are tasked to perform perpetual exploration of the component which contains the *home* node, where agents are initially co-located. Naturally, PerpExploration-BBH-Home is a special case of PerpExploration-BBH. The mobile agents are controlled by a synchronous scheduler, and they communicate via *face-to-face* model of communication.

The main objective in this paper is to determine the minimum number of agents necessary and sufficient to solve these problems. We first consider the problems in acyclic networks, and we obtain optimal algorithms that solve PerpExploration-BBH with 4 agents, and PerpExploration-BBH-Home with 6 agents in trees. The lower bounds hold even in path graphs. In general graphs, we give a non-trivial lower bound of $2\Delta - 1$ agents for PerpExploration-BBH, and an upper bound of $3\Delta + 3$ agents for PerpExploration-BBH-Home. To the best of our knowledge, this is the first paper that studies a variant of a black hole in arbitrary networks, without initial topological knowledge about the network.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Design and analysis of algorithms

Keywords and phrases mobile agents, perpetual exploration, malicious host, Byzantine black hole

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.16

Related Version *Full Version*: https://arxiv.org/abs/2508.07703

Funding *Adri Bhattacharya*: Supported by CSIR, Govt. of India, Grant Number: 09/731(0178)/2020-EMR-I.

Acknowledgements We thank the DISC 2025 reviewers for their careful reading and useful feedback.

1 Introduction

In distributed mobile agent algorithms, a fundamental task is the collaborative exploration of a network by a collection of mobile agents. It was introduced and formulated by Shannon [20] in 1951. Later, the real life applicability of this problem in fields like unmanned search and rescue, monitoring, network search, etc. has garnered a lot of interest from researchers across the world which leads them to study this problem in many different settings. Ensuring the security of agents against threats or breaches is one of the major concerns in the design of exploration algorithms. Among the various security threats, two types have received the most attention in the literature of mobile agent algorithms so far. These are threats from *malicious agents* [6, 14] and *malicious hosts* [4, 8, 9]. In this work, we are interested in the latter. A malicious host in a network is a stationary node that can destroy any incoming agent without leaving any trace. Dobrev et al. introduced this type of malicious node in their 2006 paper [9], referring to such malicious hosts as *black holes*. In this paper, we will use interchangeably the terms "classical black hole" and "black hole".

There is extensive literature on *Black Hole Search* (BHS) problem, that requires locating the black hole by multiple mobile agents in a network. The BHS problem has been studied under many different scenarios and under many different communication models ([3, 4, 5, 7, 9, 10, 11]). In all of these above mentioned works, the black hole is considered to be the classical black hole which always destroys any incoming agent without fail. In this work we are interested in a more general version of the classical black hole, called *Byzantine Black Hole*, or BBH [13]. A BBH has the choice to act, at any given moment, as a black hole, destroying any data present on that node, or as a non-malicious node. Moreover, it is assumed that the initial position of the agents is safe. Note that the BHS problem does not have an exact equivalent under the Byzantine black hole assumption. Indeed, if the BBH always acts as an non-malicious node, then it can never be detected by any algorithm. Thus, in this work, our goal is not to locate the Byzantine black hole (BBH), but rather to perpetually explore the network despite its presence. Note that, if all agents are initially co-located and the BBH is a cut-vertex of the network, then it becomes impossible to visit every vertex, as the BBH can block access by behaving as a black hole. Consequently, in this work, we modify the exploration problem to focus on perpetually exploring *one connected component* of the graph, among the connected components which would be obtained if the BBH and all its incident edges was removed from the network. In [13], the problem was first introduced as PerpExploration-BBH for a ring network, where it was studied under various communication models and also under any initial deployment. In this work, in addition to investigating PerpExploration-BBH, we focus on a variant in which agents are required to perpetually explore specifically the connected component that includes their initial position, referred to as the *home*. We call this variant PerpExploration-BBH-Home. PerpExploration-BBH-Home is particularly relevant in practical scenarios where the starting vertex serves as a central base, for example to aggregate the information collected by individual agents or as a charging station for energy-constrained agents. Therefore, during perpetual exploration, it is essential to ensure that the component being explored by the agents includes their *home*. Note that the two variants are equivalent if the BBH is not a cut vertex.

1.1 Related work

Exploration of underlying topology by mobile agents in presence of a malicious host (also termed as black hole) has been well studied in literature. This problem, also known as black hole search or BHS problem was first introduced by Dobrev et al. [9]. Detailed surveys of

the problems related to black holes can be found in [16, 17, 19]. Královič and Miklík [15, 18] were the first to introduce some variants of the classical black hole, among which the *gray hole*, a Byzantine version of the black hole (controlled by the adversary) with the ability, in each round, to behave either as a regular node or as a black hole. They considered the model, in which each agents are controlled by an asynchronous scheduler, where the underlying network is a ring with each node containing a whiteboard (i.e., each node can store some data, with which the agents can communicate among themselves). In this model, one of the problems they solved is Periodic Data Retrieval in presence of a gray hole with 9 agents. Later, Bampas et al. [1] improved the results significantly under the same model. For the gray hole case, they obtained an optimal periodic data retrieval strategy with 4 agents. It may be noted that BBH behavior coincides with the "gray hole" considered in these papers. Moreover, periodic data retrieval is similar to perpetual exploration, as the main source of difficulty lies in periodically visiting all non-malicious nodes in order to collect the generated data. Goswami et al. [13] first studied the perpetual exploration problem in presence of BBH (i.e., PerpExploration-BBH) in ring networks, where the agents are controlled by a synchronous scheduler. They studied this problem under different communication models (i.e., *face-to-face*, *pebble* and *whiteboard*, where *face-to-face* is the weakest and *whiteboard* is the strongest model of communication), for different initial positions of the agents, i.e., for the case when the agents are initially co-located and for the case when they are initially scattered. Specifically for the case where the agents communicate *face-to-face* and they are initially co-located at a node, they obtained an upper bound of 5 agents and a lower bound of 3 agents for PerpExploration-BBH.

1.2 Our contributions

We study the PerpExploration-BBH and PerpExploration-BBH-Home problems in specific topologies and in arbitrary synchronous connected networks with at most one BBH under the *face-to-face* communication model (i.e., two agents can only communicate if both of them are on the same node). To the best of our knowledge, all previous papers in the literature which study variants of the classical black hole, do so assuming that the underlying network is a ring. Our primary optimization objective is the number of agents required to solve these problems.

For tree networks, we provide a tight bound on the number of agents for both problems. Specifically, we show that 4 agents are necessary and sufficient for PerpExploration-BBH in trees, and that 6 agents are necessary and sufficient for PerpExploration-BBH-Home in trees. Our algorithms work without initial knowledge of the size n of the network. However, knowledge of n would not reduce the number of agents, as our lower bounds do not assume that n is unknown. Note that our 4-agent PerpExploration-BBH algorithm can be used to directly improve the 5-agent algorithm for rings, with knowledge of the network size, given in [13] in the same model (*face-to-face* communication, initially co-located agents).

To simplify the presentation, we present these results for path networks, and then we explain how to adapt the algorithms to work in trees with the same number of agents.

In the case of general network topologies, we propose an algorithm that solves the problem PerpExploration-BBH-Home, hence also PerpExploration-BBH, with $3\Delta + 3$ agents, where Δ is the maximum degree of the graph, without knowledge of the size of the graph.

In terms of lower bounds, we first show that, if the BBH behaves as a classical black hole (i.e., if it is activated in every round), then at least $\Delta + 1$ agents are necessary for perpetual exploration, even with knowledge of n. In the underlying graph used in this proof, the BBH is not a cut vertex, hence the lower bound holds even for PerpExploration-BBH. This

can be seen as an analogue, in our model, of the well-known $\Delta + 1$ lower bound for black hole search in asynchronous networks [8]. In passing, under the same assumption of the BBH behaving as a classical black hole, we discuss an algorithm that performs perpetual exploration with only $\Delta + 2$ agents, without knowledge of n.

We then use the full power of the Byzantine black hole to prove a stronger, and more technical, lower bound of $2\Delta - 1$ agents. In this last lower bound, the structure and, indeed, the size of the graph is decided dynamically based on the actions of the algorithm. Hence, the fact that agents do not have initial knowledge of n is crucial in this proof. In this case, the BBH may be a cut vertex, but the adversarial strategy that we define in the proof never allows any agents to visit any other component than the one containing the home node. Therefore, this lower bound also carries over to PerpExploration-BBH.

The above results are the first bounds to be obtained for a more powerful variant of a black hole, for general graphs, under the assumption that the agents have no initial topological knowledge about the network.

Several technical details and proofs are omitted due to lack of space, and they can be found in the full version of the paper.

2 Model and basic definitions

The agents operate in a simple, undirected, connected port-labeled graph $G = (V, E, \lambda)$, where $\lambda = \big(\lambda_v\big)_{v \in V}$ is a collection of port-labeling functions $\lambda_v : E_v \to \{1, \ldots, \delta_v\}$, where E_v is the set of edges incident to node v and δ_v is the degree of v. We denote by n the number of nodes and by Δ the maximum degree of G.

An algorithm is modeled as a deterministic Turing machine. Agents are modeled as instances of an algorithm (i.e., copies of the corresponding deterministic Turing machine) which move in G. Each agent is initially provided with a unique identifier.

The execution of the system proceeds in synchronous rounds. In each round, each agent receives as input the degree of its current node, the local port number through which it arrived at its current node, i.e. $\lambda_v(\{u, v\})$ if it just arrived at node v by traversing edge $\{u, v\}$, or 0 if it did not move in the last round, and the configurations of all agents present at its current node. It then computes the local port label of the edge that it wishes to traverse next (or 0 if it does not wish to move). All agents are activated, compute their next move, and perform their moves in simultaneous steps within a round. We assume that all local computations take the same amount of time and that edge traversals are instantaneous.

Note that we will only consider initial configurations in which all agents are co-located on a node called "the *home*". In this setting, the set of unique agent identifiers becomes common knowledge in the very first round.

At most one of the nodes $\mathfrak{b} \in V$ is a Byzantine black hole. In each round, the adversary may choose to activate the black hole. If the black hole is activated, then it destroys all agents that started the round at $\mathfrak{b}$, as well as all agents that choose to move to $\mathfrak{b}$ in that round. The agents have no information on the position of the Byzantine black hole, except that it is not located at the *home* node. Furthermore, the agents do not have initial knowledge of the size of the graph.

2.1 Problem definition

We define the Perpetual Exploration problem with initially co-located agents in the presence of a Byzantine black hole, hereafter denoted PerpExploration-BBH, as the problem of perpetually exploring at least one of the connected components resulting from the removal of the Byzantine black hole from the graph. If the graph does not contain a Byzantine black hole, then the entire graph must be perpetually explored.

If, in particular, the perpetually explored component *must* be the component containing the *home*, then the corresponding problem is denoted as PerpExploration-BBH-Home.

▶ **Definition 1.** *An* instance *of the PerpExploration-BBH problem is a tuple* $\langle G, k, h, \mathfrak{b}\rangle$, *where* $G = (V, E, \lambda)$ *is a connected port-labeled graph,* $k \geq 1$ *is the number of agents starting on the home* $h \in V$, *and* $\mathfrak{b} \in (V \setminus \{h\}) \cup \{\perp\}$ *is the node that contains the Byzantine black hole. If* $\mathfrak{b} = \perp$, *then* G *does not contain a Byzantine black hole.*

For the following definitions, fix a PerpExploration-BBH instance $I = \langle G, k, h, \mathfrak{b}\rangle$, where $G = (V, E, \lambda)$, and let $\mathcal{A}$ be an algorithm.

▶ **Definition 2.** *We say that an execution of* $\mathcal{A}$ *on* I perpetually explores a subgraph H of G *if every node of* H *is visited by some agent infinitely often.*

▶ **Definition 3.** *Let* $C_1, C_2, \ldots, C_t$ *be the connected components of the graph* $G - \mathfrak{b}$, *resulting from the removal of* $\mathfrak{b}$ *and all its incident edges from* G. *If* $\mathfrak{b} = \perp$, *then* $t = 1$ *and* $C_1 \equiv G$. *Without loss of generality, let* $h \in C_1$.

We say that $\mathcal{A}$ solves PerpExploration-BBH on I, *if for every execution starting from the initial configuration in which* k *agents are co-located at node* h, *at least one of the components* $C_1, C_2, \ldots, C_t$ *is perpetually explored.*

We say that $\mathcal{A}$ solves PerpExploration-BBH-Home on I, *if for every execution starting from the initial configuration in which* k *agents are co-located at node* h, *the component* C_1 *(containing the home) is perpetually explored.*

Finally, we say that $\mathcal{A}$ *solves PerpExploration-BBH with* k_0 *agents* if it solves the problem on any instance with $k \geq k_0$ agents (similarly for PerpExploration-BBH-Home). Note that any algorithm that solves PerpExploration-BBH-Home also solves PerpExploration-BBH.

3 Perpetual exploration in path and tree networks

In this section, our main aim is to establish the following two theorems, giving the optimal number of agents that solve PerpExploration-BBH and PerpExploration-BBH-Home in path graphs.

▶ **Theorem 4.** *4 agents are necessary and sufficient to solve PerpExploration-BBH in path graphs, without initial knowledge of the size of the graph.*

▶ **Theorem 5.** *6 agents are necessary and sufficient to solve PerpExploration-BBH-Home in path graphs, without initial knowledge of the size of the graph.*

For the necessity part, we prove that there exists no algorithm solving PerpExploration-BBH (resp. PerpExploration-BBH-Home) with 3 agents (resp. 5 agents), even assuming knowledge of the size of the graph. For the sufficiency part of Theorem 5, we provide an algorithm, which we call Path_PerpExplore-BBH-Home, solving PerpExploration-BBH-Home with 6 initially co-located agents, even when the size of the path is unknown to the agents. Thereafter, we modify this algorithm to solve PerpExploration-BBH with 4 initially co-located agents, thus establishing the sufficiency part of Theorem 4.

In Section 3.1, we give a brief sketch of the approach we have used to prove the lower bounds on the number of agents. Then, in Section 3.2, we describe the algorithms.

3.1 Lower bounds in paths

▶ **Theorem 6.** *At least 4 agents are necessary to solve* PerpExploration-BBH *in all path graphs with $n \geq 9$ nodes, even assuming initial knowledge of the size of the graph.*

In order to prove Theorem 6, we establish, in a series of lemmas, several different types of configurations from which the adversary can force an algorithm to fail, by destroying all agents or by failing to entirely perpetually explore any component. Omitting a lot of details, these configurations are as follows:

- If one agent remains alive, and at least two BBH locations are consistent with the history of the agent up to that time, then the algorithm fails.
- If two agents remain alive on the same side *of the path* of at least three potential BBH locations, then the algorithm fails. Also, if two agents remain alive on either side of at least three potential BBH locations, then the algorithm fails.
- If three agents remain alive on the same side of at least eight potential BBH locations, of which at least the first three are consecutive nodes in the path, then the algorithm fails.

Theorem 6 then follows by observing that, in the initial configuration with three co-located agents, all nodes apart from the home node are potential BBH positions.

▶ **Theorem 7.** *At least 6 agents are necessary to solve* PerpExploration-BBH-Home *in all path graphs with $n \geq 145$ nodes, even assuming initial knowledge of the size of the graph.*

The first element of the proof of Theorem 7 is that at least one agent must remain at the *home* node up to at least the destruction of the first agent by the BBH (actually, up to the time of destruction of the first agent plus the time it would take for that information to arrive at the home node), otherwise the adversary can trap all agents in the wrong component, i.e., the one not containing the *home* node.

Then, very informally, we argue that, no matter how the algorithm divides the 5 available agents into a group of *exploring* agents that must go arbitrarily far from the home node, and a group of *waiting* agents that must wait at the home node, either the number of *exploring* agents is not enough for them to explore arbitrarily far from the *home* node and, at the same time, be able to detect exactly the BBH position if it is activated, or the number of *waiting* agents is not enough to allow them to recover the information of the exact BBH position from one of the remaining *exploring* agents that has been stranded on the other side of the BBH.

3.2 Description of Algorithm Path_PerpExplore-BBH-Home

Here we first describe the algorithm, assuming that the BBH does not intervene and destroy any agent. Then we describe how the agents behave after the BBH destroys at least one agent (i.e., the BBH intervenes).

We call this algorithm Path_PerpExplore-BBH-Home. Let $\langle P, 6, h, \mathfrak{b} \rangle$ be an instance of PerpExploration-BBH-Home, where $P = (V, E, \lambda)$ is a port-labeled path. Per Definition 3, all agents are initially co-located at h (the home node). To simplify the presentation, we assume that h is an extremity of the path. Our algorithm works with 6 agents. Initially, among them the four least ID agents will start exploring P, while the other two agents will wait at h, for the return of the other agents. We first describe the movement of the four least ID agents say, a_0, a_1, a_2 and a_3, on P. Based on their movement, they identify their role as follows: a_0 as F, a_1 as I_2, a_2 as I_1 and a_3 as L. The identities F, I_2, I_1 and L are denoted as Follower, Intermediate2, Intermediate1 and Leader, respectively. The exploration is performed by these four agents in two steps, in the first step, they form a

particular pattern on P. Then in the second step, they move collaboratively in such a way that the pattern is translated from the previous node to the next node in five rounds. Since the agents do not have the knowledge of n, where $|V| = n$, they do the exploration of P, in $\log n$ phases, and then this repeats. In the i-th phase, the four agents start exploring P by continuously translating the pattern to the next node, starting from h, and moves up to a distance of 2^i from h. Next, it starts exploring backwards in a similar manner, until they reach h. It may be observed that, any phase after $j-$th phase (where $2^j \geq n$), the agents behave in a similar manner, as they behaved in $j-$th phase, i.e., they move up to 2^j distance from h, and then start exploring backwards in a similar manner, until each agent reaches h. Note that, this can be done with agents having $\mathcal{O}(\log n)$ bits of memory, in order to store the current phase number.

We now describe how the set of four agents, i.e., L, I_1, I_2 and F create and translate the pattern to the next node.

Creating pattern. L, I_1, I_2 and F takes part in this step from the very first round of any phase, starting from h. In the first round, L, I_1 and I_2 moves to the next node. Then in the next round only I_2 returns back to *home* to meet with F. Note that, in this configuration the agents L, I_1, I_2 and F are at two adjacent nodes while F and I_2 are together and L and I_1 are together on the same node. We call this particular configuration the *pattern*. We name the exact procedure as Make_Pattern.

Translating pattern. After the pattern is formed in first two rounds of a phase, the agents translate the pattern to the next node until the agent L reaches either at one end of the path graph P, or, reaches a node at a distance 2^i from h in the i-th phase. Let, v_0, v_1, v_2 be three consecutive nodes on P, where, suppose L and I_1 is on v_1 and F and I_2 is on v_0. This translation of the pattern makes sure that after 5 consecutive rounds L and I_1 is on v_2 and F and I_2 is on v_1, thus translating the pattern by one node. We call these 5 consecutive rounds where the pattern translates starting from a set of 2 adjacent nodes, say v_0, v_1, to the next two adjacent nodes, say v_1, v_2, a *sub-phase* in the current phase. The description of the 5 consecutive rounds i.e., a sub-phase without the intervention of the BBH at $\mathfrak{b}$ (such that $\mathfrak{b} \in \{v_0, v_1, v_2\}$) are as follows.

Round 1: L moves to v_2 from v_1.
Round 2: I_2 moves to v_1 from v_0 and L moves to v_1 back from v_2.
Round 3: I_2 moves back to v_0 from v_1 to meet with F. Also, L moves back to v_2 from v_1.
Round 4: F and I_2 moves to v_1 from v_0 together.
Round 5: I_1 moves to v_2 from v_1 to meet with L.

So after completion of round 5 the pattern is translated from nodes v_0, v_1 to v_1, v_2. The pictorial description of the translate pattern is explained in Fig. 1.

Now, suppose v_2 is the node upto which the pattern was supposed to translate at the current phase (or, it can be the end of the path graph also). So, when L visits v_2 for the first time, in round 1 of some sub-phase it knows that it has reached the end of the path graph for the current phase. Then in the same sub-phase at round 2 it conveys this information to I_2 and I_1. In round 3 of the same sub-phase, F gets that information from I_2. So at the end of the current sub-phase all agents has the information that they have explored either one end of the path graph or the node upto which they were supposed to explore in the current phase. In this case, they interchange the roles as follows: the agent which was previously had role L changes role to F, the agent having role F changes it to L, I_1 changes role to I_2 and I_2 changes role to I_1 (refer to Fig. 2). Then from the next sub-phase onwards they

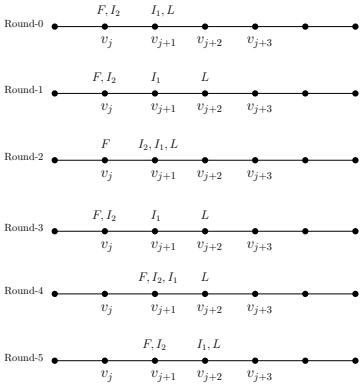

Figure 1 Depicts translating pattern steps.

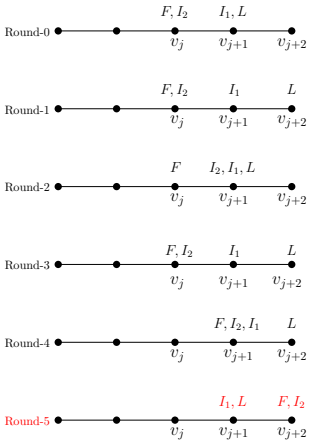

Figure 2 Depicts the step-wise interchange of roles, when the agents reach the end of the path graph, while performing translate pattern.

start translating the pattern towards h. It may be noted that, once L (previously F) reaches h in round 1 of a sub-phase, it conveys this information to the remaining agents in similar manner as described above. So, at round 5 of this current sub-phase, F (previously L) and I_2 (previously I_1) also reaches h, and meets with L (previously F) and I_1 (previously I_2). We name the exact procedure as TRANSLATE_PATTERN.

Intervention by the BBH. Now we describe the behaviour of the agents while the BBH kills at least one exploring agent during create pattern or, during translate pattern. We first claim that, however the BBH intervenes while the four agents are executing creating pattern or translating pattern, there must exists at least one agent, say A_{alive} which knows the exact location of the BBH. The justification of the claim is immediate if we do case by case studies fixing a BBH position and the round it is intervening with the agents. This can be during creating pattern or in a particular sub-phase of translating pattern. Now, there can be two cases:

Case-I: Let $A_{alive} \in C_1$ (C_1 being the connected component of $G - \mathfrak{b}$ that contains h) then it can perpetually explore every vertex in C_1 satisfying the requirement.

Case-II: Let $A_{alive} \in C_2$ (C_2 being the connected component of $G - \mathfrak{b}$ such that $h \notin C_2$). In this case A_{alive} places itself to the adjacent node of the BBH on C_2. Let T_i be the maximum time, required for the 4 agents (i.e., L, I_1, I_2 and F) to return back to h in $i-$th phase if BBH does not intervene. Let us denote the aing agents at h, i.e., a_4, a_5 as F_1, F_2. Starting from the i-th phase, they wait for T_i rounds for the other agents to return. Now, if the set of agents L, I_1, I_2 and F fail to return back to *home* within T_i rounds in phase, i, the agents F_1 and F_2 starts moving *cautiously*. In the cautious move, first F_1 visits the next node and in the next round, returns back to the previous node to meet with F_2, that was waiting there for F_1. If F_1 fails to return then F_2 knows the exact location of $\mathfrak{b}$, which is the next node F_1 visited. In this case, F_2 remains in the component of h, hence can explore it perpetually. Otherwise, if F_1 returns back to F_2, then in the next round both of them moves together to the next node.

It may be noted that, A_{alive} knows which phase is currently going on and so it knows the exact round at which F_1 and F_2 starts moving cautiously. Also, it knows the exact round at which F_1 first visits $\mathfrak{b}$, say at round r. A_{alive} waits till round $r-1$, and at round r it

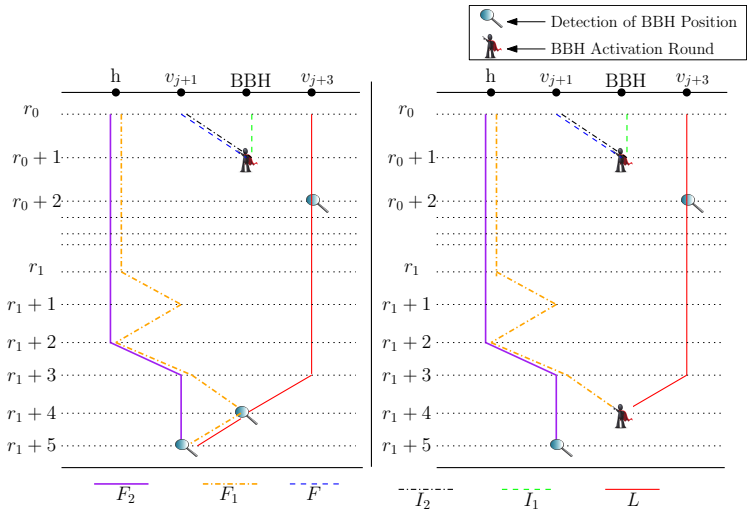

Figure 3 Represents the time diagram, in which at least one among F_1 and F_2 detects $\mathfrak{b}$, and perpetually explores the path graph.

moves to $\mathfrak{b}$. Now at round r, if adversary activates $\mathfrak{b}$, it destroys both F_1 and A_{alive} then, F_1 fails to return back to F_2 in the next round. This way, F_2 knows the exact location of $\mathfrak{b}$, while it remains in C_1. So it can explore C_1 by itself perpetually. The right figure in Fig. 3, represents the case where L detects $\mathfrak{b}$ at round r_0+2, and waits till round r_1+3. In the meantime, at round r_1 (where $r_1 = r'_0 + T_i$, $r'_0 < r_0$ and r'_0 is the first round of phase i), F_1 and F_2 starts moving cautiously. Notably, along this movement, at round r_1+4, F_1 visits $\mathfrak{b}$, and at the same time L as well visits $\mathfrak{b}$ from v_{j+3}. The adversary activates $\mathfrak{b}$, and both gets destroyed. So, at round r_1+5, F_2 finds failure of F_1's return and understands the next node to be $\mathfrak{b}$, while it is present in C_1. Accordingly, it perpetually explores C_1.

On the other hand, if at round r, adversary doesn't activate $\mathfrak{b}$, then F_1 meets with A_{alive} and knows that they are located on the inactivated $\mathfrak{b}$. In this case they move back to C_1 and starts exploring the component C_1, avoiding $\mathfrak{b}$. The left figure in Fig. 3 explores this case, where L detects the position of $\mathfrak{b}$ at round r_0+2, and stays at v_{j+3} until r_1+3, then at round r_1+4, when F_1 is also scheduled to visit $\mathfrak{b}$, L also decides to visit $\mathfrak{b}$. But, in this situation, the adversary does not activate $\mathfrak{b}$ at round r_1+4, so both F_1 and L meets, gets the knowledge from L that they are on $\mathfrak{b}$. In the next round, they move to v_{j+1} which is a node in C_1, where they meet F_2 and shares this information. After which, they perpetually explore C_1.

The correctness of the algorithm follows from the description of the algorithm and the case by case study of all possible interventions by the BBH. We have thus established the sufficiency part of Theorem 5.

▶ Note 8. It may be noted that, our algorithm Path_PerpExplore-BBH-Home, without F_1 and F_2, is sufficient to solve PerpExploration-BBH. It is because, if A_{alive} in C_i ($i \in \{1,2\}$) after one intervention by the BBH, it can perpetually explore C_i without any help from F_1 and F_2 as it knows the exact location of the BBH. This proves the sufficiency part of Theorem 4.

▶ Remark 9. Note that our 4-agent PerpExploration-BBH algorithm for paths directly improves the 5-agent algorithm for rings with knowledge of the network size, given in [13], in the *face-to-face* model with initially co-located agents. Indeed, the 4-agent pattern can

keep moving around the ring as if in a path graph. Naturally, it will never reach a node of degree 1. If and when the BBH destroys an agent, then, as we showed, at least one agent remains alive and knows the position of the BBH. As the agent knows the size of the ring in this case, it can perform perpetual exploration of the non-malicious ring nodes, without ever visiting the BBH.

3.2.1 Modification of the path algorithms to work in trees

We modify the algorithms for path graphs of Section 3.2 to work for trees, with the same number of agents. The algorithms for trees work by translating the pattern from one node to another by following the $k-$`Increasing-DFS` [12] algorithm up to a certain number of nodes in a certain phase.

4 Perpetual exploration in general graphs

In this section, we establish upper and lower bounds on the optimal number of agents required to solve Perpetual Exploration in arbitrary graphs with a BBH, without any initial knowledge about the graph. In particular, we give a lower bound of $2\Delta-1$ agents for PerpExploration-BBH (Theorem 10 below), which carries over directly to PerpExploration-BBH-Home, and an algorithm for PerpExploration-BBH-Home using $3\Delta+3$ agents (Theorem 12 below), which also solves PerpExploration-BBH.

We give a brief sketch of the lower bound proof in Section 4.1, and we present the algorithm in Section 4.2.

4.1 Lower bound in general graphs

For the lower bound, we construct a particular class of graphs in which any algorithm using $2\Delta-2$ agents or less must fail.

▶ **Theorem 10.** *For every $\Delta \geq 4$, there exists a class of graphs $\mathcal{G}$ with maximum degree Δ, such that any algorithm using at most $2\Delta-2$ agents, with no initial knowledge about the graph, fails to solve PerpExploration-BBH in at least one of the graphs in $\mathcal{G}$.*

Proof sketch. For a fixed $\Delta \geq 4$, we construct the corresponding graph class $\mathcal{G}$ by taking an underlying path $\mathcal{P}$, consisting of two types of vertices $\{v_i : 1 \leq i \leq \Delta\}$ and, for each $i \leq \Delta$, the nodes $\{u_j^i : 1 \leq j \leq l_i\}$, where $l_i \geq 1$. The nodes $\{u_j^i\}$ form a subpath of length l_i+1 connecting v_i to v_{i+1}. The nodes v_i (for $\in \{1,2,\ldots,\Delta-1\}$) are special along $\mathcal{P}$, because they are connected to the BBH $\mathfrak{b}$ either directly or via a new node w_i. Every node of $\mathcal{P}$, as well as $\mathfrak{b}$, now completes its degree up to Δ by connecting to at most $\Delta-1$ trees of height 2 (see example in Figure 4). Every node w_i completes its degree up to Δ by connecting to $\Delta-2$ new nodes.

Note that, to reach $\mathfrak{b}$ from a vertex of the form u_j^i, it is required to visit either v_i or v_{i+1}. In the example of Figure 4, we have $\Delta=4$, $l_1=l_2=2$ and $l_3=1$, so in $\mathcal{P}$ there are two vertices u_1^1, u_2^1 between v_1 and v_2, two vertices u_1^2, u_2^2 between v_2 and v_3, and one vertex u_1^3 between v_3 and v_4. In addition, v_2 is directly connected to $\mathfrak{b}$ (or BBH) whereas v_1 and v_3 are connected to $\mathfrak{b}$ via a path of length 2.

Given an algorithm $\mathcal{A}$ which claims to solve PerpExploration-BBH with $2\Delta-2$ co-located agents, the adversary returns a port-labeled graph $G=(V,E,\lambda) \in \mathcal{G}$ in which $\mathcal{A}$ fails, by choosing the lengths l_i and the connection between v_i and $\mathfrak{b}$ (direct or through w_i). A high-level idea of the proof is as follows. First, it can be ensured that, $\mathcal{A}$ must instruct at

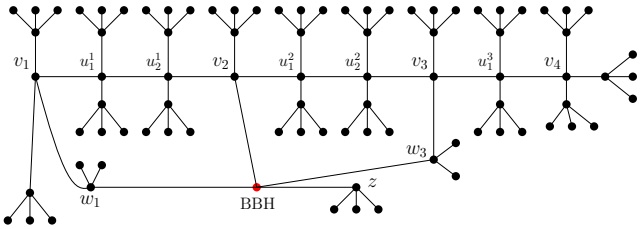

Figure 4 An example graph of the class $\mathcal{G}$, used in the proof of Theorem 10.

least one agent to visit a node which is at 2 hop distance from v_i, for all $i \in \{1, 2, \ldots, \Delta - 1\}$. Let the agent which first visits a node which is at 2 hop distance from v_i, starts from v_i at time t_i. **Case-1**: Now, based on $\mathcal{A}$, if at t_i, at least 2 agents are instructed to move from v_i along the same port, then the adversary chooses a graph from $\mathcal{G}$, such that v_i is connected to $\mathfrak{b}$, and returns a port labeling, such that at $t_i + 1$, these agents reach $\mathfrak{b}$. **Case-2**: Otherwise, the adversary chooses a graph from $\mathcal{G}$ such that, either v_i is connected to $\mathfrak{b}$ or there exists an intermediate vertex w_i connecting v_i with $\mathfrak{b}$. In this case also, the adversary returns the port labeling such that, the agent must move to $\mathfrak{b}$ (or w_i) at $t_i + 1$ and to some vertex z (or $\mathfrak{b}$) at t_i' (where $t_i' > t_i + 1$).

In addition, the graph chosen by the adversary sets the distance between v_{i-1} and v_i in a way to ensure that, within the time interval $[t_i, t_i']$, no agent from any v_α ($\alpha \in \{0, 1, \ldots, i-1\}$) attempts to visit $\mathfrak{b}$. So, for **Case-1**, two agents directly gets destroyed. For **Case-2**, after the first agent is destroyed, remaining agents do not understand which among the next 2 nodes from v_i is $\mathfrak{b}$. Now, for $\mathcal{A}$ to succeed, at least one other agent from v_i gets destroyed, at some round t_i''. This shows that from each v_i, for all $i \in \{1, 2, \ldots, \Delta - 1\}$ at least 2 agents each, are destroyed. Thus, $\mathcal{A}$ fails. ◀

4.2 Description of Algorithm Graph_PerpExplore-BBH-Home

Here we discuss the algorithm, termed as Graph_PerpExplore-BBH-Home that solves PerpExploration-BBH-Home on a general graph, $G = (V, E, \lambda)$. We will show that our algorithm requires at most $3\Delta + 3$ agents. Let $\langle G, 3\Delta + 3, h, \mathfrak{b} \rangle$ be an instance of the problem PerpExploration-BBH-Home, where $G = (V, E, \lambda)$ is a simple port-labeled graph. The structure of our algorithm depends upon four separate algorithms Translate_Pattern along with Make_Pattern (discussed in Section 3.2), Explore (explained in this section) and BFS-Tree-Construction [2]. So, before going in to details of our algorithm that solves PerpExploration-BBH-Home, we recall the idea of BFS-Tree-Construction.

An agent starts from a node $h \in V$ (also termed as *home*), where among all nodes in G, only h is marked. The agent performs breadth-first search (BFS) traversal, while constructing a BFS tree rooted at h. The agent maintains a set of edge-labeled paths, $\mathcal{P} = \{P_v \ : \ \text{edge labeled shortest path from } h \text{ to } v, \forall v \in V \text{ such that the agent has visited } v\}$ while executing the algorithm. During its traversal, whenever the agent visits a node w from a node u, then to check whether the node w already belongs to the current BFS tree of G constructed yet, it traverses each stored edge labeled paths in the set $\mathcal{P}$ from w one after the other, to find if one among them takes it to the marked node h. If yes, then it adds to its map a cross-edge (u, w). Otherwise, it adds to the already constructed BFS tree, the node w, accordingly $\mathcal{P} = \mathcal{P} \cup P_w$ is updated. The underlying data structure of Root_Paths [2] is used to perform these processes. This strategy guarantees as per Proposition 9 of [2], that BFS-Tree-Construction algorithm constructs a map of G, in presence of a marked node, within $\mathcal{O}(n^3\Delta)$ steps and using $\mathcal{O}(n\Delta \log n)$ memory, where $|V| = n$ and Δ is the maximum degree in G.

In our algorithm, we use k agents (in Theorem 12, it is shown that $k = 3\Delta + 3$ agents are sufficient), where they are initially co-located at a node $h \in V$, which is termed as *home*. Initially, at the start our algorithm asks the agents to divide in to three groups, namely, `Marker`, `SG` and $\texttt{LG}_0$, where `SG` (or smaller group) contains the least four ID agents, the highest ID agent among all k agents, denoted as `Marker` stays at h (hence h acts as a marked node), and the remaining $k - 5$ agents are denoted as $\texttt{LG}_0$ (or larger group). During the execution of our algorithm, if at least one member of $\texttt{LG}_0$ detects one port leading to the BBH from one of its neighbor, in that case at least one member of $\texttt{LG}_0$ settles down at that node, acting as an *anchor* blocking that port which leads to the BBH, and then some of the remaining members of $\texttt{LG}_0$ forms $\texttt{LG}_1$. In general, if at least one member of $\texttt{LG}_i$ detects the port leading to the BBH from one of its neighbors, then again at least one member settles down at that node acting as an *anchor* to block that port leading to the BBH, and some of the remaining members of $\texttt{LG}_i$ forms $\texttt{LG}_{i+1}$, such that $|\texttt{LG}_{i+1}| < |\texttt{LG}_i|$. It may be noted that, a member of $\texttt{LG}_i$ only settles at a node v (say) acting as an *anchor*, only if no other *anchor* is already present at v. Also, only if a member of $\texttt{LG}_i$ settles as an *anchor*, then only some of the members of $\texttt{LG}_i$ forms $\texttt{LG}_{i+1}$.

In addition to the groups $\texttt{LG}_0$ and `SG`, the `Marker` agent permanently remains at h. In a high-level the goal of our Graph_PerpExplore-BBH-Home algorithm is to create a situation, where eventually at least one agent blocks, each port of C_1 that leads to the BBH (where $C_1, C_2, \ldots, C_t$ are the connected components of $G - \mathfrak{b}$, such that $h \in C_1$), we term these blocking agents as *anchors*, whereas the remaining alive agents must perpetually explore at least C_1.

Initially from h, the members of `SG` start their movement, and the members of $\texttt{LG}_0$ stays at h until they find that, none of the members of `SG` reach h after a certain number of rounds. Next, we explain one after the other how both these groups move in G.

Movement of `SG`. The members (or agents) in `SG` works in phases, where in each phase the movement of these agents are based on the algorithms Make_Pattern and Translate_Pattern (both of these algorithms are described in Section 3.2). Irrespective of which, the node that they choose to visit during *making pattern* or *translating pattern* is based on the underlying algorithm BFS-Tree-Construction.

More specifically, the i-th phase (for some $i > 0$) is divided in two sub-phases: i_1-th phase and i_2-th phase. In the i_1-th phase, the members of `SG` makes at most 2^i translations, while executing the underlying algorithm BFS-Tree-Construction. Next, in the i_2-th phase, irrespective of their position after the end of i_1-th phase, they start translating back to reach h. After they reach h during the i_2-th phase, they start $(i+1)$-th phase (which has again, $(i+1)_1$ and $(i+1)_2$ sub-phase). Note that, while executing i_1-th phase, if the members of `SG` reach h, in that case they continue executing i_1-th phase. We already know as per Section 3.2, each translation using Translate_Pattern requires 5 rounds and for creating the pattern using Make_Pattern it requires 2 rounds. This concludes that, it requires at most $Ti_j = 5 \cdot 2^i + 2$ rounds to complete i_j-th phase, for each $i > 0$ and $j \in \{1, 2\}$.

If at any point, along their traversal, the adversary activates the BBH, such that it interrupts the movement of `SG`. In that scenario, at least one member of `SG` must remain alive, exactly knowing the position of the BBH from its current node (refer to the discussion of **Intervention by the BBH** in Section 3.2). The agent (or agents) which knows the exact location of the BBH, stays at the node adjacent to the BBH, such that from its current node, it knows the exact port that leads to the BBH, or in other words they act as *anchors* with respect to one port, leading to the BBH. In particular, let us suppose, the agent holds the adjacent node of BBH, with respect to port α from BBH, then this agent is termed as `Anchor(`α`)`.

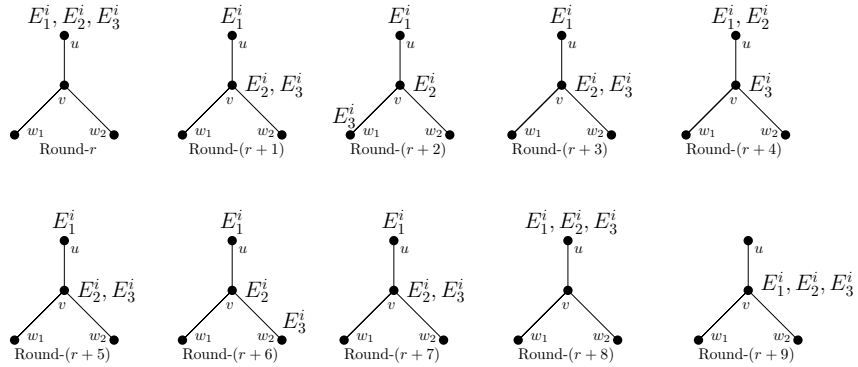

Figure 5 Depicts the round-wise execution of Explore(v) from u by the explorer agents of $\mathtt{LG}_i$, for some $i \geq 0$ on the neighbors w_1 and w_2 of v.

Movement of $\mathtt{LG}_0$. These group members stay at h with `Marker`, until the members of `SG` are returning back to h in the i_2-th phase, for each $i > 0$. If all members of `SG` do not reach h, in the i_2-th phase, i.e., within Ti_2 rounds since the start of i_2-th phase, then the members of $\mathtt{LG}_0$ start their movement.

Starting from h, the underlying movement of the members of $\mathtt{LG}_0$ is similar to BFS-Tree-Construction, but while moving from one node to another they do not execute neither Make_Pattern nor Translate_Pattern, unlike the members of `SG`. In this case, if all members of $\mathtt{LG}_0$ are currently at a node $u \in V$, then three lowest ID members of $\mathtt{LG}_0$ become the explorers, they are termed as E_1^0, E_2^0 and E_3^0 in increasing order of their IDs, respectively. If based on the BFS-Tree-Construction, the next neighbor to be visited by the members of $\mathtt{LG}_0$ is v, where $v \in N(u)$, then the following procedure is performed by the explorers of $\mathtt{LG}_0$, before $\mathtt{LG}_0$ finally decides to visit v.

Suppose at round r (for some $r > 0$), $\mathtt{LG}_0$ members reach u, then at round $r+1$ both E_2^0 and E_3^0 members reach v. Next at round $r+2$, E_3^0 traverses to the first neighbor of v and returns to v at round $r+3$. At round $r+4$, E_2^0 travels to u from v and meets E_1^0 and then at round $r+5$ it returns back to v. This process iterates for each neighbor of v, and finally after each neighbor of v is visited by E_3^0, at round $r + 4 \cdot (\delta_v - 1) + 1$ both E_2^0 and E_3^0 returns back to u. And in the subsequent round each members of $\mathtt{LG}_0$ visit v. The whole process performed by E_1^0, E_2^0 and E_3^0 from u is termed as Explore(v), where v symbolizes the node at which the members of $\mathtt{LG}_0$ choose to visit from a neighbor node u. After the completion of Explore(v), each member of $\mathtt{LG}_0$ (including the explorers) visit v from u. A pictorial description is explained in Fig. 5.

It may be noted that, if the members of $\mathtt{LG}_0$ at the node u, according to the BFS-Tree-Construction algorithm, are slated to visit the neighboring node v, then before executing Explore(v), the members of $\mathtt{LG}_0$ checks if there exists an *anchor* agent blocking that port which leads to v. If that is the case, then $\mathtt{LG}_0$ avoids visiting v from u, and chooses the next neighbor, if such a neighbor exists and no *anchor* agent is blocking that edge. If no such neighbor exists from u to be chosen by $\mathtt{LG}_0$ members, then they backtrack to the parent node of u, and start iterating the same process.

From the above discussion we can have the following remark.

▶ Remark 11. If at some round t, the explorer agents of $\mathtt{LG}_i$ (i.e., E_1^i, E_2^i and E_3^i), are exploring a two length path, say $P = u \to v \to w$, from u, then all members of $\mathtt{LG}_i$ agrees on P at t. This is due to the fact that the agents while executing Explore(v) from u must follow the path $u \to v$ first. Now from v, E_3^i chooses the next port in a particular pre-decided

order (excluding the port through which it entered v). So, whenever it returns back to v to meet E_2^i after visiting a node w, E_2^i knows which port it last visited and which port it will chose next and relay that information back to other agents of $\texttt{LG}_i$ on u. So, after E_2^i returns back to v again from u when E_3^i starts visiting the next port, all other agents know about it.

During the execution of Explore(v) from u, the agent E_3^0 can face one of the following situations:

- It can find an *anchor* agent at v, acting as $\texttt{Anchor}(\beta)$, for some $\beta \in \{1, \ldots, \delta_v\}$. In that case, during its current execution of Explore(v), E_3^0 does not visit the neighbor of v with respect to the port β.
- It can find an *anchor* agent at a neighbor w (say) of v, acting as $\texttt{Anchor}(\beta')$, where $\beta' \in \{1, \ldots, \delta_w\}$. If the port connecting w to v is also β', then E_3^0 understands v is the BBH, and accordingly tries to return to u, along the path $w \to v \to u$, and if it is able to reach to u, then it acts as an *anchor* at u, with respect to the edge (u, v). On the other hand, if port connecting w to v is not β', then E_3^0 continues its execution of Explore(v).

The agent E_2^0 during the execution of Explore(v), can encounter one of the following situations, and accordingly we discuss the consequences that arise due to the situations encountered.

- It can find an *anchor* agent at v where the *anchor* agent is not E_3^0, in which case it continues to execute Explore(v).
- It can find an *anchor* agent at v and finds the *anchor* agent to be E_3^0. In this case, E_2^0 returns back to u, where $\texttt{LG}_1$ is formed, where $\texttt{LG}_1 = \texttt{LG}_0 \setminus \{E_3^0\}$. Next, the members of $\texttt{LG}_1$ start executing the same algorithm from u, with new explorers as E_1^1, E_2^1 and E_3^1.
- E_2^0 can find that E_3^0 fails to return to v from a node w (say), where $w \in N(v)$. In this case, E_2^0 understands w to be the BBH, and it visits u in the next round to inform this to remaining members of $\texttt{LG}_0$ in the next round, and then returns back to v, and becomes $\texttt{Anchor}(\beta)$, where $\beta \in \{1, \ldots, \delta_v\}$ and β is the port for (v, w). On the other hand, $\texttt{LG}_0$ after receiving this information from E_2^0, transforms to $\texttt{LG}_1$ (where $\texttt{LG}_1 = \texttt{LG}_0 \setminus \{E_2^0, E_3^0\}$) and starts executing the same algorithm, with E_1^1, E_2^1 and E_3^1 as new explorers.

Lastly, during the execution of Explore(v) the agent E_1^0 can face the following situation.

- E_2^0 fails to return from v, in this situation E_1^0 becomes $\texttt{Anchor}(\beta)$ at u, where β is the port connecting u to v. Moreover, the remaining members of $\texttt{LG}_0$, i.e., $\texttt{LG}_0 \setminus \{E_1^0, E_2^0, E_3^0\}$ forms $\texttt{LG}_1$ and they start executing the same algorithm from u, with new explorers, namely, E_1^1, E_2^1 and E_3^1, respectively.

For each E_1^0, E_2^0 and E_3^0, if they do not face any of the situations discussed above, then they continue to execute Explore(v).

A pictorial explanation of two scenarios, of how an *anchor* settles at neighbor nodes of BBH is explained in Fig. 6.

Correctness. In order to prove the correctness of the algorithm Graph_PerpExplore-BBH-Home, we give a brief sketch of the proof of the following theorem.

▶ **Theorem 12.** *Algorithm Graph_PerpExplore-BBH-Home solves PerpExploration-BBH-Home in arbitrary graphs with $3\Delta + 3$ agents, having $\mathcal{O}(n\Delta \log n)$ memory, without initial knowledge about the graph.*

Proof sketch. We first show that if $\mathfrak{b}$ (i.e., BBH) never destroys any agent then the problem PerpExploration-BBH-Home is solved only by the four agents in $\texttt{SG}$ (follows from the correctness of BFS-Tree-Construction). On the otherhand if BBH destroys any

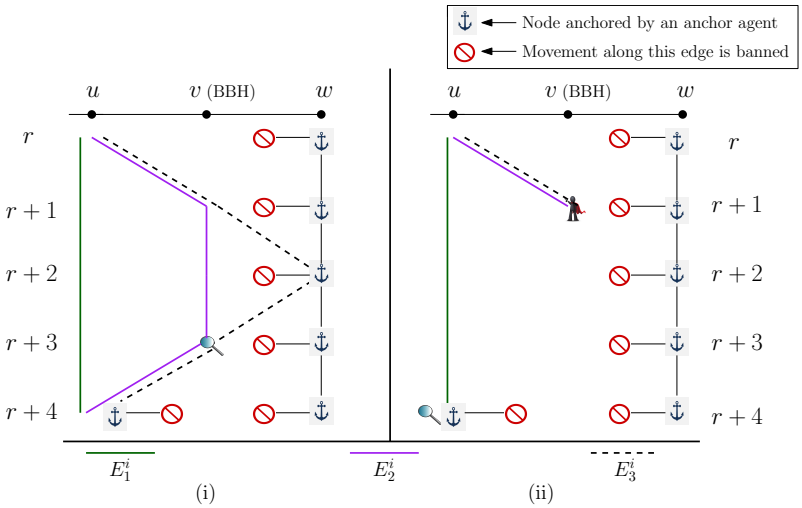

Figure 6 Depicts the time diagram of EXPLORE(v) of $\texttt{LG}_i$ along a specific path $P = u \to v \to w$, where $v = \mathfrak{b}$, in which w contains an *anchor* agent for the edge (v, w). In (i), it depicts even if $\mathfrak{b}$ is not activated, an explorer agent settles as an *anchor* at u for the edge (u, v). In (ii), it depicts that activation of $\mathfrak{b}$ destroys both E_2^i and E_3^i, then an explorer E_1^i gets settled as *anchor* at u for (u, v).

agent, then we prove that there exists an agent from $\texttt{SG}$ acting as an *anchor* at a node in $N(\mathfrak{b})$, where $N(\mathfrak{b})$ indicates the neighbors of $\mathfrak{b}$. The node can be either in C_1 or C_j, where $G - \mathfrak{b} = C_1 \cup \cdots \cup C_t$ for some $t \geq 1$ and *home* $\in C_1$ and $j \neq 1$. Now if we assume, BBH destroys at least one agent then, we define a set $\mathcal{U}$ of nodes in G, where $\mathcal{U} \subseteq N(\mathfrak{b}) \cap C_1$ such that no vertex in $\mathcal{U}$ contains an *anchor*. A node $u \in \mathcal{U}$ ceases to exist in $\mathcal{U}$, whenever an agent visits u and blocks the edge $(u, \mathfrak{b})$, by becoming an *anchor*. We show that during the execution of our algorithm, $\mathcal{U}$ eventually becomes empty. In addition to that, we also show that, the agents in $\texttt{LG}_i$ (for some $i \geq 0$) never visit a node which is not in C_1. By $\texttt{LG}_i$ visiting a node, we mean to say that all agents are located at that node simultaneously. This concludes that, all neighbors in C_1 of $\mathfrak{b}$, gets anchored by an *anchor* agent. So, eventually all the agents, which are neither an *anchor* nor a `Marker`, can perpetually explore C_1. Also, to set up an *anchor* at each neighbor of $\mathfrak{b}$, at most 2 agents are destroyed. So, this shows that $3(\Delta - 1)$ agents are required to *anchor* each neighbor of C_1 by the members of $\texttt{LG}_i$ (for any $i \geq 0$). So, in total $3\Delta + 3$ agents are sufficient to execute GRAPH_PERPEXPLORE-BBH-HOME, including `Marker`, 4 agents in $\texttt{SG}_0$, and one agent which is neither a `Marker` nor an *anchor*, i.e., the one which perpetually explores C_1. Moreover, to execute BFS-TREE-CONSTRUCTION as stated in [2, Proposition 9], each agent requires $\mathcal{O}(n\Delta \log n)$ memory. This concludes the proof. ◀

4.3 Perpetual exploration in presence of a black hole

In the special case in which the Byzantine black hole is activated in each round, i.e., behaves as a classical black hole, we show that the optimal number of agents for perpetual exploration (we call this problem PERPEXPLORATION-BH) drops drastically to between $\Delta + 1$ and $\Delta + 2$ agents.

The lower bound holds even with initial knowledge of n, and even if we assume that agents have knowledge of the structure of the graph, minus the position of the black hole and the local port labelings at nodes. This can be seen as an analogue, in our model, of the well-known $\Delta + 1$ lower bound for black hole search in asynchronous networks [8].

Details of the proof and a discussion of the algorithm that solves the problem with $\Delta + 2$ agents can be found in the full version.

5 Conclusion

We gave the first non-trivial upper and lower bounds on the optimal number of agents for perpetual exploration, in presence of at most one Byzantine black hole in general, unknown graphs.

One noteworthy point, as regards the related problem of *pinpointing* the location of the malicious node, is that all our algorithms have the property that, even in an execution in which the Byzantine black hole destroys only one agent, all remaining agents manage to determine exactly the position of the malicious node, at least within the component which ends up being perpetually explored. By contrast, this is not the case for the algorithms proposed in [13] for PerpExploration-BBH in ring networks, as the adversary can time a single activation of the Byzantine black hole, destroying at least one agent, so that the remaining agents manage to perpetually explore the ring, but without ever being able to disambiguate which one of two candidate nodes is the actual malicious node.

A few natural open problems remain. First, close the important gap between $2\Delta - 1$ and $3\Delta + 3$ for the optimal number of agents required for PerpExploration-BBH and PerpExploration-BBH-Home, in general graphs. Second, note that our general graph lower bound of $2\Delta - 1$ holds only for graphs of maximum degree at least 4 (Theorem 10). Could there be a 4-agent algorithm for graphs of maximum degree 3? Third, in the special case of a black hole (or, equivalently, if we assume that the Byzantine black hole is activated in each round), close the gap between $\Delta + 1$ and $\Delta + 2$ agents. Fourth, our 4-agent algorithm in paths induced a direct improvement of the optimal number of agents for perpetual exploration in a ring with known n, under the face-to-face communication model, from 5 agents (which was shown in [13]) to 4 agents. However, it is still unknown whether the optimal number of agents is 3 or 4 in this case.

Finally, it would be interesting to consider stronger or weaker variants of the Byzantine black hole, and explore the tradeoffs between the power of the malicious node and the optimal number of agents for perpetual exploration. Orthogonally to this question, one can consider different agent communication models or an asynchronous scheduler.

References

1 Evangelos Bampas, Nikos Leonardos, Euripides Markou, Aris Pagourtzis, and Matoula Petrolia. Improved periodic data retrieval in asynchronous rings with a faulty host. *Theor. Comput. Sci.*, 608:231–254, 2015. doi:10.1016/J.TCS.2015.09.019.

2 Jérémie Chalopin, Shantanu Das, and Adrian Kosowski. Constructing a map of an anonymous graph: Applications of universal sequences. In *Principles of Distributed Systems: 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings 14*, pages 119–134. Springer, 2010. doi:10.1007/978-3-642-17653-1_10.

3 Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71(2-3):229–242, 2006. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi71-2-3-05.

4 Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability and Computing*, 16(4):595–619, 2007. doi:10.1017/S0963548306008133.

5 Giuseppe Antonio Di Luna, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Black hole search in dynamic rings. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 987–997. IEEE, 2021. doi:10.1109/ICDCS51616.2021.00098.

6 Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Trans. Algorithms*, 11(1):1:1–1:28, 2014. doi:10.1145/2629656.

7 Stefan Dobrev, Paola Flocchini, Rastislav Královič, and Nicola Santoro. Exploring an unknown dangerous graph using tokens. *Theoretical Computer Science*, 472:28–45, 2013. doi:10.1016/J.TCS.2012.11.022.

8 Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocols. *Distributed Comput.*, 19(1):1–35, 2006. doi:10.1007/S00446-006-0154-Y.

9 Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48:67–90, 2007. doi:10.1007/S00453-006-1232-Z.

10 Stefan Dobrev, Nicola Santoro, and Wei Shi. Locating a black hole in an un-oriented ring using tokens: The case of scattered agents. In *European Conference on Parallel Processing*, pages 608–617. Springer, 2007. doi:10.1007/978-3-540-74466-5_64.

11 Paola Flocchini, David Ilcinkas, and Nicola Santoro. Ping pong in dangerous graphs: Optimal black hole search with pebbles. *Algorithmica*, 62:1006–1033, 2012. doi:10.1007/S00453-011-9496-3.

12 Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005. doi:10.1016/J.TCS.2005.07.014.

13 Pritam Goswami, Adri Bhattacharya, Raja Das, and Partha Sarathi Mandal. Perpetual exploration of a ring in presence of byzantine black hole. In Silvia Bonomi, Letterio Galletta, Etienne Rivière, and Valerio Schiavoni, editors, *28th International Conference on Principles of Distributed Systems, OPODIS 2024, December 11-13, 2024, Lucca, Italy*, volume 324 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.OPODIS.2024.17.

14 Wayne A Jansen. Intrusion detection with mobile agents. *Computer Communications*, 25(15):1392–1401, 2002. doi:10.1016/S0140-3664(02)00040-3.

15 Rastislav Královic and Stanislav Miklík. Periodic data retrieval problem in rings containing a malicious host. In Boaz Patt-Shamir and Tínaz Ekim, editors, *Structural Information and Communication Complexity, 17th International Colloquium, SIROCCO 2010, Sirince, Turkey, June 7-11, 2010. Proceedings*, volume 6058 of *Lecture Notes in Computer Science*, pages 157–167. Springer, 2010. doi:10.1007/978-3-642-13284-1_13.

16 Euripides Markou. Identifying hostile nodes in networks using mobile agents. *Bull. EATCS*, 108:93–129, 2012. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/52.

17 Euripides Markou and Wei Shi. Dangerous graphs. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 455–515. Springer, 2019. doi:10.1007/978-3-030-11072-7_18.

18 Stanislav Miklík. *Exploration in faulty networks*. PhD thesis, Comenius University, Bratislava, 2010.

19 Mengfei Peng, Wei Shi, Jean-Pierre Corriveau, Richard Pazzi, and Yang Wang. Black hole search in computer networks: State-of-the-art, challenges and future directions. *Journal of Parallel and Distributed Computing*, 88:1–15, 2016. doi:10.1016/J.JPDC.2015.10.006.

20 Claude E Shannon. Presentation of a maze-solving machine. *Claude Elwood Shannon Collected Papers*, pages 681–687, 1993.