Distributed Algorithms for Treasure Hunt and Variations of Black Hole

Thesis submitted in partial fulfilment of

the requirements for the degree of

Doctor of Philosophy

by

Adri Bhattacharya

(Roll No. 206123002)

under the supervision of

Prof. Partha Sarathi Mandal

to the

Department of Mathematics

Indian Institute of Technology Guwahati

Guwahati - 781039, India

February 2026

Dedicated to Ma, Baba and Swasrita.

CERTIFICATE

This is to certify that this thesis entitled "**Distributed Algorithms for Treasure Hunt and Variations of Black Hole**" is being submitted by Mr. Adri Bhattacharya to the Department of Mathematics, Indian Institute of Technology Guwahati, is a record of bona fide research work under our supervision and is worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.

The results contained in this thesis have not been submitted in part or full to any other university for the award of any degree or diploma.

IIT Guwahati
November 2025

(**Prof. Partha Sarathi Mandal**)
Department of Mathematics
Indian Institute of Technology Guwahati
Guwahati - 781039, Assam, India

ACKNOWLEDGEMENTS

I feel very fortunate to have the blessing, goodwill and help of the individuals who stood behind me and supported me during this journey of PhD.

I feel that mere words of acknowledgement will not do justice to the care, guidance, mentoring, help, and motivation that my guide, Prof. Partha Sarathi Mandal, has bestowed upon me during this journey. There was not a single moment during my tenure as a PhD student when I did not receive his suggestions. He not only guided and helped me during my research journey, but also in my personal life. I really feel fortunate to have him as my supervisor.

I sincerely want to thank Dr. Barun Gorain, my co-author and also one of my mentors. He helped me learn this subject immensely. His contribution to building my knowledge in this subject has been huge. Next, I would like to sincerely thank Prof. Giuseppe F. Italiano. He is not only my co-author, but his vast knowledge has also really helped me solve some challenging problems during this period. I have really learned how to be humble in life, even after achieving everything. Further, I want to express my immense gratitude towards my other co-author Dr. Evangelos Bampas. He has significantly changed my approach to research. His way of perfectionism and focusing on minute details really inspired me to become a better researcher than I really was. I really look up to him. I also want to extend my appreciation towards Pritam Goswami, a senior and a friend, the deep research conversations that I had with him during the years have really kept me updated. I would also like to thank my co-authors, Subhajit Pramanick and Saswata Jana, for their invaluable suggestions. Saswata deserves a special mention, as he did a thorough scrutiny of my thesis.

The thesis has benefited from the suggestions given by my doctoral committee members, Prof. Gautam K. Das, Prof. Ashok Singh Sairam, Prof. K.V. Krishna and Prof. Kalpesh Kapoor. My institute, the Indian Institute of Technology Guwahati, and my department, i.e., Department of Mathematics, have played a significant role in shaping my research by providing a friendly and vibrant research atmosphere. I would like to thank the Council of

Scientific and Industrial Research (CSIR), Grant Number: 09/731(0178)/2020-EMR-I, Ministry of Science and Technology, Government of India, for their financial assistance during this period of PhD. In addition, I would like to acknowledge the Anusandhan National Research Foundation (ANRF) for supporting me financially during my visit to the DISC conference, held in Berlin, Germany, from October 27th to 31st, 2025.

I acknowledge the discussions with Prof. Krishnendu Mukhopadhyaya, Dr. Anisur Rahaman Molla, Dr. Gokarna Sharma, Dr. Kaushik Mandal, Dr. Anjan K. Chakrabarty, Dr. Pratyoosh Kumar, Dr. Shreemoyee Bora, Dr. Shyamoshree Upadhyay, Dr. K.V. Srikanth, Dr. Sukanta Bhattacharjee, Ashish, Tanvir, Debasish Da they directly or indirectly enhanced my research culture. I thank all my friends, seniors and juniors for supporting me and helping me in various ways, especially, I would like to name Gurinder, Manali, Himanshu, Saurabh, Kannan, Bala, Partha, Tanay, Aritra, Aritra Da, Anjishnu Da, Nabajyoti, Sanket Da, Jaspreet, Rajarshree, Nabanita, Devang and Achyuta. I want to specially mention Rohit, who has always been supporting me since my bachelor's days.

On a separate note, I want to mention the immense care, concern and support that Dr. Debarati Mitra has given me, which is unexplainable. She has been one of the fundamental reasons why I feel this campus is my second home.

My didu and mama have always been unwaveringly supportive of me, not only during this PhD journey, but throughout my life. I know that, whatever difficulty I am in, they will always be there to support me and help me get out of it. Last but not least, my parents and Swasrita have been the three pillars of my life. They equally deserve all the appreciation and accolades that I get for this journey. Without them, I am sure this would not have been possible. They inspire me, motivate me, give me strength and support, and everything that I can think of. I cannot thank them enough. I really feel blessed to have them in my life. Finally, I finish by thanking all my well-wishers.

IIT Guwahati (Adri Bhattacharya)

ABSTRACT

In the last decade, designing distributed algorithms for mobile entities (such as mobile agents) has garnered a lot of interest. There are many fundamental problems in this domain, among them our focus in this thesis has been on *search* and *exploration* problems. The thesis diversifies from an underlying topology being a continuous domain (such as the Euclidean plane) to a discrete domain (i.e., a graph network). Under the graph networks, the thesis focuses on both static as well as dynamic graphs. The first two problems broadly fall under the class of *search* problems.

In the first problem, we study the treasure hunt problem, where the mobile agent is required to find an inert target (or treasure) in an unknown environment. We study this problem in the Euclidean plane, where a mobile agent finds the treasure with the help of pebbles. The treasure is situated at a distance of at most D (> 0) from the initial position of the agent. An Oracle, knowing the treasure's position and the agent's initial location, places some pebbles to guide the agent towards the treasure. The agent has no knowledge about the treasure's position or how many pebbles are placed and where they are placed. Here, in this problem, we raise the following question and answer it: "For given $k \geq 0$, what is the most efficient treasure hunt algorithm if, at most, k pebbles are placed by the Oracle?"

In the second problem, we study the black hole search problem by a team of mobile agents. A black hole is a dangerous stationary node in a graph that eliminates any visiting agent without leaving any trace of its existence. Key parameters that dictate the complexity of finding the black hole include the number of agents required to locate the black hole, the number of moves performed by the agents and the time taken to determine the black hole location. We study this problem when the underlying topology is a dynamic cactus. We introduce two models of dynamicity: we examine the scenario when the underlying graph has at most one dynamic edge, and secondly, when the underlying graph can have at most k dynamic edges. In both these cases of dynamicity, the only constraint is that the underlying graph must be connected, irrespective of which edge (or edges) are dynamic. The choice of edge (or edges) to be dynamic at a certain round is determined by the adversary.

For each of these two cases, we have established lower and upper bounds on the number of agents, moves and rounds required to find the black hole.

The following two problems fall under the class of *exploration* problem. In both of these problems, we study perpetual exploration in the presence of a malicious node that is more powerful than a black hole, in terms of maliciousness. We call this version of a more powerful black hole a *Byzantine black hole* or, in short, a BBH. In our third problem, we study perpetual exploration of a static synchronous ring in the presence of a BBH. We investigated this problem for any arbitrary starting configuration of the agents (i.e., the agents can be either co-located or scattered). Next, for each of these starting configurations, we also looked into various communication models (such as *face-to-face*, *pebble* and *whiteboard*). The main objective in this problem is to emphasize minimizing the number of agents required to guarantee perpetual exploration under all these various conditions and in the presence of a BBH.

In the fourth problem, we answered the following question: "How can a group of initially co-located agents explore an unknown graph, when one stationary node occasionally behaves maliciously, under the control of an adversary?" In other words, we extended our earlier problem (which is confined to just static rings) to any arbitrary topology. Formally, 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 that 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.

Contents

Chapter 1

Introduction

Previously, the term *distributed systems* generally referred to a network of computers, each with its own specific memory, connected by a dedicated network. These computers used to be similar (or homogeneous) with the same kind of processors and operating system. The connection between hosts and these computers used to be reliable as well. However, these networks have undergone significant evolution over time, thanks to the rapid growth of internet connectivity. Nowadays, internet connectivity has become part of our lives, and with it, two terms have become common in network connectivity: *pervasive computing* and *nomadic computing*. Pervasive computing means that everything is now considered a node in the underlying network, whether it is a computer, mobile device, or any other appliance. Nomadic computing means that these nodes are not static or fixed in one place; they move. With all these advancements come challenges as well. For example, the previous client-server model used in distributed systems cannot solve these huge advances. To tackle these challenges, the concept of *mobile agents* has arisen. These are codes that have the ability to migrate from one node to another while performing the assigned task. They have the ability to communicate with other agents as well as the host node. These agents can be thought of as web crawlers, viruses, mobile code, and other similar entities. David Wall [123] in his pioneering work, visualised mobile agents as autonomous entities, operating on graphs. This visualisation is an extension of the earlier message-passing network setting. The hardware (or physical) agents are termed as *mobile robots*. Unlike mobile

agents, which operate in a networked environment, mobile robots predominantly operate in a physical environment. These multi-agent systems study how a group of hardware agents (i.e., mobile robots) or software agents collaboratively execute the tasks to achieve a common goal.

From an algorithmic perspective, the focus has been on developing efficient and robust strategies for these agents or robots to perform complex tasks. Some of these fundamental tasks are *search* [17], *exploration* [121], *rendezvous* [121], *gathering* [2], *flocking* [126], *patrolling* [74] and *dispersion* [105], etc. A comparative survey of all well-known problems and their surveys in both the mobile agent and mobile robot domains can be found in [69]. The thesis focuses on some different varieties of the two fundamental problems, *search* and *exploration*. The objective of *search* problem is to search for a target in the underlying topology (may be a continuous or discrete topology) with the help of a mobile agent (or agents). On the other hand, the *exploration* problem is predominantly focused on the underlying topology to be a graph network. Here, the objective of the mobile agent is to visit each node in the network at least once. However, before delving into the details of the problems studied in the thesis, we discuss the conventional models in the following section that align with this thesis work and are used in the domain of mobile agents.

1.1 Conventional Models

Here, we present some of the different model attributes commonly used for mobile agents or mobile robots, as well as those that also intersect with the thesis's interests. We start with *underlying topology*, followed by *computational model*.

1.1.1 Underlying Topology

Mobile agents generally work in discrete domains, i.e., networks, so the underlying network is characterised to be a graph network. On the other hand, hardware agents (or mobile robots) traditionally work on continuous domains, i.e., Euclidean planes, lines, m-rays, or geometric terrains.

Accordingly, we discuss about discrete and continuous domains.

Continuous Domain

The underlying topology, where the mobile robot operates can be on infinite line [7], polygons [66], rays [8], disk [40], euclidean plane [29], and further extending to triangles [11,44], squares [44] and l_p balls [73]. The robots operating in these domains are generally considered to be point [110] or fat [2] objects.

Discrete Domain

The underlying topology in this case is considered to be a graph network. It is generally considered to be a finite, simple, undirected (or directed) graph, with $G = (V, E)$, where V denotes the set of nodes and E indicates the set of edges. There are also studies that focus on directed graphs. The nodes of G are either labeled or unlabeled. The incident edges from a node are labeled, i.e., it indicates the total order on the set of incident edges from a particular node. These orderings are independent with respect to any node, i.e., the ordering of one node does not depend on the ordering of another node. The nodes and edges remain stationary over time.

Now, since networks are predominantly dynamic in nature, researchers have begun to investigate networks that are not static. Technically, the time is a discrete entity, where it is measured in terms of *rounds*. A dynamic graph is defined to be an *evolving graph*, where an *evolving graph* $\mathcal{G}$ is a collection of a sequence of static graphs, $G_0, G_1, \ldots, G_r, \ldots$. The graph $G_i = (V, E_i)$ indicates the static graph at round i, where $G_0 = (V, E_0)$ indicates the initial graph at round 0. We have $E_0 = \cup_{i=1}^{\infty} E_i$, since no dynamicity is defined for round 0, and hence $E_i \subseteq E_0$. In this thesis, our work has revolved around both static and dynamic networks.

Connectivity is an important issue in dynamic graphs. Various connectivity models of dynamic graphs have been studied in the literature. We define some of the relevant models, which are either used in the thesis or are of future relevance to the problems studied here.

- *T-interval connectivity [96]:* The dynamic graph $\mathcal{G}$ is said to be T-interval connected

for $T \geq 1$, if for all $t \in \mathbb{N} \cup \{0\}$, the static graph $G_{t,T} = (V, \cap_{i=t}^{t+T-1} E(i))$ is connected, where $E(i)$ indicates the set of edges present in the static graph $G_{i,T}$ at time i.

- *Connectivity Time [100]:* The connectivity time of a dynamic graph $\mathcal{G}$ indicates the minimum time T (where, $T \in \mathbb{N}$), such that for all $t \in \mathbb{N} \cup \{0\}$, the static graph $G_{t,T} = (V, \cup_{i=t}^{t+T-1} E(i))$ is connected.

- *T-path connectivity [114]:* A dynamic graph $\mathcal{G}$ is said to be T-path connected for $T \geq 1$, if for all $t \in \mathbb{N} \cup \{0\}$, and for any two nodes $u, v \in V$, there exists at least one round $i \in [t, t+T-1]$, for which there exists a path from u to v in G_i.

Note that, in this thesis, we focus only on dynamic graphs that maintain 1-*interval connectivity*, i.e., a special case of T-interval connectivity, where $T = 1$.

1.1.2 Computational Model

The aim of distributed algorithms for mobile agents or mobile robots is to create decentralisation such that these entities can execute their tasks autonomously. In order to decentralise them, certain limitations on agents (such as opaqueness, visibility, communication type, memory, etc.), on graph nodes (fault type), and scheduler types need to be assumed. Here, in this section, we discuss some of these limitations in detail.

Opaqueness

This idea is valid only for mobile robots. The mobile robots can either be transparent [116] or opaque [110]. In the transparent model, there is no obstruction as one robot can see through another robot. In an opaque model, the vision of one robot can be affected due to the presence of another robot lying in the line of sight.

Visibility

The concept of visibility in a discrete domain differs from that in a continuous domain. In the discrete domain, i.e., in graph networks, there are mainly two varieties: zero-hop visibility [75] and l-hop visibility [1]. In zero-hop visibility, agents present at a node in

the underlying graph can only see the current node itself. In other words, it can see the presence of all the mobile agents at the current node, and it can also identify all the port numbers associated with it. In l-hop visibility, agents can see other mobile agents and the ports associated with all nodes located within l hops of the current node of the agent. If l equals the diameter of the underlying graph, it implies that the agent has full visibility.

In the continuous domain, there are variations in the visibility model based on the agent (or robot) model. The robots may have limited visibility [125] or infinite visibility [110]. An opaque robot may create an obstruction in visibility, because two robots, r_i and r_j, cannot see each other if there exists a third robot, r_k, along the line segment joining r_i and r_j. This phenomenon is termed *obstructed visibility*. So, we reformulate the visibility as the robots can see all the unobstructed robots present inside their visibility range.

Schedulers

The notion of *time* is an important entity. Even if these agents or robots are bestowed with some local clock, these clocks may not be synchronised. To execute a cycle of movement and computation, agents (or robots) may not have uniform execution times. Moreover, the *delay* between any two cycles of the same agent (or robot) might not be the same, and it varies with different agents (or robots). So, to define the *activation* schedule and timing of the *operations*, certain models are defined.

Operation Cycle: Each cycle comprises of Look-Compute-Move phase (termed as LCM cycle).

- *Look:* The mobile agent or robot takes a snapshot of its surroundings, based on the visibility that it has. This snapshot may also include the communications it may have made with other agents or the data it has collected from its current position.
- *Compute:* It runs the algorithm, based on the information it gathered during the *look* phase, and it decides either to stay at the current position or move to some target position.

- *Move:* It performs the action (move or stay) that it has decided during the *compute* phase.

Activation Schedule: There are three models of activation scheduler, based on which either mobile agents or mobile robots are activated.

- *Fully-Synchronous (FSYNC):* Time is divided into global rounds, wherein each round, the agents get activated simultaneously and perform the LCM cycle in synchrony. A visual representation is shown in Fig. 1.1(a).
- *Semi-Synchronous (SSYNC):* This model is similar to the FSYNC model; the only difference is that not all agents need to be activated at each round; only a subset of agents can be activated. The agents that are activated perform the LCM cycle in synchrony. Also, each agents are activated infinitely often. Refer to Fig. 1.1(b).
- *Asynchronous (ASYNC):* This is the most general model, where the agents (or robots) have no common notion of time. Any agent (or robot) can be activated at any point in time. The time required to perform Look, Compute, Move, or inactive states varies for each agent. The time required to perform each of these operations takes finite but unbounded time. Refer to Fig. 1.1(c).

Communication Protocol

Inter-agent communication and interaction are among the most important protocols to deal with. Several models of these communication protocols have been studied in the literature, each with its own distinct characteristics. We discuss some of these protocols.

- *Global space*: This is the most powerful model of communication. There exists a globally shared memory, where agents can read information concurrently from their respective positions, whereas writing data must be done in a mutually exclusive manner.

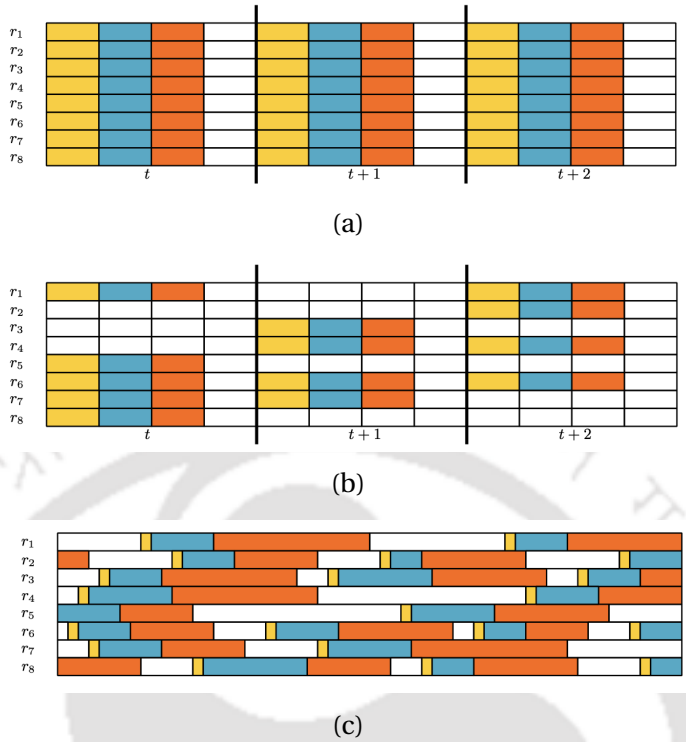


Figure 1.1: The three main activation scheduler models as indicated in [69]. Yellow indicates Look, blue indicates Compute, red indicates Move and white indicates inactive state.

- *Whiteboard*: This model is less powerful than the *global space* model. Here, each node in the network contains a locally shared memory, which is referred to as the *whiteboard*. Any agent visiting a node in the network, say u, can read or write or modify any data from the whiteboard at u. The agents can access the information on the whiteboard in a fairly mutually exclusive manner, i.e., concurrent access is not permitted.

- *Pebble or Token*: The token (or also called pebble) model is a very classical idea, initially introduced for the problems related to graph explorations and rendezvous [15, 20]. These are identical objects that can be placed at a node, picked from a node and also can be carried to other nodes by the mobile agents. In this model, agents can only communicate using these tokens; there is no other means of communication available. Further, there are certain variations of the token model. A more powerful version is the *enhanced token* model, in which the tokens can be placed at a node to mark certain adjacent ports of the node.

There is a simple relationship between the *whiteboard* and the *token* model. If any algorithm uses at most l tokens to execute the algorithm, then the same algorithm can be executed in the *whiteboard* model, where the *whiteboards* are of size $\lceil \log l \rceil$ bits.

- *Face-To-Face*: It is a simple yet very weak model of communication. Here, the agents present at a node can communicate with one another. Communications between these agents occur among themselves, where an agent reads the memory of another agent, and vice versa. This model is normally used when the scheduler is *fully-synchronous*.

Memory

An agent's memory is an important entity, as the amount of persistent memory it can store directly indicates the number of tasks it has the ability to perform. The persistent memory indicates the quantity of information that the agent can carry from the end of the earlier stage to the beginning of the new stage. Based on these capabilities, we have certain variations of agents. The agents can be *oblivious*, i.e., they can have no persistent memory. The agents can have bounded memory, i.e., of $O(1)$ bits; these types of agents are called *finite-state* mobile agents. The bounded memory depends on some graph parameters, such as the cardinality of the underlying graph network. The agents may also have unbounded memory; these agents can store any amount of information. In this thesis, we have used agents with both bounded and unbounded memory.

Knowledge

The agents may require certain initial conditions to execute their algorithm; otherwise, it may be impossible to execute the algorithm successfully. We describe this phenomenon with an example: in an unknown and unlabeled graph network, it is not possible for any agent to visit every node of the network under the face-to-face model of communication. It is because, in the worst case the agent's may loop in a cycle of the underlying graph, infinitely. We provide a brief overview of the type of knowledge typically provided to agents.

- *Number of nodes in the underlying graph network* - a precise knowledge of the number of nodes (n) or an upper bound on the number of nodes is provided.
- *Number of edges in the underlying graph network* - a precise knowledge or upper bound on the number of edges is provided. One can deduce the knowledge of n from this, in any connected graph.
- *Number of agents executing the algorithm.*
- *Underlying topology* - the agents may be provided with the information about its underlying topology, this helps to execute specific protocols depending on the family of graph that it belongs, i.e., trees, rings, cactuses, tori, etc.
- *Number of pebbles or tokens placed in the underlying topology* - the knowledge about the total number of tokens or pebbles that can be used or are already placed.
- *Map of the underlying graph network* - here, the agent is provided with a copy of the underlying port-labelled network, with a special marked node depicting the agent's current location.
- *Full knowledge of the underlying graph network + positional knowledge of agents* - here, the agent not only gets a map of the network, but also gets the knowledge about the positions of other agents inside the map.

Faults of Agents

The faults of agents can be classified into two types: *external agents* or *internal agents.*

- *External Agents*: These agents are normally intruders or viruses. The sole purpose of these types of agents is to infect or damage nodes in the underlying network.
- *Internal Agents*: Normally, internal agents are thought to be correct. But during the execution of assigned protocols, they may develop a glitch, which in turn makes them harmful. This glitch renders faulty behaviour. Mainly, two main varieties exist in literature: *crash faults* (where an agent stops executing its protocol) and *byzantine*

faults (where the agent may behave arbitrarily). It is obvious that a byzantine fault is more powerful than a crash fault. There are two variations of byzantine fault, which are: *weak byzantine* and *strong byzantine*. In a weak byzantine model, agents cannot alter their IDs, whereas in a strong byzantine model the agents can also fake their IDs.

Faults of Nodes

There are certain faults that can occur on the nodes of the network as well. We discuss some of them.

- *Black Hole* [59]: It is a stationary node that has the capability to destroy any agent without leaving any trace of its existence. Similarly, one can define *black edges* as the edges in the underlying network that have the capability to destroy any agent without leaving any trace of its existence.

- *Black$^+$ Hole* [13]: It is a stationary node that acts similarly to a black hole, but from time t_j onwards, where j denotes the j-th round. This time, t_j (or precisely the j) is chosen by the adversary. If $j = \infty$, then the node is a completely safe node.

- *Gray Hole* [88]: It is a stationary node that sometimes behaves as a normal node and sometimes as a black hole, where these changes are completely adversarial. This unique characteristic makes it difficult to detect its location. Trivially, it is a more powerful version of a black and black$^+$ hole.

- *Byzantine Black Hole (BBH)* [76]: In addition to the behaviours of *gray* hole, this variation of black hole also has the ability to choose to destroy any information stored at that node (mainly the case for pebble or whiteboard model of communication). Trivially, this model is more powerful than *gray* hole in terms of malicious behaviour.

  In Chapter 5, we introduce the concept of BBH.

- *Gray$^+$ Hole* [13]: In this variation, the *Gray$^+$ hole* not only has the power of *gray hole*, but it also has the power to do many other malicious behaviours. It can hinder the

fairness by neglecting some agents from its queue forever. It can send the agent to a node different from the one it is intended to move to, or it can send an agent to a neighbouring node even if it has not requested to do so, or it can fake its own node ID (if it exists) to the agents present at that node. It may not respect the queue order of accessing the whiteboard in that particular node. Finally, it also has the capability to create multiple fake copies of the agents present at that node.

- *Red Hole* [13]: In addition to *gray*$^+$ hole, it has the ability to alter the run-time environment (i.e., by changing the whiteboard information). *Red* hole is more powerful than *gray*$^+$ hole.

- *Black Virus* [33]: It is a different model in comparison to *black hole*. Here, the black virus is also a stationary node that has the capability of destroying any visiting agent, but during destruction, it also spreads its clone along all neighboring nodes, where there exist no other agents.

1.1.3 Common Problems

We discuss some of the fundamental problems in this domain.

- **Rendezvous or Gathering**: If the aim is to make two agents meet in an unknown environment (may be in a discrete or continuous domain), then the problem is termed as *rendezvous*, whereas if the task is to make more than two agents meet, then it is called *gathering*.

- **Searching**: The task for a single agent or a group of agents is to search for an entity in an unknown environment, where it may be a discrete domain or a continuous domain.

- **Black Hole Search**: This problem is specifically designed for graph networks, where there exists a malicious node; the aim of the agents is to locate its position.

- **Exploration**: The task for a single or a group of agents is to visit each node of the underlying network at least once.

- **Perpetual Exploration**: It is a special version of the exploration problem, where the aim is to visit each node in the network infinitely often.
- **Dispersion**: Starting from any arbitrary position, the aim is to create an arrangement where each node in the network contains at most one agent.

1.2 Scope of Thesis

In Chapter 2, a detailed discussion of the related works is presented. Chapter 3 discuss about the treasure hunt problem in the Euclidean plane with mobile agents in the presence of pebbles. Chapter 4 addresses the black hole search problem in the presence of co-located mobile agents, for the dynamic cactus graph. Chapter 5 and 6 investigate the problem of perpetual exploration in the presence of a Byzantine black hole (also termed as BBH) with mobile agents, when the underlying topology is a ring and any arbitrary graph, respectively. Lastly, the thesis concludes in Chapter 7. In the following section, we give a brief overview of the results obtained in each chapter.

1.2.1 Treasure Hunt in Euclidean plane

In Chapter 3, we introduce the problem of Treasure Hunt in the presence of pebbles in the Euclidean plane. The idea behind this problem is that a mobile agent, starting from a point, must locate the treasure, which is situated at an unknown location in the Euclidean plane. The agent has no control over the speed, so without any reference, it can move to an infinite distance without even locating the treasure. To tackle this issue, an external entity, termed Oracle, which knows the algorithm of the mobile agent, its starting location, and the position of the treasure, places some pebbles. These pebbles are nothing but markers. Now, the Oracle also has a constraint in placing these pebbles. Firstly, it can only place k of them. Secondly, the distance between each pebble must be at least 1. With these notions, we have formulated a trade-off between the cost of finding the treasure vs. the number of pebbles the Oracle can place. We have shown that it is impossible to locate the treasure if the Oracle has only one pebble to place. Next, for different values of k we give different

strategies. Our algorithms highlight that, as we increase k, the cost of finding the treasure reduces.

1.2.2 Black Hole Search in a Dynamic Cactus

In Chapter 4 we consider the problem of black hole search (or BHS). This is a very fundamental problem in this domain. This problem requires the mobile agents to locate (or identify) the location of the black hole. We considered our underlying topology to be a dynamic cactus graph, where a black hole exists, whose position is unknown to the agents. The adversary plays a major role by introducing dynamicity in the underlying cactus graph. It does so by disappearing or reappearing edge(s) from the underlying static graph, considering that at any point in time, the graph is connected. These edges act as dynamic edges. The agents initially start at a single node and are controlled by a synchronous scheduler. The main contributions include two variations of dynamicity. First, the underlying cactus graph can have at most one dynamic edge. In this case, we show that two agents cannot solve the BHS problem. With three agents, we provide upper and lower bound results on the move and round complexity. Second, the underlying cactus graph can have k (> 1) dynamic edges, and in this case, we also provide upper and lower bound results on the number of agents, move, and round complexities.

1.2.3 Perpetual Exploration of a Ring with a Byzantine Black Hole

In Chapter 5, we address the perpetual exploration problem of a ring in the presence of a Byzantine black hole (BBH). A BBH is a more generalised version of the black hole. The agents are controlled by a synchronous scheduler, and our objective is to optimise the number of agents required to perform perpetual exploration in the presence of the BBH. We solve the problem by taking into account various communication models of the agent, such as *whiteboard, pebble* and *face-to-face*, and with different initial configurations of the agent, such as *co-located* and *scattered*.

1.2.4 Perpetual Exploration of Arbitrary Graphs with a Byzantine Black Hole

We revisit the problem of perpetual exploration in the presence of at most one Byzantine black hole (BBH) in Chapter 6. We consider that the agents are initially co-located, they work synchronously and communicate via *face-to-face* model of communication. We propose two variations of this problem. One variation is a specific version of the other variation. In each variation, we find optimal results in terms of the number of agents when the underlying graph is acyclic. Further, we propose upper and lower bounds on the number of agents when the underlying graph is any arbitrary graph. We also highlight the fact that the presence of a BBH makes solving perpetual exploration harder compared to the presence of a black hole.

Chapter 2

Literature Review

In this chapter, our focus is on discussing related works that address fundamental problems closely aligned with the work presented in this thesis. Since all the works fall under two main fundamental problems in the domain of distributed computing with mobile agents, namely the search and exploration problem, we will start with these problems accordingly. Subsequently, we discuss the specific problems in more detail.

2.1 Search and Exploration Problems

Search and Exploration are two of the most fundamental problems in this domain. These problems have various applications in different fields, ranging from operations research to robotics and also in theoretical computer science.

The history of the search problem dates back to the 1940s, where it was introduced by Koopman [86]. Later, Beck and Bellman [17,18] solved many deterministic and probabilistic search problems. Subsequently, numerous surveys and books have narrated the advancement of this search problem [4,6,38,41]. Nowadays, the search problem is more concentrated on calculating the competitive ratio [27], where the competitive ratio is defined as the ratio between the cost of an online algorithm (which does not have full knowledge) and that of an offline optimal algorithm. The search can be performed by a single agent or multiple agents. A classic example of searching by a single entity is the cow-path problem [10], where the problem asks a single agent to find the target optimally, where the target

is a point object. This kind of search problem is defined as *one-dimensional search*. A specific variation of this one-dimensional search problem is the *Treasure Hunt* problem, and this problem has been investigated in Chapter 3. Further, there exists a *two-dimensional search* problem as well, where the target is an infinite line. These kinds of problems are termed *shoreline search*, and they were introduced by Bellman [19]. Furthermore, multi-agent or collaborative search has also garnered significant attention [41, 81]. The ANTS problem [65] is a popular version of the collaborative search problem, where the agents are identical in nature and their task is to collaboratively search for the target. The main parameters of complexity that dominate are the trade-off between the number of agents and the cost of finding the target. There are further extension works on fault-tolerance and multi-agent coordination [25].

Many tasks in the domain of distributed computing with mobile agents require the agents to traverse the graph, i.e., visit each node in the network. This phenomenon of visiting each node in the network is referred to as the *exploration* problem. The exploration problem can have many variations: it may be of the form "exploration with stop", "exploration with return" and "perpetual exploration". In "exploration with stop", the agent (or agents) can stop exploration once they visit each node in the underlying graph network. In "exploration with return", the agent (or agents) may be instructed to return to the starting node. Next, in "perpetual exploration" problems, the agent (or agents) are required to continuously monitor the nodes in the network; in these problems, the agents do not need to terminate, but rather, they are required to visit the nodes infinitely often. The problem of exploration becomes more relevant and challenging when the agents initially have no knowledge of the underlying graph. This problem of exploring unknown graphs by finite automatons (or mobile agents with finite memory), was first introduced by Shannon [117] in the year of 1951, to explore mazes or labyrinths. But exploring mazes or labyrinths is far less complicated than exploring graphs. A popular result by Rollik [112] explains that exploring all graphs is impossible with any finite team of finite-state automata (or agents with finite memory). Fraigniaud et al. [70] proposed that $\Omega(\log n)$ bits (n is the total number of nodes in the graph network) are required to explore all graphs with mobile agents. Further, an exploration algorithm that requires log-space was proposed by Reingold [111].

Next, Disser et al. [58] proposed that even agents with less than logarithmic memory can perform exploration, but then the agents must be provided with $O(\log\log n)$ many distinct pebbles. Panaite and Pelc [107] designed the fastest known exploration algorithm that requires $m+O(n)$ moves, m indicates the number of edges in the underlying graph. There are also various studies involving exploration of specific families of graphs, such as trees [57], rings [24, 97], grids [26, 31, 52], tori [16], hypercubes [67], etc. Trivially, exploring directed graphs is more difficult than exploring undirected graphs, as there might be existence of no path to backtrack. Deng et al. [49] studied the exploration of directed graphs, whereas Bender et al. [20] studied the exploration of unlabeled directed graphs with a single pebble, where the agents only know the size of the graph. There are many variations of the classical exploration problem, which have also been studied in the literature. Bramas et al. [32] studied a new version of this problem, in which mobile agents can share energy among themselves. They considered the underlying topology to be trees, and solved the exploration problem in the presence of asynchronous and synchronous schedulers, where these agents are prone to crash fault. Devismes et al. [51] studied the graph exploration with mobile agent in *distance-constraint* model, i.e., the agent needs to deal with the knowledge of knowing the exact path (or a set of edges) of length D from its current position to its initial position s.

Next, we discuss about some specific problems related to these two fundamental problems.

2.1.1 Dispersion Problem

This problem was first introduced by Augustine and Moses Jr. [9]. The main complexity measures studied by them are the trade-off between the memory requirements of the agents and the time required for the agents to disperse. After this seminal work, there have been a plethora of studies done on the dispersion problem under various parameters [84, 90, 92, 93, 118]. This problem has been investigated for trees [95], grids [93], and arbitrary graphs [91, 95, 119], dynamic rings [1], etc. Further, there are studies focusing on reducing the space complexity by exploring randomized algorithms [46, 105], whereas

some studies focused on achieving dispersion in the presence of fault [37, 104, 108]. Exploration with a single agent is related to the dispersion problem, where, unlike the exploration problem, here only k nodes need to be determined, and for that, k agents are required.

2.1.2 Treasure Hunt Problem

As discussed earlier, a Treasure hunt is a search problem, where the mobile agent needs to find an inert target. The papers by Miller et al. [103] and Ta-Shma et al. [122] are the first to relate the rendezvous and treasure hunt problems in graphs. Many problems in relation to treasure hunt are studied in the book [5], where several of these algorithms are randomized. Kao et al. [83] studied the randomized version of the treasure hunt problem, where the underlying topology is a star with m-rays. The authors in [82, 98] studied the treasure hunt problem, where the treasure is a line located in the plane. Treasure hunt in the Euclidean plane in the advice model is studied by [109]. Bouchard et al. [30], studied how different kinds of initial knowledge impact the cost of treasure hunt in a tree network. Bouchard et al. [28], gave an almost optimal treasure hunt algorithm in a unweighted graph, where they showed that there does not exist any algorithm which works better than $\theta(e(d))$ complexity for all graphs, where $e(d)$ represents the number of edges whose at least one endpoint is located less than d distance from the starting point s. Recently, Angelopoulos [7] studied the problem of searching for a target on line with advice. But in this case, he considered that the advice can either be provided by a trusted source (which is correct) or by an adversary (which is wrong). In this setting, he studied the Pareto efficiency of his search algorithms based on the advice provided.

Next, the three following papers [21, 75, 109] are closest to the work done in Chapter 3. Pelc et al. [109] provided a trade-off between cost and information of solving the treasure hunt problem in the plane. They achieved optimal and nearly optimal results across various ranges of vision radius. Gorain et al. [75] presented an optimal treasure hunt algorithm in graphs using pebbles, referred to as advice. In [21], the authors studied a trade-off between the number of pebbles vs. the cost of the treasure hunt algorithm in an undirected

port-labeled graph. We then investigated the trade-off between the number of pebbles vs. the cost of the treasure hunt problem when the underlying topology is the Euclidean plane (refer to Chapter 3), and as far as our knowledge goes, we were the first to do so.

2.1.3 Rendezvous Problem

This problem basically asks two mobile agents starting from different locations to meet at the same location. The problem was first mentioned in the book: *The Strategy of Conflict* [115]. Following this, a rigorous study has been conducted on this problem. The reason for this rigorous study is partly due to its applicability beyond distributed algorithms, such as in game theory. A detailed survey of the rendezvous problem can be found in [3, 5, 89]. In brief, this problem can be categorised into two parts: rendezvous in a graph and rendezvous in a plane. In the graph, Dessmark et al. [50] solved the rendezvous problem under a synchronous scheduler, where the number of rounds required for those two agents to meet using their protocol depends on n (size of graph), l (length of the shorter of the two labels), τ (delay between their wake-up time). They proposed an open problem, whether it is possible to perform rendezvous in polynomial time, where the complexity depends only on n and l. Accordingly, Kowalski and Malinowski [87] proposed a protocol that achieves rendezvous in $O(\log^3 l + n^{15}\log^{12} l)$. Ta-Shma and Zwick [122] improved this complexity to $O(n^5 \log l)$, whereas the lower bound proposed by Dessmark et al. [50] is $\Omega(n \log l)$. Das et al. [47] and Miller and Pelc [102] studied the impact on the rendezvous problem when an external entity (e.g., an Oracle) provides the agent with information in almost every round. Fraigniaud and Pelc [71,72] investigated the minimum amount of memory required by these agents to solve the rendezvous problem. Fault-tolerant rendezvous algorithms are established by Ooshita et al. [106]. There have also been works where the agents are considered to be controlled by an asynchronous scheduler [45].

Rendezvous in a plane has been studied under both synchronous and asynchronous schedulers. Suzuki and Yamashita [120] introduced this problem in a plane under a synchronous scheduler. Numerous studies have examined rendezvous under asynchronous schedulers in various settings [39].

2.1.4 Black Hole Problem

The black hole search problem (BHS) is well-studied in the domain of mobile agents. This problem has been studied under varying underlying topologies, including rings, grids, tori, and arbitrary topologies. The problem was first introduced by Dobrev et al. [59], who solved the BHS problem by considering the underlying topology as an arbitrary network. In this setting, they established tight bounds on the number of agents and calculated the cost of a size-optimal BHS protocol. After this seminal paper, there has been a plethora of work done in this domain under different graph classes such as trees [43], rings [12, 36, 62, 63], tori [35, 99] and in graphs of arbitrary and unknown topology [42, 60]. Mainly, two variations of this problem are studied: first, when the agents are initially co-located [43] and second, when the agents are initially scattered [35, 36, 62] in the underlying network.

Most of these studies have been conducted for static networks; there is limited knowledge about this BHS problem for non-static networks. Researchers have started investigating some of the fundamental problems in this domain for dynamic networks. Notably, the exploration problem has already been studied in dynamic rings [53, 79], dynamic tori [78], dynamic cactuses [80] and in dynamic general graphs [77]. In addition to the exploration problem, other problems regarding mobile agents are also studied in dynamic networks, such as gathering [55], compacting of oblivious agents [48], and dispersion of mobile agents [1, 94] are some of them. Compared to that, the only papers regarding BHS on dynamic networks are as follows [22, 54, 56, 68, 85]. Flocchini et al. [68] studied the BHS problem in *carrier graphs* (specifically termed as *subway* networks), which is a special graph within the class of periodic temporal graphs. They showed that the minimum number of agents required to solve the BHS problem on such a subway network, where an asynchronous scheduler controls the mobile agents, is $\gamma + 1$ (γ is denoted to be the minimum number of carrier stops at black holes). On the other hand, Di Luna et al. [54, 56] explored the BHS problem in a dynamic ring. In [56], they considered that the agents are initially co-located, whereas in [54], they considered the agents are initially scattered arbitrarily along the nodes of the ring (where each such initial node does not contain the black hole). In both papers, they obtained optimal bounds on the number of agents, moves and

round complexities. Next, Bhattacharya et al. [22,23] extended the BHS problem in the dynamic cactus and dynamic torus, in which they gave upper and lower bound results on the number of agents and round complexity required to execute a BHS algorithm. In Chapter 4, a detailed explanation of the work [23] is provided. Recently, Kaur et al. [85] extended this problem to dynamic general graphs. Table 2.1 provides a brief survey of all the BHS problems studied in 1-interval connected graphs.

Dyn. Graph	**IC**	**Prob.**	**Comm.**	**Agent LB**	**Time UB**	**Agent UB**	**Mem.(WB)**
Ring [54,56]	R	1-BHS	F2F	3	$\Theta(n^2)$	3	$O(\log(n))$
			WB	3	$\Theta(n^{1.5})$	3	
	A		F2F	3	Impossible	-	
			Pebble	3	$\Theta(n^2)$	3	
Cactus [Chapter 4]	R	1-BHS	WB & F2F	3	$O(n^2)$	3	$O(\Delta(\log\Delta + k\log k))$
		k-BHS		$k+2$	$O(kn)$	$2k+3$	
Tori [22] (Size $n \times m$)	R	$(n+m)$ -BHS	WB & F2F	$n+2$	$O(nm^{1.5})$	$n+3$	$O(1)$
					$O(nm)$	$n+4$	
	A			$n+3$	$O(nm^{1.5})$	$n+6$	
					$O(nm)$	$n+7$	
Gen. [85]	R	1-BHS	WB & F2F	-	$O(\|E\|^2)$	9	$O(\log(\Delta))$
		k-BHS		$2k+2$	$3\Delta^n(\Delta+1)^{2k+n}$ $(n-1)^{2k}$	$6k$	
	A	1-BHS		$2deg(BH)$ $+1$	-	-	-
		k-BHS		$2k+2$	-	-	-

Table 2.1: Summary of results of BHS in 1-interval connected graphs, where WB indicates Whiteboard, F2F indicates Face-to-Face, Gen. indicates general graph, n is the number of nodes in Ring, Cactus and arbitrary graph, Δ indicates the maximum degree of the underlying graph, IC indicates initial configuration, LB indicates lower bound, UB as upper bound, $deg(BH)$ indicates degree of the black hole node, R and A indicates rooted and arbitrary initial configuration, respectively, t-BHS means solving BHS when at most t many edges can be dynamic.

2.1.5 Variations of Black Hole

A black hole may not be this simple in general. Accordingly, some variations of black holes are first introduced by Královič et al. [88] and Miklík [101], and they are formally defined by Bampas et al. [13]. Here, the authors introduced the concept of *black*$^+$ *hole*, *gray hole*, *gray*$^+$ *hole* and *red hole* (refer the definitions in Section 1.1.2 in Chapter 1). Královič et al. [88] in their paper considered the scheduler to be asynchronous, with the underlying topology to be a ring, communication via whiteboard and agents initially co-located. Next, Bampas et al. [13] significantly improved their results in the same setting. These studies were confined to only a specific initial configuration, to a specific underlying topology and to a specific mode of communication.

Graph	IC	Comp.	Type	Comm.	Scheduler	LB	UB
Ring [88]	Co-loc	Any	Gray	WB	ASYNC	3	9
			Red			3	27
Ring [13]	Co-loc	Any	Gray	WB	ASYNC	4	4
			Red			5	7
Ring [Chapter 5]	Co-loc	Any	BBH	F2F	FSYNC	3	4
				Pebble		3	3
				WB		3	3
	Arb.	Any	BBH	F2F	FSYNC	-	-
				Pebble		4	4
				WB		3	3
Path [Chapter 6]	Co-loc	Any	BBH/Gray	F2F	FSYNC	4	4
		HC				6	6
Tree [Chapter 6]	Co-loc	Any	BBH/Gray	F2F	FSYNC	4	4
		HC				6	6
Arb. Graph [Chapter 6]	Co-loc	Any	BBH/Gray	F2F	FSYNC	$2\Delta-1$	$3\Delta+3$
		HC				-	$3\Delta+3$

Table 2.2: Table gives a summary of all the results in the literature and studied in this thesis related to gray hole, BBH and red hole. Here, HC implies Home Component, WB indicates Whiteboard and F2F indicates face-to-face model of communication.

Further, we defined another variation of the black hole, termed as *Byzantine black hole* (BBH), which is referred to in Section 1.1.2 in Chapter 1 for the definition of BBH. This variation is introduced in our work [76], which is explained in detail in Chapter 5. Our two chapters focus on the presence of this variant of black hole, refer to Chapter 5 and

Chapter 6. In Chapter 5, we studied the perpetual exploration problem in the presence of BBH, and considered the underlying topology to be a ring, scheduler to be synchronous, but initial configuration we considered both co-located and arbitrary (i.e., scattered), with different modes of communication (i.e., face-to-face, pebble and whiteboard). Further, all these studies are again confined to the underlying topology being a ring. Hence, in Chapter 6, we extended our problem of perpetual exploration in the presence of at most one BBH in arbitrary topologies. Moreover, the results obtained in Chapter 6 are the first results obtained on an arbitrary graph for any powerful variant of black hole. A detailed survey of results related to gray hole, BBH and red hole is described in Table 2.2.

Chapter 3

Treasure Hunt in Euclidean Plane

3.1 Introduction

The treasure hunt problem is a significant searching problem, with numerous studies having been conducted in the literature since its inception. The use of pebbles as a form of advice is not new. It has also been used to perform a treasure hunt; however, all these studies are confined to graph networks. In this chapter, we introduce the use of pebbles as a means for the mobile agent to determine the location of the treasure, considering the underlying topology to be a Euclidean plane.

We study the problem of treasure hunting in the Euclidean plane under a very weak scenario, which assumes very little knowledge and control power for the mobile agent. Specifically, the agent does not have any prior knowledge about the position of the treasure or its distance from the treasure. Moreover, the agent has no control over its movement speed, and it is assumed that an adversary completely controls the agent's speed. In practice, for software agents in a network, the movement speed of the agent depends on various factors, such as congestion in the network. In the case of hardware mobile robots, their speeds depend on many mechanical characteristics as well as environmental factors. The agent is equipped with a perfect compass, which helps the agent to rotate and move in

[3]This chapter has been published as: "Treasure Hunt in Euclidean plane: Cost vs. Pebble Trade-off" in *International Journal of Foundations of Computer Science (IJFCS)*, 2025 and as "Pebble Guided Treasure Hunt in Plane" in *Proceedings of the 11th International Conference on Networked Systems (NETYS 2023)*.

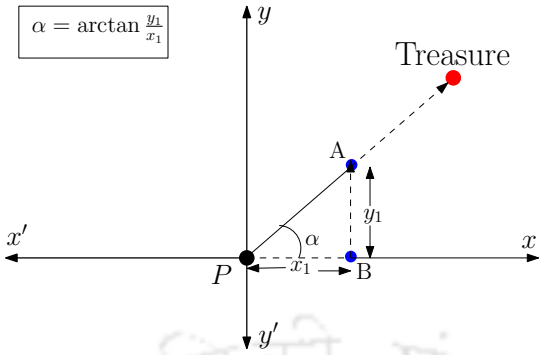


Figure 3.1: Signifies the case when treasure is in the first quadrant and pebbles can be placed anywhere. Placement of pebbles is indicated by blue dots

a prescribed direction, and we also assume that the agent has unbounded memory, i.e., it is a Turing machine with infinite tape. In practicality, if we suppose the agent has no angular inaccuracy but has no control over its speed, then it becomes challenging. Consider an uneven terrain where the slope varies depending on the terrain. In that case, the speed of the agent may also vary depending on the slope of that terrain, and controlling it becomes a challenging task. The agent is initially placed at a point P in the plane. The treasure T is located at most $D > 0$ distance (unknown to the agent) from P. Without loss of generality, we may assume the coordinates of P is (0,0). Let the position of the treasure be the point T with coordinates (x_T, y_T), which is D distance apart from P. The agent finds the treasure only when it reaches the exact position of the treasure. The agent's initial position is considered a special point, and the agent can detect this point whenever it visits P (a similar model is assumed in [64]).

The treasure can be easily found if the agent has full control over its speed and can accurately measure travel distance during its movement. Let us assume that the position of the treasure is in the first quadrant. We briefly describe the strategy of the treasure hunt for the following two cases.

1. Pebbles are allowed to be placed at any point. In this case, draw a line connecting the starting point of the agent P and the position of the treasure T. Choose a point A on the line PT within distance $\epsilon > 0$ from P. Draw a perpendicular line from A to the positive x axis. Let B be the foot of the perpendicular. Two pebbles are placed, one at A and the other at B. The agent, starting from P, moves along the positive x axis

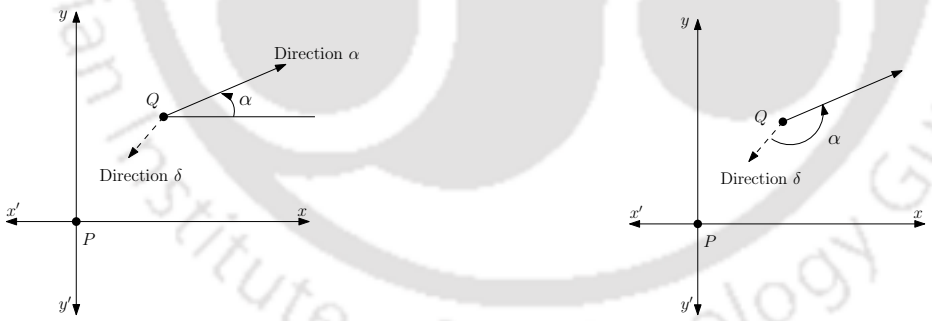
until it finds a pebble. Once the pebble is found, it measures its distance x_1 from P and starts moving upwards until it finds another pebble at a distance y_1 from its last position on the x axis. After reaching the position of the second pebble, the agent calculates $\alpha = \arctan(\frac{y_1}{x_1})$, and from y, starts moving at an angle α with respect to the x axis, to find the treasure. The total distance traveled by the agent is $dist(P,T)+O(\epsilon)$. Figure 3.1 pictorially visualizes this case.

2. The distance between any two placed pebbles must be at least 1. In this case also, draw a line connecting the starting point of the agent P and the position of the treasure T. Let the slope of that line be m. Now, choose a point $A = (x_1, y_1)$ on the line with slope m, such that $|y_1 - x_1| \geq 1$. Now, draw a perpendicular line from A to the positive $x-$axis. Let the foot of the perpendicular be B. Two pebbles are placed, one at A and the other at B. The agent follows the same algorithm as discussed in the earlier case and finds the treasure. The total distance traveled by the agent is $dist(P,T) + O(d)$, where $d = |x_1| + |y_1|$.

Now, for other positions of the treasure, i.e., when the treasure lies on a quadrant different from the first quadrant, then to determine the treasure's position, place a few more pebbles. These extra pebbles indicate in which quadrant the treasure belongs.

In the absence of control over its movement speed, once the agent decides to move along a particular angle, it is very important for the agent to learn when to stop its movement. Otherwise, the adversary can increase the speed arbitrarily high, and the agent ends up traversing an arbitrarily large distance. To enable the agent to have some control over its movement, an Oracle, knowing the position of the treasure and the agent's initial position, places some stationary pebbles on the plane. Note that without any additional information provided to the agent, the treasure hunt is not possible to solve even if the treasure is placed on $x-$axis. This is because any algorithm must instruct the agent to choose a direction of movement from its initial position without having any knowledge about the position of the treasure. Hence, if the agent is instructed to move along the positive $x-$axis, then if the treasure is located on the negative $x-$axis, the adversary can make the agent move an arbitrarily large distance along the positive $x-$axis. Hence, some additional in-

formation about the treasure location must be provided. In this chapter, some additional information is provided by placing some pebbles on the plane. Further, the agent neither has the knowledge about the number of pebbles placed nor does it have the knowledge about the position of the pebbles. We assume a restriction on the pebble placement by the Oracle: any two pebbles must be separated by a constant distance, i.e., no two pebbles are placed arbitrarily close [1]. We assume that any two pebbles must be placed at least 1 distance apart. Since the plane is a continuous domain, the pebble placements are designed in a way to implement some kind of discretization of the plane. If the pebbles are allowed to be placed arbitrarily, then an uncountable number of pebbles can be placed in a very small interval, which is equivalent to allowing multiple pebble placements in a point. In order to avoid such a scenario, and to create a discretization of the plane, pebbles are allowed to be placed at least some positive distance apart. Note that allowing pebble placement at least r distance apart for any positive real number works. For simplicity, we assume $r = 1$. The agent can detect the existence of a pebble only when it reaches the position where the pebble is placed by the Oracle. This pebble placement helps the agent control its movement and rules out the possibility of traversing arbitrarily large distances.



(a) The meaning of moving at an angle α means (b) The meaning of rotating at an angle α means

Figure 3.2: Illustrates the meaning of rotation and move.

Starting from some position of the plane, the agent, moving along a specific direction, stops or changes its direction once it encounters a pebble along the path of its movement. Initially, the agent is facing toward the east direction, i.e., to the direction of the positive x-axis. At any time, by "the agent moves along an angle α from a point Q", we mean that from

[1] This is required if the sensing capability of the agent is weak; two pebbles placed very close to each other may not be distinguished by the agent.

Q, it started moving along a line that creates a counter-clockwise angle α with the positive x-axis (refer to Fig. 3.2(a), where the agent at Q while facing direction δ, is instructed to move at an angle α). Also, by "the agent rotates along an angle α from a point Q", we mean that from Q, it started moving along a line that creates a counter-clockwise angle α with from its previous direction of movement (refer to Fig. 3.2(b), where the agent at Q while facing direction δ, is instructed to rotate at an angle α). In general, any movement algorithm of the agent gives instructions to move along a specific angle α until it encounters a special point (i.e., the initial position P or the position of the treasure T) or it hits a pebble.

Notation	**Meaning**
P	Initial position of the agent
T	Position of the treasure with co-ordinates (x_T, y_T)
k	Indicates the total number of pebbles to be placed by the Oracle
D	T is at most D distance apart from P
r	Indicates the least distance between any two pebbles
f_k	It is the function for Oracle to place pebbles
m_1, m_2	Indicates slopes of lines
B	The square bounded by lines $x = 1$, $x = -1$, $y = 1$ and $y = -1$
L_i	Denotes the half-lines ($i \in \{0, 1, \cdots, 2^{k-10}\}$)
S_i	Sector bounded by the half-lines L_i and L_{i+1}
S_{n_T}	Sector containing the treasure
F	Foot of the perpendicular from T on L_{n_T} or L_{n_T+1}
θ_i, α_i β_i	Indicates the angles of rotation, required to identify i-th bit, there have been abuse in notation between α_i and θ_i, they mean the same
μ	Binary string representing the sector of treasure, i.e., the sector S_{n_T}
p_0	Pebble placed at P
p_1, p_2, p_3	Pebbles placed at specific positions
p_{t_1}, p_{t_2}	Pebbles for termination detection
p_{b_i}	Pebble placed to determine i-th bit
p_{T_1}, p_{T_2}, p_T	Pebbles placed in the sector containing treasure

Table 3.1: Important notations used in Chapter 3

Formally, for a given positive integer $k \geq 0$, the Oracle is a function $f_k : (E \times E) \rightarrow E^k$, where E is the set of all the points in the Euclidean Plane. The function takes two points as input: the first one is the initial position of the agent, and the second one is the position of the treasure, and gives k points in the plane as output, which represent the placement of a pebble at each of these k points. The *cost* of the treasure hunt is defined as the distance the

agent travels from its initial position until it finds the treasure.

The central question studied in this chapter is: "For given $k \geq 0$, what is the minimum cost of treasure hunt if at most k pebbles are placed in the plane?"

We discuss an overview of the list of important notations used in this chapter, details of which can be found in Table 3.1.

3.1.1 Our Contribution

In this chapter we solve the treasure hunt problem on the Euclidean plane, with the help of pebbles. Our contributions in this chapter are summarized below.

- For $k = 1$ pebble, we have shown that it is impossible to design a treasure hunt algorithm that finds the treasure with finite cost.
- For $k = 2$ pebbles, we propose an algorithm that finds the treasure with cost at most $(\sqrt{2}+2)D + (\sqrt{2}+2)$, where D is the distance between the initial position of the agent and the position of the treasure.
- For $k \geq 11$, we design an algorithm that finds the treasure using k pebbles with cost at most $O(k^2) + D\left(\sin\theta' + \cos\theta'\right)$, where $0 \leq \theta' < \frac{\pi}{2^{k-10}}$. For sufficiently large values of D and k, the cost of this algorithm is arbitrarily close to D, the cost of the optimal solution in case the position of the treasure is known to the agent.

Table 3.2, represents the results obtained in this chapter.

Number of Pebbles (k)	**Cost to Find Treasure**
$k = 1$	Impossible
$k = 2$	$(\sqrt{2}+2)D + (\sqrt{2}+2)$
$k \geq 11$	$O(k^2) + D\left(\sin\theta' + \cos\theta'\right)$, where $0 \leq \theta' < \frac{\pi}{2^{k-10}}$

Table 3.2: Results obtained in Chapter 3

3.2 Feasibility of Treasure hunt

In this section, we discuss the feasibility of the treasure hunt problem when the Oracle places one or two pebbles, respectively.

3.2.1 Impossibility Result for $k = 1$ pebble

In this section, the following theorem shows that it is impossible to find the treasure at finite cost using at most one pebble.

Theorem 3.2.1. *It is impossible to design a treasure hunt algorithm using at most one pebble that finds the treasure at a finite cost.*

Proof. The agent initially placed at P and the pebble is placed somewhere in the plane by the Oracle. Since the agent has no prior information about the location of the treasure, the treasure can be positioned anywhere in the plane by the adversary. The only possible initial instruction for the agent is to move along a certain angle from P, say α. The agent, along its movement, must encounter a pebble; otherwise, it will continue to move in this direction for an infinite distance, as specified in the model assumptions. After encountering the pebble, there are three possibilities: (1) either it may return back to P and move at a certain angle from P or, (2) it may return back to P and move along the same path traversed by the agent previously to reach the pebble or, (3) it may rotate at a certain angle from the pebble itself.

Suppose, in the above cases, the agent is next instructed to rotate at an angle θ from the pebble or move at an angle θ from P. Now, in each case, suppose the adversary places the treasure at an angle $\theta + \epsilon$ ($\neq \alpha$), for some $\epsilon > 0$, and at a distance $D(> 0)$ from P. In each case, the agent following any of the possible strategies can never encounter the treasure, as the position of the treasure is not on the path to be traversed by the agent. Moreover, this process can continue indefinitely, since the adversary is always able to choose a real-valued ϵ, such that the treasure placement is not along any of the traveled trajectories. Hence, it is impossible to find the treasure at a finite cost. □

3.2.2 Treasure Hunt for $k = 2$ pebbles

In this part, we discuss the strategy of pebble placement and the respective traversal of the agent towards the treasure when two pebbles are placed by the Oracle.

Pebble Placement: Let the coordinates of the treasure T be (x_T, y_T). Based on the location of the treasure, two pebbles p_1 and p_2 are placed as follows.

- If $x_T \geq 0$ and $y_T \geq 0$, then place the first pebble, i.e., p_1 at $(z+1, z+1)$, where $z = \max\{|x_T|, |y_T|\}$, whereas, place the second pebble, i.e., p_2 at $(x_T, z+1)$.
- If $x_T < 0$ and $y_T \geq 0$, then place p_1 at (y_T+1, y_T+1) and p_2 at (x_T, y_T+1).
- If $x_T < 0$ and $y_T < 0$, then place p_1 at $(1,1)$ and p_2 at $(x_T, 1)$.
- If $x_T \geq 0$ and $y_T < 0$, then place p_1 at (x_T+1, x_T+1) and p_2 at (x_T, x_T+1).

Treasure Hunt by the agent: The agent initially at P, moves at an angle $\frac{\pi}{4}$ with the positive x axis until it encounters treasure or a pebble (i.e., p_1). If a pebble is encountered, then from this position the agent rotates along an angle $\pi - \frac{\pi}{4}$ and then moves, until it encounters the treasure or reaches a pebble (i.e., p_2). If a pebble is encountered (i.e., from p_2), the agent further rotates along an angle $\frac{\pi}{2}$ and moves until it reaches the treasure T.

Theorem 3.2.2. *The agent finds the treasure with cost at most* $(\sqrt{2}+2)D + (\sqrt{2}+2)$ *using the above algorithm.*

Proof. First, we show that for each possible position of the treasure, our algorithm successfully locates it. Based on the position of the treasure, we have the following cases:
Case-1: $T = (x_T, y_T)$, where $x_T \geq 0$ and $y_T \geq 0$. The agent moves along an angle $\frac{\pi}{4}$ from P. If the treasure lies along this line, then no pebbles are placed, and the agent finds the treasure.

If the treasure does not lie along this line, then the agent encounters a pebble (i.e., p_1) at the point $(z+1, z+1)$, where $z = \max\{|x_T|, |y_T|\}$, since $\angle p_1 PB = \frac{\pi}{4}$, where B is the foot of the perpendicular from p_1 to x-axis. Next, from p_1, the agent is instructed to rotate at an angle $\pi - \frac{\pi}{4}$ and then move. While moving along this direction, the agent encounters

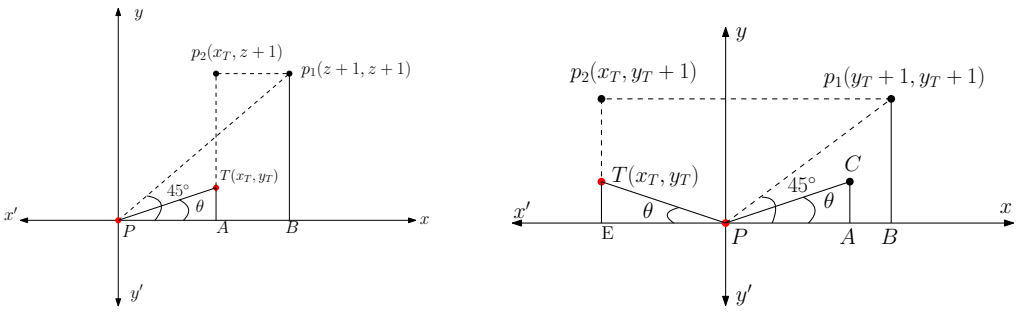
the second pebble (i.e., p_2). It is because $\angle P p_1 p_2 = \frac{\pi}{4}$ (refer to Fig. 3.3(a)). Next, note that the location of p_2 is such that it is perpendicular above the treasure T. Also, since the line $p_1 p_2$ being parallel with respect to $x-$axis (p_1 and p_2 locations have same y coordinate), therefore rotating at an angle $\frac{\pi}{2}$ from p_2, directly aligns the agent along the line $p_2 T$. So, while moving along this line in the third step, it eventually finds the treasure.

Case-2: $T = (x_T, y_T)$, where $x_T < 0$ and $y_T \geq 0$. In this case, as per our pebble placement strategy, the pebble p_1 is placed at $(y_T + 1, y_T + 1)$, and the second pebble p_2 is placed at $(x_T, y_T + 1)$. Hence, the agent, starting from P, starts moving along an angle $\frac{\pi}{4}$ and finds the pebble p_1. According to the movement algorithm, it then moves along the direction of the negative x-axis and finds the pebble p_2. And finally, moving along the negative y-axis from the position of p_2, it finds the treasure.

Case-3: $T = (x_T, y_T)$, where $x_T < 0$ and $y_T < 0$. In this case, as per our pebble placement strategy, the pebble p_1 is placed at $(1, 1)$, and the second pebble p_2 is placed at $(x_T, 1)$. Hence, the agent, starting from P, starts moving along $\frac{\pi}{4}$ and finds the pebble p_1. According to the movement algorithm, it then moves along the direction of the negative $x-$axis and finds the pebble p_2. And finally, moving along the negative $y-$axis from the position of p_2, it locates the treasure.

Case-4: $T = (x_T, y_T)$, where $x_T \geq 0$ and $y_T < 0$. In this case, as per our pebble placement strategy, the pebble p_1 is placed at $(x_T + 1, x_T + 1)$, and the second pebble p_2 is placed at $(x_T, x_T + 1)$. Hence, the agent, starting from P, moves along $\frac{\pi}{4}$ and finds the pebble p_1. Next, as per the movement algorithm, it then moves along the direction of the negative $x-$axis and finds the pebble p_2. And finally, moving along the negative $y-$axis from the position of p_2, it locates the treasure.

Next, according to the proposed algorithm, the maximum cost of finding the treasure is the cost of traversing the path $P p_1 + p_1 p_2 + p_2 T$ (refer to Fig. 3.3 and 3.4), where P is the initial position of the agent, p_1 and p_2 are the positions of the first and second pebble, and T is the position of the treasure, respectively. Let $f_i : \theta \longrightarrow \mathbb{R}$, where $i = 1, \ldots, 5$ and $0 \leq \theta \leq 2\pi$, be the set of cost functions for each of the following cases, we analyze each of them as follows:



(a) Treasure in 1st Quadrant (b) Treasure in 2nd Quadrant

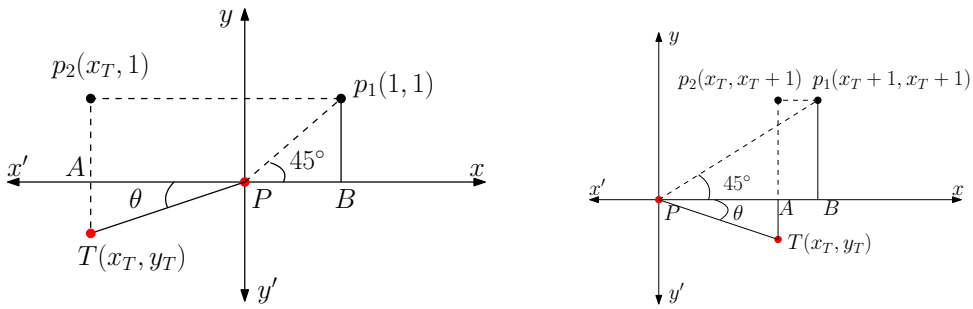
Figure 3.3: Movement of the agent when the treasure is located in the upper half of the plane

1: If the treasure is in the first quadrant, then let A and B each be the foot of the perpendicular drawn from T and p_1 to the $x-$axis, respectively. Let θ be the $\angle TPA$ (refer to Fig. 3.3(a)), so we have $|PA| = x_T = D\cos\theta$ and $|AT| = y_T = D\sin\theta$. Now we have the following cases:

1(a): When $x_T \geq y_T$, then by the pebble placement strategy the pebbles p_1 and p_2 are placed at (x_T+1, x_T+1) and (x_T, x_T+1), respectively. So, $|PB| = D\cos\theta + 1$ (since $|PA| = D\cos\theta$ and $|AB| = 1$) and $|PB| = |Bp_1|$, (since $\angle p_1PB = 45°$ and $\tan 45° = \frac{|Bp_1|}{|PB|}$, so in Δp_1PB we have $|PB| = |Bp_1|$). This implies $|Pp_1| = \sqrt{2}(D\cos\theta+1)$. Moreover in this case, $|p_1p_2| = 1$ and, as $|p_1B| = |p_2A| = x_T+1 = D\cos\theta+1$, hence $|p_2T| = |p_2A| - |TA| = D\cos\theta+1-D\sin\theta$. So, the total cost is: $\sqrt{2}(D\cos\theta+1)+1+(D\cos\theta+1-D\sin\theta) = (\sqrt{2}+1)D\cos\theta+(\sqrt{2}+2)-D\sin\theta$.

1(b): When $y_T > x_T$, then by the pebble placement strategy, the pebbles p_1 and p_2 are placed at (y_T+1, y_T+1) and (x_T, y_T+1), respectively. So, $|Bp_1| = D\sin\theta+1$ and $|PB| = |Bp_1|$ (since $\angle p_1PB = 45°$ and $\tan 45° = \frac{|Bp_1|}{|PB|}$, so in Δp_1PB we have $|PB| = |Bp_1|$), this implies $|Pp_1| = \sqrt{2}(D\sin\theta+1)$. Also, $|Bp_1| = |p_2A| = y_T + 1 = D\sin\theta+1$. Moreover, in this case, $|p_1p_2| = (y_T+1) - x_T = (D\sin\theta+1) - D\cos\theta$ and $|p_2T| = |p_2A| - |TA| = D\sin\theta+1-D\sin\theta = 1$. So, the total cost is: $\sqrt{2}(D\sin\theta+1)+(D\sin\theta+1-D\cos\theta)+1 = (\sqrt{2}+1)D\sin\theta+(\sqrt{2}+2)-D\cos\theta$.

So, we define $f_1(\theta) = \max\{(\sqrt{2}+1)D\cos\theta+(\sqrt{2}+2)-D\sin\theta, (\sqrt{2}+1)D\sin\theta+(\sqrt{2}+2)-D\cos\theta\}$.



(a) Treasure in 3rd Quadrant (b) Treasure in 4th Quadrant

Figure 3.4: Movement of the agent when the treasure is located at the lower half of the plane

2: If the treasure is in the second quadrant, let C be the mirror image of T on the first quadrant (refer to Fig. 3.3(b)), then consider E, A and B to be the foot of the perpendicular drawn from T, C and p_1, respectively. Let $\angle TPE = \theta$, and hence $\angle CPA = \theta$. So, we have $|PA| = |x_T| = D\cos\theta$ and $|AC| = y_T = D\sin\theta$. According to the pebble placement strategy, p_1 is placed at (y_T+1, y_T+1), whereas, p_2 is placed at (x_T, y_T+1). So, we have $|Bp_1| = D\sin\theta + 1$ and $|Bp_1| = |PB|$ (since $\angle p_1PB = 45°$ and $\tan 45° = \frac{|Bp_1|}{|PB|}$, so in Δp_1PB we have $|PB| = |Bp_1|$), this implies $|Pp_1| = \sqrt{2}(D\sin\theta + 1)$. Now, $|p_1p_2| = |PB| + |PE| = D\sin\theta + 1 + D\cos\theta$ and $|p_2T| = 1$, as $p_2 = (x_T, y_T + 1)$ and $T = (x_T, y_T)$. So, the total cost is: $\sqrt{2}(D\sin\theta + 1) + (D\sin\theta + 1 + D\cos\theta) + 1 = (\sqrt{2} + 1)D\sin\theta + D\cos\theta + (\sqrt{2} + 2)$. So, we define $f_2(\theta) = (\sqrt{2}+1)D\sin\theta + D\cos\theta + (\sqrt{2}+2)$.

3: If the treasure is in the third quadrant, let A and B each be the foot of the perpendicular drawn from p_2 and p_1 to the x-axis, respectively. Let $\angle TPA = \theta$ (refer to Fig. 3.4(a)) and by the pebble placement strategy, the pebbles p_1 and p_2 are placed at $(1, 1)$ and $(x_T, 1)$, respectively. So, $|Pp_1| = \sqrt{2}$, $|p_1p_2| = |PB| + |PA| = 1 + D\cos\theta$ and $|p_2T| = |p_2A| + |AT| = 1 + D\sin\theta$. So, the total cost is $\sqrt{2} + 2 + D\cos\theta + D\sin\theta$. So, we define $f_3(\theta) = \sqrt{2} + 2 + D\cos\theta + D\sin\theta$.

4: If the treasure is in the fourth quadrant, then let A and B each be the foot of the perpendicular drawn from T and p_1 to the positive x-axis, respectively, and $\angle TPA = \theta$ (refer to Fig. 3.4(b)), hence, $|PA| = x_T = D\cos\theta$ and $|AT| = |y_T| = D\sin\theta$. According to the pebble placement strategy, p_1 and p_2 are placed at $(x_T + 1, x_T + 1)$ and $(x_T, x_T +$

1), respectively. So, we have $|PB| = x_T + 1 = D\cos\theta + 1$ and $|Bp_1| = x_T + 1 = D\cos\theta + 1$, this implies $|Pp_1| = \sqrt{2}(D\cos\theta + 1)$. Now, $|p_1p_2| = 1$ and $|p_2T| = |p_2A| + |AT| = (x_T + 1) + |y_T| = D\cos\theta + 1 + D\sin\theta$. So, the total cost is: $\sqrt{2}(D\cos\theta + 1) + (D\cos\theta + 1 + D\sin\theta) + 1 = (\sqrt{2}+1)D\cos\theta + (\sqrt{2}+2) + D\sin\theta$. So, we define $f_4(\theta) = (\sqrt{2}+1)D\cos\theta + (\sqrt{2}+2) + D\sin\theta$.

Further, the worst-case cost is $f_5(\theta) = \max_{\forall i \in \{1,\ldots,4\}}\{f_i(\theta)\} \leq (\sqrt{2}+2)D + (\sqrt{2}+2)$, since $|\sin\theta| \leq 1$ and $|\cos\theta| \leq 1$, $\forall\ 0 \leq \theta \leq 2\pi$. Hence, from all the above cases, we conclude that following the above algorithm, the maximum cost required to find the treasure is $(\sqrt{2}+2)D + (\sqrt{2}+2)$. □

3.3 Improved solution for treasure hunt

In this section, we propose a faster algorithm that requires at least 11 pebbles to perform the treasure hunt.

3.3.1 High level idea

Before we give the details of the pebble placement algorithm, we describe the high-level idea. Intuitively, depending on the number of pebbles available, the Oracle divides the plane into multiple sectors as described in Section 3.3.2. Then it identifies the sector number n_T in which the treasure is located and 'encodes' this number by placing the pebbles. The agent, looking at the pebble placements, 'decodes' this encoding and moves to the specified sector to find the treasure. There are certain challenges that need to be overcome to implement this idea.

Learning the sector number: The first difficulty is how different placements of pebbles enable the agent to differentiate between the bit 0 and the bit 1. Since the agent has no sense of time, distance, or the number of pebbles placed by the Oracle, it is possible that two different pebble placements may appear identical to the agent. On the other hand, the agent has no prior information about the encoded integer, so its movement should also be planned in a way that, using the same movement strategy, the agent will detect the bit

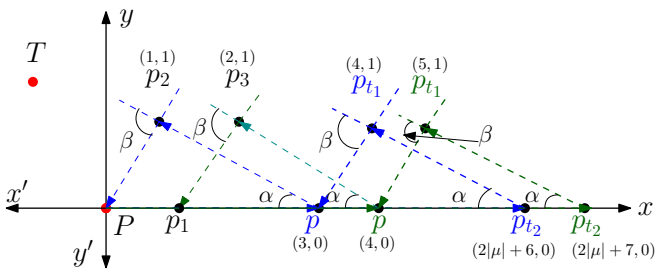


Figure 3.5: Placement of pebbles by Oracle when the first bit is 1

zero for some instances and the bit 1 for other instances. The capability of detecting P as a special point is used to overcome this difficulty.

First, we place a pebble p_1 at the point (1,0) and two additional fixed pebbles p_2 at (1,1) and p_3 at (2,1). The rest of the pebbles are placed based on whether a particular bit of the encoding is a 0 or 1. Initially, consider the specific scenario of encoding only one bit 0 or 1. The idea is to place a particular pebble p in two possible positions (refer to Fig. 3.5) on the x-axis such that the agent, starting from P, reaches p_1 then p, then rotating at a certain fixed angle $\pi - \alpha$ from p and moving in this direction, will reach p_2 for one position of p and p_3 for the other position of p. The agent can not distinguish between p_2 and p_3, but rotating in a particular angle β from p_2 and then moving will reach P and from p_3 will reach p_1. These two different scenarios are distinguished as 1 and 0, respectively. To achieve and implement the idea, the pebble p is placed at the point (3,0) in the case of encoding 1 and (4,0) in the case of encoding 0. The advantage of this specific placement is that in case of placing p at (3,0) moving from P to p, and then rotating at an angle $\pi - \arctan(\frac{1}{2})$ [2] and moving in this direction, the agent reaches p_2 and then rotating at an angle $\arctan(3)$ [3] and moving in this direction, it reaches P. On the other hand, in the case of placing p at (4,0), using the same angular movement, the agent arrives at p_1. Hence, it detects these two different observations as two different bits, 1 and 0, respectively.

We extend the above idea for encoding any binary string μ as follows: in addition to the pebbles p_1, p_2, and p_3, one additional pebble for each of the bits of μ is placed. To be

[2] Consider two lines Pp with slope m_1, where $p = (3,0)$ and pp_2 with slope m_2, now we have $\tan\alpha = \left|\frac{m_1 - m_2}{1 + m_1 m_2}\right|$.

[3] In this case, consider two lines pp_2 with slope m_2, where $p = (3,0)$ and p_2P with slope m_3, accordingly calculate $\tan\beta = \left|\frac{m_2 - m_3}{1 + m_2 m_3}\right|$.

specific, for $1 \leq i \leq |\mu|$, a pebble p_{b_i} is placed at $(2i+1,0)$ if the i-th bit is 1, else p_{b_i} is placed at $(2i+2,0)$. Starting from P to p_{b_i}, rotating at an angle $\pi - \arctan\left(\frac{1}{2i}\right)$ and moving, until a pebble is reached, then again rotating at an angle $\arctan\left(\frac{2i+1}{2i-1}\right)$ and moving, the agent reaches either P or p_1 depending on whether the i-th bit is 1 or 0, respectively. If the agent ends up at a pebble, it then rotates along $\frac{7\pi}{4}$ and moves until it reaches P, at which point it is ready to restart the process to learn the next bit.

A difficulty that remains to be overcome is how the agent detects the end of the encoding. This is important because if the termination is not indicated, then there is a possibility that the agent moves to find more pebbles p_{b_j}, $j > |\mu|$, and continues its movement for an infinite distance. We use two additional pebbles, namely p_{t_1} and p_{t_2}, specifically for the purpose of termination detection. The position of these two pebbles p_{t_1} and p_{t_2} are as follows: if the 1st bit of the binary string μ is 1, i.e., p_{b_1} is placed at $(3,0)$, then the pebbles p_{t_1} and p_{t_2} are placed at $(4,1)$ and $(2|\mu|+6,0)$, respectively. Otherwise, if the 1st bit is 0 then these two pebbles are placed at $(5,1)$ and $(2|\mu|+7,0)$, respectively (refer to Fig. 3.5). After visiting the pebble $p_{|\mu|}$ for the last bit of μ, the agent returns to P, and moves as usual to find a pebble, expecting to learn more bits of the binary string. From P, once it reaches p_{t_2}, it rotates at an angle $\pi - \arctan\left(\frac{1}{2(|\mu|+1)}\right)$ and moves until a pebble is reached. Note that the two pebbles p_{t_1} and p_{t_2} are placed in such a way that the angle $\angle P p_{t_2} p_{t_1} = \arctan\left(\frac{1}{2(|\mu|+1)}\right)$. Hence using the movement from p_{t_2} at angle $\pi - \arctan\left(\frac{1}{2(|\mu|+1)}\right)$ the agent reaches p_{t_1} and from p_{t_1} rotating at angle $\arctan\left(\frac{2(|\mu|+1)+1}{2(|\mu|+1)-1}\right)$ and moving, it reaches to p_{b_1}. By following the movement specified above, the agent reaches a pebble, so it will assume that it learned the bit zero, and will move west (with respect to $x-$axis) to try to reach P. But moving west from p_{b_1}, it reaches another pebble (i.e., the pebble p_1), instead of P. This special occurrence indicates that the agent should ignore this final bit 0, and terminate the process of learning μ. Hence, in this way, the agent learns the binary string μ, and the integer n_T whose binary representation is μ.

Finding the treasure inside the sector: One more pebble p_T is placed on the foot of the perpendicular drawn from T on L_{n_T+1} (or L_{n_T}) (refer to Fig. 3.10), where L_{n_T} and L_{n_T+1} are the two boundaries that form S_{n_T}, as defined under case $1a$ in Section 3.3.2. After learning the encoding of μ (where the binary string μ obtained represents the integer n_T), the agent

decodes the integer n_T and correctly identifies the two lines L_{n_T} and L_{n_T+1}, which the agent will use to locate the treasure.

A difficulty arises here while placing the pebble p_T inside the sector, as some pebbles that are used to encode the sector number might be very close (i.e., distance < 1) from the prescribed position of p_T. To resolve this, we place the encoding pebbles on the positive x-axis if the treasure's position is in the left half plane, whereas we place the encoding pebbles on the negative x-axis if the treasure's position is in the right half plane. In order to instruct the agent whether it should move east or west from P to find the encoding pebbles, one additional pebble p_0 is placed at P in one of the two cases.

The following cases are handled separately, based on the position of the pebble p_T and the treasure T.

1.If the treasure is in a position (x, y) such that $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$, then in that case, placement of the pebble p_T on the prescribed position inside the sector may create a problem as the distance between p_0 at P and p_T may be less than 1, violating our pebble placement criteria. To address this case, we propose a specific pebble placement strategy (refer to Case 2 in Section 3.3.2) that ensures this situation never arises. This strategy only requires three pebbles, namely, p_1, p_2 and p_3, in which p_1 is always placed at $(1,0)$ (or, $(-1,0)$, based on the position of the treasure) whereas the other two pebbles, are always outside the region $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$. This, in turn, solves the problem.

2.The position of the treasure (x, y) is such that $x > 1$, or $x < -1$, or $y > 1$, or $y < -1$, but the position of the pebble p_T at (x_1, y_1) is such that $-1 \leq x_1 \leq 1$ and $-1 \leq y_1 \leq 1$. In this case, the placement of the pebble p_T may create a problem as the distance between p_0 at P and p_T may be less than 1. To handle this case, an additional two pebbles p_{T1} and p_{T2} may be used. So, in some cases instead of just p_T, two additional pebbles p_{T1} and p_{T2} are required.

3.3.2 Pebble placement

The agent is initially placed at P, and the treasure is placed at T. The Oracle, knowing the initial position P of the agent and the position $T = (x_T, y_T)$ of the treasure, places k pebbles

in the Euclidean plane. Let B be the square region bounded by the lines $x = 1$, $x = -1$, $y = 1$, and $y = -1$. The following two cases describe the pebble placement strategy, based on the treasure's position.

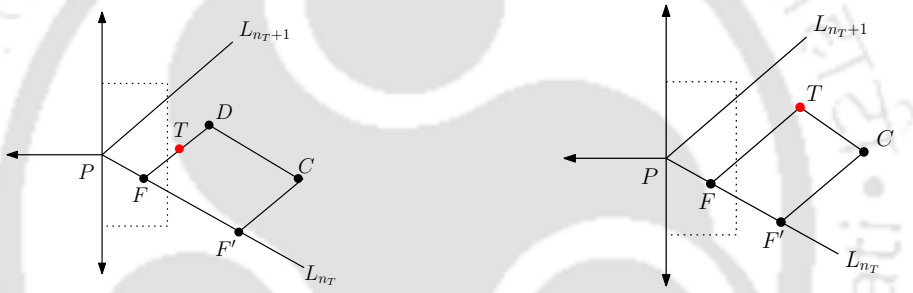
Case 1: If the treasure T is located outside B (i.e., $T \notin B$), then for any fixed $k \geq 11$, the pebble placement strategy is as follows.

- **Case** $1a$**:** If $x_T \geq 0$, then we have the following pebble placement strategy.
  - Place a pebble p_0 at P.
  - Draw $2^{k-10}+1$ half-lines $L_0, L_1, \ldots, L_{2^{k-10}}$, starting at the initial position P of the agent, such that L_0 goes South and the counter-clockwise angle between consecutive half-lines, i.e., L_i and L_{i+1} is $\pi/2^{k-10}$ for $i = 0, 1, \cdots, 2^{k-10}-1$. The sector S_i is defined as the set of points in the plane between L_i and $L_{(i+1)}$, including the points on L_i and excluding the points on $L_{(i+1)}$. Let $n_T \in \{0, 1, \cdots, 2^{k-10}-1\}$ such that $T \in S_{n_T}$, and place $k \geq 11$ pebbles as follows.
    * Place the pebbles p_1 at (-1,0), p_2 at (-1,-1) and p_3 at (-2,-1).
    * Let μ be the binary representation of the integer n_T with leading $k-11-\lfloor \log n_T \rfloor$ many zeros, when $n_T > 0$ and with leading $k-11$ zeroes, when $n_T = 0$. If $0 \leq x_T \leq 1$ and $y_T > 1$, then $\mu = 0 \cdot \mu$, else $\mu = 1 \cdot \mu$ (For example, suppose $\mu = 0101$ and if $0 \leq x_T \leq 1$ and $y_T > 1$, then $\mu = 00101$, otherwise $\mu = 10101$). For $1 \leq \ell \leq |\mu|$, if the ℓ-th bit of μ is 1, then place a pebble at $(-2\ell - 1, 0)$, else place a pebble at $(-2\ell - 2, 0)$.
    * If the 1st bit of μ is 1, then place a pebble p_{t_1} at (-4,-1), else place p_{t_1} at (-5,-1).
    * If the 1st bit of μ is 1, then place a pebble p_{t_2} at $(-2|\mu_j| - 6, 0)$, else place p_{t_2} at $(-2|\mu_j| - 7, 0)$.
  - If the first bit of μ is 0, then let F be the foot of the perpendicular drawn from T to L_{n_T+1}, else let F be the foot of the perpendicular drawn from T to L_{n_T}. We have the following cases:

1. If $F \in B$, then let F' be the point on L_{n_T+1}, or L_{n_T} (same line as F), such that $|FF'| = 2$. Place the pebble p_T at F'. Next, we have the further cases:

* If $|FT| < 1$, then C be the point towards T, such that $\angle PF'C = \frac{\pi}{2}$ and $|F'C| = 1$. Also let D be the point towards T, such that $\angle F'CD = \frac{\pi}{2}$ and $|CD| = |FF'|$. Place a pebble p_{T1} at C and a pebble p_{T2} at D. Refer to Fig. 3.6(a).
* If $|FT| \geq 1$, then let C be the foot of the perpendicular from T on the perpendicular line originating from F'. Place a pebble p_{T1} at C. Refer to Fig. 3.6(b).

2. If $F \notin B$, then place a pebble p_T at F.



(a) Represents the pebble placement when $|FT| < 1$

(b) Represents the pebble placement when $|FT| \geq 1$

Figure 3.6: Represents the pebble placement situation when the point F is inside B

- **Case** $1b$**:** If $x_T < 0$, then the pebble placement strategy is as follows. For each pebble placed at (m,n), where $m \neq 0$ or $n \neq 0$ in the above case, place the corresponding pebble at $(-m,-n)$ in this scenario. Also, place no pebble at P. An example of the pebble positions is illustrated in Fig. 3.10.

Lemma 3.3.1. *With the above pebble placement strategy, the binary string μ always consists of exactly $k-9$ bits.*

Proof. To prove the above lemma, we have two cases:

Case $n_T = 0$: In this case, initially $|\mu| = 2$ ($|\mu|$ defines the length of μ), as $\mu = 0$ and then either 0 or 1 is appended to μ, based on position of T. Hence, $|\mu| = 2$. Finally, appending $k-11$ zeroes at the front of μ, transforms μ in to a binary string of length $k-9$ (=$k-11+2$).

Case $n_T > 0$: In this case, we already know that the binary representation of n_T has $1 + \lfloor \log n_T \rfloor$ many bits. Next appending 0 or 1 to μ, based on the position of T, makes $|\mu| = 2 + \lfloor \log n_T \rfloor$. So, adding $k - 11 - \lfloor \log n_T \rfloor$ to μ, makes $|\mu| = k - 11 - \lfloor \log n_T \rfloor + 2 + \lfloor \log n_T \rfloor = k - 9$. □

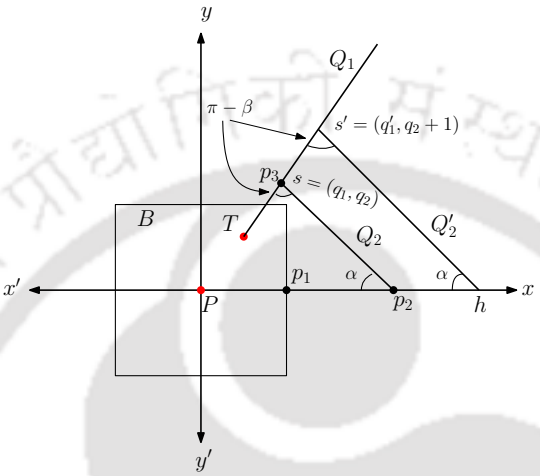


Figure 3.7: Placement of pebbles when the treasure is inside B

Thus, in this case where the treasure is outside B, according to the pebble placement strategy, we use exactly k pebbles. First, a pebble p_0 may be needed to be placed at P, then 8 pebbles p_1, p_2, p_3, p_{t_1}, p_{t_2}, p_T, p_{T1} and p_{T2} are placed at appropriate positions based on the location of T. Lastly, the remaining $k-9$ pebbles are placed on the x-axis, where each such pebble represents a bit value of the binary string μ.

Case 2: If the treasure T is located inside B (i.e., $T \in B$), then the pebble placement strategy for each of the following cases is as follows.

- **Case** $2a$**:** If $x_T \geq 0$, then three pebbles are placed as follows (refer to Fig. 3.7).
  - Place a pebble p_1 at (1,0).
  - Let us consider the slope $m_1 = \tan\left(\pi - \arctan(\frac{1}{2}) - (\pi - \arctan(3))\right) = \tan\left(\arctan(3) - \arctan(\frac{1}{2})\right)$ whereas we have $m_2 = \tan\left(\pi - \arctan\left(\frac{1}{2}\right)\right)$. Draw a line Q_1 through T with slope m_1 and draw a line Q_2 through the point $(2,0)$ with slope m_2. Let $s = (q_1, q_2)$ be the point of intersection between these two lines. Let s' be the point on the line Q_1 whose y coordinate is $q_2 + 1$. Draw the line Q_2'

parallel to Q_2 and going through s'. Let h be the point of intersection of the line Q_2' with the x-axis.

Two additional pebbles p_2 and p_3 are placed as follows. If $q_2 < 1$, then place p_2 at h and p_3 at s'. Otherwise, place p_2 at $(2,0)$ and p_3 at s.

- **Case** $2b$**:** If $x_T < 0$, then four pebbles are placed as follows.
  - Place the pebbles p_0 at P and p_1 at (-1,0).
  - Let the slopes be denoted by m_1 and m_2, where we have the value of $m_1 = \tan\left(\pi - \arctan(\frac{1}{2}) - (\pi - \arctan(3))\right) = \tan\left(\arctan(3) - \arctan(\frac{1}{2})\right)$ and the value of $m_2 = \tan\left(\pi - \arctan\left(\frac{1}{2}\right)\right)$. Draw a line Q_1 through T with slope m_1 and draw a line Q_2 through the point $(-2,0)$ with slope m_2. Let $r = (r_1, r_2)$ be the point of intersection between these two lines. Let r' be the point on the line Q_1 whose y coordinate is $r_2 - 1$. Draw the line Q_2' parallel to Q_2 and going through r'. Let n be the point of intersection of the line Q_2' with the x-axis.

    Two additional pebbles p_2 and p_3 are placed as follows. If $r_2 > -1$, then place p_2 at n and p_3 at r'. Otherwise, place p_2 at $(-2,0)$ and p_3 at r. Note that these pebble placements are just mirror placements of case 2a, where $x_T \geq 0$.

The pseudo-code of the pebble placement strategy is explained in Algorithms 1, 2, 3 and 4. More precisely, the case where T is inside B is explained in Algorithm 2, whereas the case where T is outside B is explained in Algorithm 3 and in this case, the placement of the pebbles on the sector is explained in Algorithm 4.

The following lemma states that, irrespective of which direction the agent is moving, the pebbles are placed in such a way that they satisfy the following principle. If after rotating at a counter-clockwise angle θ from (x, y) will take an agent to (x', y'), then with the same angular rotation from $(-x, -y)$, the agent will reach $(-x', -y')$.

Lemma 3.3.2. *Suppose that the agent located at (x, y) facing direction δ, performs a counter-clockwise rotation of θ, then moves forward and reaches point (x', y') facing direction δ'. Then, if the agent is located at $(-x, -y)$ facing direction $\delta + \pi$, performs a counter-clockwise rotation of θ, then moves forward, it will reach the point $(-x', -y')$ facing direction $\delta' + \pi$.*

Algorithm 1: PEBBLEPLACEMENT(k)

Draw $2^{k-10}+1$ half lines $L_0,\cdots,L_{2^{k-10}}$ starting from P in the corresponding half plane in which the treasure lies, where the counter-clockwise angle between two consecutive half-lines is $\frac{\pi}{2^{k-10}}$. If $x_T \geq 0$, then L_0 goes South, otherwise L_0 goes North. Let Sector S_i be the sector bounded by the half lines L_i and L_{i+1} and let $T \in S_{n_T}$, $n_T \in \{0,1,\cdots,2^{k-10}-1\}$
if $x_T \geq 0$ **then**
  if $0 \leq x_T \leq 1$ *and* $-1 \leq y_T \leq 1$ **then**
    SQUAREPLACEMENT(2)
  else
    Place a pebble p_0 at P
    if $x_T \leq 1$ *and* $y_T > 1$ **then**
      NONSQUAREPLACEMENT(1,0)
      SECTORPEBBLE(0,1)
    else
      NONSQUAREPLACEMENT(1,1)
      SECTORPEBBLE(1,1)
else
  if $-1 \leq x_T < 0$ *and* $-1 \leq y_T \leq 1$ **then**
    Place a pebble p_0 at P and perform SQUAREPLACEMENT(1)
  else
    if $-1 \leq x_T < 0$ *and* $y_T < -1$ **then**
      NONSQUAREPLACEMENT(2,0)
      SECTORPEBBLE(0,-1)
    else
      NONSQUAREPLACEMENT(2,1)
      SECTORPEBBLE(1,-1)

Algorithm 2: SQUAREPLACEMENT($count$)

Place a pebble p_1 at $((-1)^{count},0)$.
$m_1 = \tan\left(\arctan(3) - \arctan(\frac{1}{2})\right)$ and $m_2 = \tan\left(\pi - \arctan\left(\frac{1}{2}\right)\right)$.
Q_1 a line drawn through T with slope m_1.
Q_2 a line drawn through $((-1)^{count}\cdot 2,0)$ with slope m_2.
$s = (q_1,q_2)$ point of intersection of the lines Q_1 and Q_2.
s' the point on the line Q_1 with y coordinate $(q_2 + (-1)^{count})$.
Q_2' a line parallel to Q_2 passing through s'.
h point of intersection of Q_2' with x- axis.
if $|q_2| < 1$ **then**
  Place a pebble p_2 at h.
  Place a pebble p_3 at s'.
else
  Place a pebble p_2 at $((-1)^{count}\cdot 2,0)$.
  Place a pebble p_3 at s.

Proof. Let L_1 be the line originating from $A_1 = (x,y)$ with slope m_1 towards the direction δ (refer to Fig. 3.8). Again, suppose L_2 be the line originating from $A_1' = (-x,-y)$ with slope m_1' towards the direction $\delta+\pi$. Hence, the lines L_1 and L_2 must be parallel to each other,

Algorithm 3: NONSQUAREPLACEMENT($count, bit$)

Initially $l = 1$
Place a pebble p_1 at $((-1)^{count}, 0)$, a pebble p_2 at $((-1)^{count}, (-1)^{count})$, and a pebble p_3 at $((-1)^{count} \cdot 2, (-1)^{count})$.
$\mu = bit \cdot (\mu \backslash bit)$, where $\mu \backslash bit$ is the binary representation of the integer n_T with leading $k - 11 - \lfloor \log n_T \rfloor$ (and $k - 11$, when $n_T = 0$) many zeroes, after the first bit, which is represented by the value bit.
while $l \leq k + 1$ **do**
    if *ℓ-th bit of μ is* 1 **then**
        Place a pebble at $((-1)^{count} \cdot (2\ell + 1), 0)$.
    else
        Place a pebble at $((-1)^{count} \cdot (2\ell + 2), 0)$.
    $l = l + 1$
if *1st bit of μ is 1* **then**
    Place a pebble p_{t_1} at $((-1)^{count} \cdot 4, (-1)^{count})$.
    Place a pebble p_{t_2} at $((-1)^{count} \cdot (2|\mu_j| + 6), 0)$.
else
    Place a pebble p_{t_1} at $((-1)^{count} \cdot 5, (-1)^{count})$.
    Place a pebble p_{t_2} at $((-1)^{count} \cdot (2|\mu_j| + 7), 0)$.

Algorithm 4: SECTORPEBBLE($value, domain$)

if $value = 0$ **then**
    Let $F = (x, y)$ be the foot of the perpendicular drawn from T on L_{n_T+1}.
else
    Let F be the foot of the perpendicular drawn from T on L_{n_T}.
if $domain = 1$ **then**
    Define $box = \{(x, y) | 0 \leq x \leq 1, -1 \leq y \leq 1\}$
else
    Define $box = \{(x, y) | -1 \leq x < 0, -1 \leq y \leq 1\}$
if $F \in box$ **then**
    Place the pebble p_T at F', where F' is the point on same line as PF such that $|FF'| = 2$.
    if $|FT| < 1$ **then**
        Let C be the point towards T, such that $\angle PF'C = \frac{\pi}{2}$ and $|F'C| = 1$.
        Let D be the point towards T, such that $\angle F'CD = \frac{\pi}{2}$ and $|CD| = 2$.
        Place a pebble p_{T1} at C and a pebble p_{T2} at D.
    else
        Place a pebble p_{T1} at C, where C is the foot of the perpendicular drawn on the perpendicular line originating from F', from T.
else
    Place a pebble p_T at F.

since $\tan(\delta) = \tan(\delta + \pi)$. So, we have $m_1 = m_1'$. Now, suppose the agent rotates at an angle θ from A_1 in a counter-clockwise direction, it aligns itself along the line A_1A_2, facing the direction δ'. Let the slope of this new line be m_2. Similarly, if the agent performs a counter-clockwise rotation of θ from A_1', facing $\delta + \pi$ direction, it aligns to the line $A_1'A_2'$. Let the

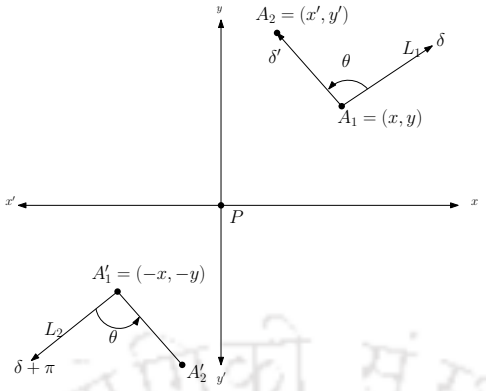


Figure 3.8: Determining that the pebble positions on the adjacent half of y-axis are mirrored

slope of this line be x. Now, we have $\tan\theta = \frac{m_1 - m_2}{1 + m_1 m_2}$ with respect to the lines A_1A_2 and L_1. Similarly, we have $\tan\theta = \frac{m_1 - x}{1 + m_1 x}$ with respect to the lines $A'_1A'_2$ and L_2. Hence, we have:

$$\frac{m_1 - m_2}{1 + m_1 m_2} = \frac{m_1 - x}{1 + m_1 x},$$

which gives $({m_1}^2 + 1)(x - m_2) = 0$. Hence, we have $x = m_2$, since $m_1^2 \neq -1$. Now, as the counter-clockwise direction from A_1 is opposite to that from A'_1, the agent will face $\delta' + \pi$. Moreover, since the slope of the line passing through $(-x', -y')$ and A'_1 is the same as the slope of the line passing through A_1A_2, and also the direction of the point $(-x', -y')$ is along the direction $\delta' + \pi$ from A'_1. This proves the lemma. □

The following lemmas and corollaries ensure that, the pebbles placed on the same half of the plane as the treasure (i.e., p_1, p_2 and p_3 when $T \in B$ and p_T, p_{T1} and p_{T2} when $T \notin B$) obey the condition that the distance between any two pebbles is at least 1. In order to prove this, first we show in the next lemma that when $T \notin B$, the encoding pebbles (i.e., those placed on the adjacent half of the plane) which are nearest to p_T, p_{T_1}, and p_{T_2} are p_0, p_1, and p_2, respectively.

Lemma 3.3.3. *If $T = (x_T, y_T) \notin B$, then among all the pebbles used for encoding, the pebbles p_0, p_1 and p_2 are nearest to the pebbles p_T, or p_{T1}, or p_{T2}.*

Proof. As per the pebble placement strategy, the pebbles p_T, or p_{T_1}, or p_{T_2} are located in the right half of y-axis (resp. left half of y-axis), if $x_T \geq 0$ (resp. $x_T < 0$), where $T = (x_T, y_T)$.

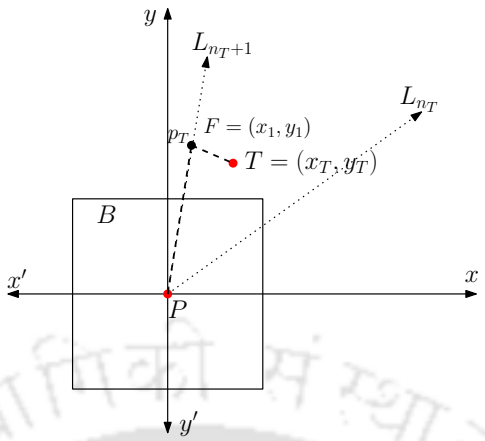


Figure 3.9: Determining the location of the pebble p_T

Note that nearest pebbles to y-axis are: p_0, placed at P (if $x_T \geq 0$), p_1 at $(-1,0)$ (resp. $(1,0)$), p_2 at $(-1,-1)$ (resp. $(1,1)$). It is because the positions of p_3, p_{b_i} (for $i \geq 1$), p_{t_1} and p_{t_2} are each further from the y-axis and have the same y coordinate as one of the pebbles p_0, p_1 or p_2 (refer, their positions in Section 3.3.2). This implies that p_0, p_1, and p_2 are the nearest to any pebble placed while traversing the sector. □

The next lemma states that when T lies inside B_1, where $B_1 = \{(x,y) |\ 0 \leq x \leq 1$ and $-1 \leq y_T \leq 1\}$, then the pebbles p_0, p_1, p_2 and p_3 are all at least 1 distance apart from each other.

Lemma 3.3.4. *If $T = (x_T, y_T) \in B_2$, where $B_2 = \{(x,y) |\ -1 \leq x < 0$ and $-1 \leq y_T \leq 1\}$ then all the pebbles are at least 1 unit distance apart from each other.*

Proof. Since $x_T < 0$, hence the pebbles p_0 is placed at P, whereas the pebble p_1 is placed at $(-1,0)$. Next, the pebble p_2 is placed at $(-2,0)$ or at n, which is located to the left of $(-2,0)$ along the x-axis. This shows the pebbles p_0, p_1 and p_2 are at least 1 distance apart from each other. Next, let the position of p_3 be (x_1, y_1). As per Algorithm 2, $|y_1|$ must be greater than 1. This guarantees that p_3 is also at least 1 unit distance apart from the rest of the pebbles. □

Corollary 3.3.5. *If $T = (x_T, y_T) \in B_1$, where $B_1 = \{(x,y) |\ 0 \leq x \leq 1$ and $-1 \leq y_T \leq 1\}$ then all the pebbles are 1 unit distance apart from each other.*

The above corollary follows from the fact that, the position of the pebbles when $x_T \geq 0$, is exactly same as $x_T < 0$, just mirrored from earlier positions. Hence, all the conditions still hold for this case as well. Also, since $B = B_1 \cup B_2$, Lemma 3.3.4 and Corollary 3.3.5 show that when $T \in B$, all pebbles are at least 1 distance apart from each other. The following lemma states that if T lies in B'_1, where $B'_1 = \{(x, y) | \ 0 \leq x \leq 1 \text{ and } y_T > 1\}$, then the foot F of the perpendicular drawn from T on L_{n_T+1}, lies outside B.

Lemma 3.3.6. *If $T = (x_T, y_T) \in B'_1$, where $B'_1 = \{(x, y) | \ 0 \leq x \leq 1$ and $y_T > 1\}$, the location of the foot F of the perpendicular drawn from T on L_{n_T+1} is outside the square B, where $B = \{(x, y) | \ -1 \leq x \leq 1$ and $-1 \leq y \leq 1\}$.*

Proof. Let the position of F be (x_1, y_1), let $m_1 = \frac{y_1}{x_1}$ be the slope of the line PF and $m_2 = \frac{y_T - y_1}{x_T - x_1}$ be the slope of the line FT (refer to Fig. 3.9). Based on the position of x_1, we have the following cases:

1. $x_1 = 0$: In this case, F must lie on $L_{2^{k-10}}$ and $T \in S_{2^{k-10}-1}$. As per case 1a of Section 3.3.2, $PF \perp FT$, so the position of F must be $(0, y_T)$. Further, since $y_T > 1$, this implies that F must lie outside B.

2. $x_1 > 0$: Again as per the position of F, the condition $PF \perp FT$ must hold. So, we have $m_1 \cdot m_2 = -1$. The slope $m_1 = \frac{y_1}{x_1}$ is positive as $y_1 > 0$ and $x_1 > 0$, so $m_2 = \frac{y_T - y_1}{x_T - x_1}$ must be negative to satisfy the above condition. Now, m_2 can be negative if one of the following cases is true.

Case 1: $y_T > y_1$ and $x_T < x_1$, and Case 2: $y_T < y_1$ and $x_T > x_1$.

In case 1, since $x_T < x_1$, the point F must be on the right side of the line $x = x_T$, which is not possible. In case 2, $1 < y_T < y_1$ and $x_T > x_1$. The fact that $y_1 > y_T > 1$ implies that F is outside B since all points in B have y coordinate at most 1. □

Corollary 3.3.7. *If $T = (x_T, y_T) \in B'_2$, where $B'_2 = \{(x, y) | \ -1 \leq x < 0$ and $y_T < -1\}$, the location of the foot F of the perpendicular drawn from T on L_{n_T+1} is outside the square B, where $B = \{(x, y) | \ -1 \leq x \leq 1$ and $-1 \leq y \leq 1\}$.*

Lemma 3.3.6 and Corollary 3.3.7 shows that whenever $T \in B'$, where $B' = \{(x, y) | -1 \leq x_T \leq 1$ and $|y_T| > 1\}$ (where $B' = B'_1 \cup B'_2$), the point F on L_{n_T+1} lies outside B. The next

lemma proves that, when $T \in B'$, the distance from pebble p_T to any other pebble is at least 1.

Lemma 3.3.8. *If $T = (x_T, y_T) \in B'$, where $B' = \{(x, y) | -1 \leq x_T \leq 1$ and $|y_T| > 1\}$, then the distance between pebble p_T and any other pebble is at least 1.*

Proof. As per Lemma 3.3.6 and Corollary 3.3.7, we show that whenever $T = (x_T, y_T)$ lies on this domain $-1 \leq x_T \leq 1$ and $|y_T| > 1$, then the location of F is outside B. Next, according to the pebble placement strategy, in this case, when F lies outside B, then only a single pebble is placed on the sector containing T. This pebble, i.e., p_T, is placed on F. Again, as F lies outside B, so it is at least 1 distance apart from the nearest pebbles p_0, p_1 and p_2 (refer to Lemma 3.3.3). This proves the lemma. □

The above lemmas and corollaries deal with the cases when $T \in B$ and when $T \in B'$, where $B = \{(x, y) | -1 \leq x \leq 1$ and $-1 \leq y \leq 1\}$ and $B' = \{(x, y) | -1 \leq x \leq 1$ and $|y| > 1\}$. Now, the last remaining case to consider is when T does not lie in B or B', but the point F lies inside B. In this case, we show that the distance from each of the pebbles p_T, p_{T_1} and p_{T_2} to all the other pebbles is at least 1.

Lemma 3.3.9. *If $T \notin B \cup B'$ and $F = (x_1, y_1) \in B_1$, where $B_1 = \{(x, y) | 0 \leq x \leq 1$ and $-1 \leq y \leq 1\}$ then the distance from each of the pebbles p_T, p_{T_1} and p_{T_2} to all the other pebbles is at least 1.*

Proof. Let $T \in S_{n_T}$ and $T \notin B \cup B'$, where $B = \{(x, y) | -1 \leq x \leq 1$ and $-1 \leq y \leq 1\}$ and $B' = \{(x, y) | -1 \leq x \leq 1$ and $|y| > 1\}$. Hence according to Algorithm 1, the point F must lie on L_{n_T} and not on L_{n_T+1}. Now, as per Algorithm 4, whenever $F = (x_1, y_1) \in B_1$, the positions at which the pebbles p_T, p_{T1} and p_{T2} may be placed are F', C and D, respectively (refer to Fig. 3.6(a) and 3.6(b)). Now, based on the distance $|FT|$ we have two cases:

1. $|FT| < 1$: In this case, all three pebbles p_T, p_{T1} and p_{T2} are placed at F', C and D, respectively. Now, as per the pebble placement strategy $|FF'| = 2$, so the pebble is outside B. This is because L_{n_T} originates from P and the maximum distance of a point inside B from P is at most $\sqrt{2}$. Next, the point C is such that $\angle PF'C = \frac{\pi}{2}$ and $|F'C| = 1$. This shows that first, the point C is 1 distance apart from F' and second, it lies outside B. The reason

is that, as T lies outside B and the line FT is parallel to $F'C$, hence C must lie outside B. Further the point D is such that, $\angle F'CD = \frac{\pi}{2}$ and $|CD| = 2$. So, as per construction, $FF'CD$ forms a rectangle. This is because $\angle FF'C = \angle F'CD = \frac{\pi}{2}$ and $|FF'| = |CD| = 2$, so the line perpendicular from F must touch the point D, i.e., $\angle DFF' = \frac{\pi}{2}$. This shows that, all three angles of $FF'CD$ are of angle $\frac{\pi}{2}$, also $|FF'| = |CD|$, so this proves $FF'CD$ is a rectangle. Now, as $FT \perp PF'$ and also $FD \perp PF'$ and $|FT| < |FD|$ (since $FF'CD$ being a rectangle, $|FD| = |F'C| = 1$), so we can conclude that T lies on the line segment FD. Hence, D lies outside B, as $T \notin B$. Moreover, the distance between points D and F' is at least 1 (since $|F'D| = \sqrt{5}$) and the distance between D and C is at least 1 (since $|CD| = 2$).

2. $|FT| \geq 1$: In this case, pebbles are placed at F' and C. C is the foot of the perpendicular from T, drawn on perpendicular line originating from F'. Again, $FF'CT$ is a rectangle, because, observe $\angle TFF' = \angle FTC = \angle FF'C = \frac{\pi}{2}$. So, we have $|FF'| = |CT| = 2$ and $|FT| = |F'C| \geq 1$. This shows that, C and F' lie outside B, and are at least 1 distance apart from each other. □

Corollary 3.3.10. *If $T \notin B \cup B'$ and $F = (x_1, y_1) \in B_2$ where $B_2 = \{(x, y) | -1 \leq x < 0$ and $-1 \leq y \leq 1\}$ then all the pebbles placed, i.e., among p_T, p_{T1} and p_{T2}, are at least 1 distance apart from the other pebbles.*

Corollary 3.3.11. *If $T \notin B'$ and $F \notin B$ then p_T is at least 1 distance apart from remaining pebbles.*

To see why Corollary 3.3.11 is true, note that $F \notin B$ and $T \notin B'$ imply that F lies on L_{n_T} (if $T \in S_{n_T}$). As per the pebble placement strategy in Algorithm 1, this means that only a single pebble is placed, and that is p_T on F. So, this pebble p_T, being outside of B, is at least 1 distance apart from the nearest pebbles.

The next theorem is a direct consequence of Lemmas 3.3.8 and 3.3.9 and also Corollaries 3.3.10 and 3.3.11.

Theorem 3.3.12. *The distance from each of the pebbles p_T, p_{T_1} and p_{T_2} to all the other pebbles is at least 1.*

3.3.3 Treasure hunt

Starting from P, the agent finds the treasure with the help of the pebbles placed at different points on the plane. On a high level, the agent performs three major tasks: (1) Learn the direction of its initial movement, (2) Learn the encoding of the sector number in which the treasure is located, and (3) Move inside the designated sector and find the treasure.

The agent learns the direction of its initial movement by observing whether a pebble is located at P or not. If a pebble is present, then it learns that the direction of its initial movement is west, and pebble placement is done for the encoding of the sector number on the negative x axis. Otherwise, it learns that the direction of its initial movement is east and pebble placement is done for the encoding of the sector number on the positive x axis (refer to steps 1-5 of Algorithm 5). Now, for each $j = 1, 2, \cdots$, it continues its movement along a specific path (depending on the value of j) and learns the j-th bit of the encoding until it detects the termination of the encoding (refer to steps 15-20 of Algorithm 5). To be specific, the j-th bit of the encoding is learned by the agent using the movements in the following order from P (the details of how the agent learns the encoding are described in Algorithm 6).

- Starting from P, move along x-axis until the $(j+1)$-th pebble is reached.
- Rotate at angle $\pi - \arctan(\frac{1}{2j})$, and continue moving in this direction until a pebble is reached.
- Rotate at an angle $\arctan\left(\frac{2j+1}{2j-1}\right)$ and move, until P or a pebble is found.
- If P is found in the previous step, then the bit is 1.
- If a pebble is found, then move along x axis towards P, i.e., rotate at an angle $2\pi - \frac{\pi}{4}$ and move.
- If P is encountered, then the bit is 0.
- If a pebble is encountered instead of P in the previous step, then the agent learns that the encoding is completed.

Algorithm 5: AGENTMOVEMENT

```
if a pebble is found at P then
    angle = π
else
    angle = 0
t = 2, μ = φ // μ indicates the obtained binary string, which at the beginning is
    the empty string φ
while the treasure is not found do
    Start moving at an angle angle with the positive x axis.
    if the treasure is found then
        Terminate
    else
        Move in the same direction until the t-th pebble or the treasure is found.
        if the treasure is found then
            Terminate
        else
            ℓ=FINDBIT(t, angle)
            if ℓ ∈ {0,1} then
                μ = μ · ℓ
                t = t + 1
            else
                FINDTREASURE(μ)
```

Algorithm 6: FINDBIT$(t, angle)$

```
Rotate at an angle π − θ_t, where θ_t = arctan(1/2t) and move until the treasure or a pebble is found.
if the treasure is found then
    Terminate
else
    Rotate at an angle β_t, where β_t = arctan((2t+1)/(2t−1)) and move in this direction.
    if the treasure is found then
        Terminate
    else if P is found then
        Return 1
    else if a pebble is found then
        Rotate at an angle 2π − π/4 and move.
        if P is found then
            Return 0
        else
            Continue its movement until P is reached.
            Return 2
```

A special case occurs, if T is inside B, then on the first iteration itself, i.e., after it reaches the 2nd pebble (i.e., p_2) on x-axis, it rotates at an angle $\pi - \arctan(\frac{1}{2})$ and moves, while moving finds a pebble (i.e., p_3). From this pebble, it rotates at an angle $\arctan(3)$ and moves,

Algorithm 7: FINDTREASURE(μ)

Let n_T be the integer whose binary representation is μ', where $\mu' = \mu \backslash \{\mu_1\}$ and μ_i represents i-th bit of μ.
if $\mu_1 = 0$ **then**
    Update $val = n_T + 1$ and perform SECTORTRAVEL(val, 2)
else
    Update $val = n_T$ and perform SECTORTRAVEL(val, 1)

Algorithm 8: SECTORTRAVEL($val, rotate1$)

Rotate at an angle $\frac{3\pi}{2} + \left(\frac{\pi \cdot val}{2^{k-10}}\right)$ and move until a pebble or treasure is found.
if *Treasure found* **then**
    Terminate.
else
    Rotate at an angle $\pi + (-1)^{rotate1}\frac{\pi}{2}$ and move until a pebble or treasure is found.
    if *Treasure found* **then**
        Terminate.
    else
        Rotate at an angle $\pi + (-1)^{rotate1}\frac{\pi}{2}$ and move until a pebble or treasure is found.
        if *Treasure found* **then**
            Terminate.
        else
            Rotate at an angle $\pi + (-1)^{rotate1}\frac{\pi}{2}$ and move until treasure is found.
            Terminate.

the agent reaches T, and the algorithm terminates (refer to Fig. 3.7).

Otherwise, if T is not reached yet (i.e., T is outside B) then let μ be the binary string learned by the agent in the above process and let n_T be the integer value of the binary string μ', where $\mu' = \mu \setminus \mu_1$, i.e., μ' is a sub-string of μ consisting of the entire binary string except the first bit of μ. Next, as per Algorithms 7 and 8, if the first bit of μ is 1 (resp. 0) and a pebble is found at P (i.e., the initial movement is along $west$ and the treasure is located along the right half of the coordinate axis), then the agent starts moving along L_{n_T} (resp. L_{n_T+1}) from P until it hits a pebble or reaches the treasure. Once a pebble is reached, then the agent rotates at angle $\frac{\pi}{2}$ (resp. $\frac{3\pi}{2}$). It continues to move in the current direction until it hits a pebble or reaches the treasure. Once a pebble is reached, the agent again rotates at an angle $\frac{\pi}{2}$ (resp. $\frac{3\pi}{2}$). It continues to move in the current direction until it hits a pebble or reaches the treasure. If it finds a pebble again, it once again rotates at an angle of $\frac{\pi}{2}$ (resp. $\frac{3\pi}{2}$). It continues to move in this direction until it reaches the treasure.

The following example helps understand the execution of the algorithm.

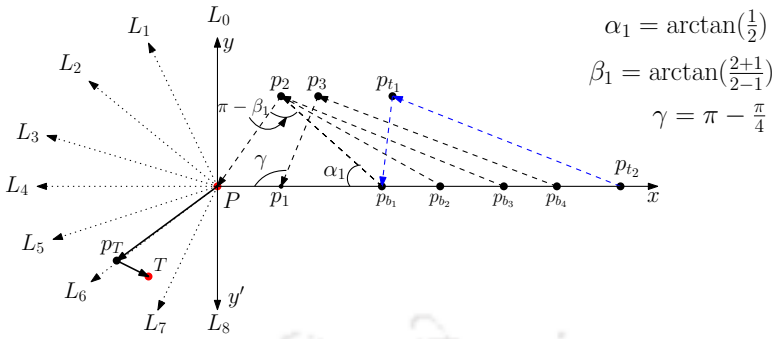


Figure 3.10: Figure showing demonstration of Example

Example: Given 13 pebbles, the Oracle divides the left half plane into 2^{13-10} sectors, where we suppose the treasure is located at sector 6 (as depicted in Fig. 3.10) and it is located outside the square B. Also suppose, the location of F, i.e., the foot of the perpendicular on L_{n_T} (or L_{n_T+1}) is also outside B. So, the Oracle places the pebbles by the PEBBLEPLACEMENT() (Algorithm 1) algorithm in such a way that the agent, after following the algorithm AGENT-MOVEMENT (Algorithm 5), learns the direction of its initial movement and the encoding of the sector number (i.e., 110 in this case) in which the treasure is located. We describe the iteration of the algorithms as follows.

An iteration of Algorithm 5 is defined as a cycle that consists of the agent's movement starting from P and returning to P. In the first iteration, the agent initially at P does not find a pebble at P. Algorithm 5 instructs the agent to move towards the east until it encounters a second pebble p_{b_1} along the positive x-axis. From p_{b_1} the agent rotates at an angle $\pi - \arctan(\frac{1}{2})$ and moves until it encounters a pebble p_2. From p_2 it further rotates at an angle $\arctan\left(\frac{2+1}{2-1}\right)$ and moves until it reaches the origin P. So, after completion of the first iteration (i.e., the path traversed $P \rightarrow p_{b_1} \rightarrow p_2 \rightarrow P$), the agent learns that the first bit is 1. In the second iteration, the agent again moves towards the east until it reaches the third pebble p_{b_2}. From p_{b_2}, the agent rotates at an angle $\pi - \arctan\left(\frac{1}{2\cdot 2}\right)$ and moves until it encounters a pebble p_2, from p_2 it further rotates at an angle $\arctan\left(\frac{2\cdot 2+1}{2\cdot 2-1}\right)$ and moves until it reaches the origin P. So, after completion of the second iteration (i.e., the path traversed $P \rightarrow p_{b_2} \rightarrow p_2 \rightarrow P$), the agent learns that the second bit is 1. In the third iteration, the agent, after a similar movement towards the east, reaches the fourth pebble p_{b_3} along the positive x-axis. From p_{b_3}, it further rotates at an angle $\pi - \arctan(\frac{1}{2\cdot 3})$ and moves until

it reaches the pebble p_2. From p_2, it rotates along an angle $\arctan\left(\frac{2\cdot3+1}{2\cdot3-1}\right)$ and moves until it reaches the origin P. So, after completion of the third iteration (i.e., the path traversed $P \rightarrow p_{b_3} \rightarrow p_2 \rightarrow P$), the agent learns that the third bit is 1. In the fourth iteration, with a similar movement the agent reaches the pebble p_{b_4}, and from this position the agent rotates at an angle $\pi - \arctan\left(\frac{1}{2\cdot4}\right)$ and moves until it reaches p_3, from p_3 it further rotates at an angle $\arctan\left(\frac{2\cdot4+1}{2\cdot4-1}\right)$ and moves until it reaches a pebble p_1. From p_1, the agent finally rotates at an angle $2\pi - \frac{\pi}{4}$ and moves until it reaches P. So, after completion of the fourth iteration (i.e., the path traversed $P \rightarrow p_{b_4} \rightarrow p_3 \rightarrow p_1 \rightarrow P$), the agent learns that the fourth bit is 0. In the fifth iteration, the agent reaches the fifth pebble, i.e., p_{t_2} (refer to Fig. 3.10), from p_{t_2} it rotates at an angle $\pi - \arctan\left(\frac{1}{2\cdot5}\right)$ and moves until it reaches a pebble p_{t_1}, from this position it further rotates at an angle $\arctan\left(\frac{2\cdot5+1}{2\cdot5-1}\right)$ until it reaches a pebble p_{b_1}. Further from p_{b_1}, the agent further rotates at an angle $2\pi - \frac{\pi}{4}$ and moves until it reaches P. Since the agent encounters the pebble p_1 after its last movement from p_{b_1}, this gives the knowledge to the agent that termination is achieved. Hence, the binary string obtained by the agent is $\mu = 1110$. The agent gathers information that the treasure is located somewhere in sector 6 with the help of the binary string μ. Further, since the first bit of μ is 1, the agent then follows the algorithms FINDTREASURE() (Algorithm 7) and SECTORTRAVEL() (Algorithm 8) in order to determine the half-line L_6 along which it travels and encounters the pebble p_T, from which it is further instructed to rotate at an angle $\frac{\pi}{2}$ and move, which ultimately takes the agent to the treasure T, since the point F (i.e., where p_T is placed) is outside B.

3.3.4 Algorithm Analysis

In this section, we prove the correctness and an upper bound on the cost of finding the treasure from the proposed algorithms. The following two lemmas show the algorithm's correctness when the treasure is inside B (where B is the square region bounded by the lines $x = 1$, $x = -1$, $y = 1$ and $y = -1$) and the corresponding upper bound of the cost of treasure hunt in this case.

Lemma 3.3.13. *When the treasure is located inside B, at most four pebbles are used, and the*

agent successfully finds the treasure.

Proof. When the treasure is present inside the square B, the Oracle places a pebble p_0 at P if the treasure is located in the left of y-axis, otherwise, no pebble is placed at P as discussed in case 2 of Section 3.3.2 (also refer to lines $3-4$ and $14-15$ of Algorithm 1). So, the agent starts its movement from P along an angle π, i.e., along negative the x-axis if it finds a pebble p_0 at P (refer to lines 1, 2 and 7 of Algorithm 5) otherwise, the agent moves along an angle 0, i.e., along the positive x-axis if no pebble is found at P (refer to lines 4 and 7 of Algorithm 5). We have the following cases, depending on the presence of a pebble at P.

- *Pebble not found at P*: In this case, the Oracle places pebbles by calling SQUARE-PLACEMENT(2) (refer to line 4 of Algorithm 1). The agent first moves along the positive $x-$axis. If the agent finds the treasure, then the algorithm terminates (refer to lines 8 and 9 of Algorithm 5). Otherwise, the agent finds a pebble p_1 placed at $(1,0)$, which it ignores and continues to move until it reaches the treasure or encounters another pebble (refer to line 11 of Algorithm 5). If the treasure is not found, then the pebble encountered is p_2, which is placed by the Oracle either at h or at $(2,0)$ (refer to lines 10 and 13 of Algorithm 2). The agent, after encountering the second pebble, rotates along an angle $\pi-\theta_1$ and moves, where $\theta_1 = \arctan\left(\frac{1}{2}\right)$. Next, if the treasure does not lie along slope $\tan(\pi - \arctan(\frac{1}{2}))$, then accordingly a pebble (i.e., p_3) is placed, either at s, or at s', where the slope of the line passing through $(2,0)$ and s, or h and s' is $\tan(\pi - \arctan(\frac{1}{2}))$. So, the agent rotating along an angle $\pi-\theta_1$ and moving in this direction, either leads to a pebble or the treasure. If a pebble is found, then we have the following cases:

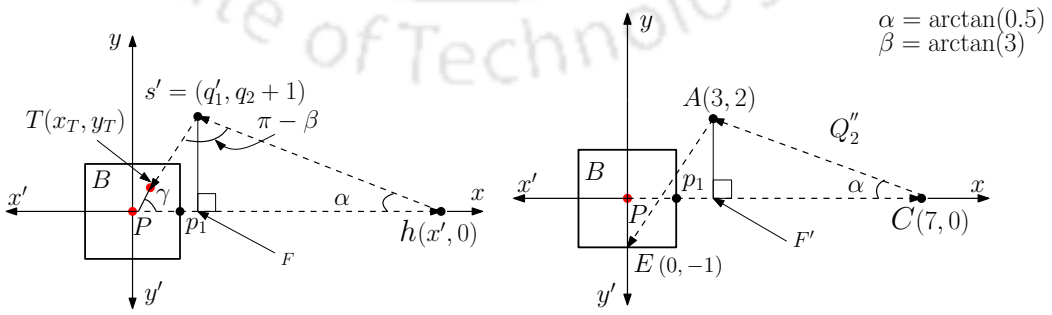
  p_2 is placed at (2,0): In this case, the pebble p_3 is placed at s (where the position of s can be determined in Algorithm 2, and it is such that the slope of the line $p_2s = \tan\left(\pi - \arctan(\frac{1}{2})\right)$ so that whenever the agent rotates at an angle $\pi-\theta_1$ and moves from p_2, it eventually encounters the pebble p_3 at s. Next, the position of s is also significant, as the slope of the line $sT = \tan(\arctan(3) - \arctan(\frac{1}{2}))$. So, as per Algorithm 6, whenever the agent further rotates at an angle $\beta_1 = \arctan(3)$ and moves from s, it gets directly aligned along the line sT. It is because, if we draw a line

parallel to the x-axis from T towards Q_2, term the intersecting point as H, then $\angle HTs = \pi - (\pi - \beta_1) - \theta_1 = \beta_1 - \theta_1$, which is exactly the slope of the line sT (see Fig. 3.7, where β is β_1 here and α is θ_1 here). Hence, moving along this direction, one eventually finds the treasure.

p_2 is placed at h: In this case, the pebble p_3 is placed at s' (where the position of s' can be determined in Algorithm 2, and the slope of the line $hs' = \tan\left(\pi - \arctan(\frac{1}{2})\right)$) so that whenever the agent rotates at an angle $\pi - \theta_1$ and moves from p_2, it encounters the pebble p_3 at s'. Next the position of s' is also significant as, the slope of $s'T = \tan(\arctan(3) - \arctan(\frac{1}{2}))$. So, as per Algorithm 6, whenever the agent further rotates at an angle $\beta_1 = \arctan(3)$ and moves from s', it gets directly aligned along the line $s'T$ as per earlier arguments, and moving along this direction, eventually finds the treasure.

- *Pebble found at P*: In this case, the Oracle places pebbles by calling SQUAREPLACEMENT(1) (refer to line 15 of Algorithm 1). The agent moves along the negative x-axis and performs the exact same movements as performed in the previous case. As per Algorithm 2, in this case, the pebble positions are just mirrored from the earlier case. So, according to Lemma 3.3.2, the agent encounters the same pebbles, as described in the previous case, with the same angular movements. This phenomenon eventually leads the agent to the treasure.

□



(a) Represents the exact path (b) Represents the approximate path

Figure 3.11: Represents the exact and approximate path, when the treasure is inside B

Lemma 3.3.14. *When the treasure is located inside B, the agent starting from P successfully finds the treasure at a cost less than* $7+2(\sqrt{3}+\sqrt{5})$.

Proof. The treasure is located at the point $T=(x_T,y_T)$. Suppose without loss of generality $x_T \geq 0$. Moreover, as $T \in B$, the distance of the treasure from the initial point P is at most $\sqrt{2}$. Now, by the pebble placement strategy, the pebbles p_2 and p_3 are placed at the positions $(2,0)$ (or $h=(x',0)$) and $s=(q_1,q_2)$ (or $s'=(q_1',q_2+1)$), respectively. Now, as per these placements, the agent needs to travel more if p_2 is placed at h and p_3 is placed at s' (refer to Fig. 3.11(a)), respectively, as explained in Algorithm 2. So, in this situation, the agent following Algorithm 5 needs to traverse the path $Ph \rightarrow hs' \rightarrow s'T$ in order to reach the treasure from P. Further, since the treasure can be located anywhere inside B at or to the right half of y-axis, we give an upper bound on the length of the path $Ph \rightarrow hs' \rightarrow s'T$. Let $\alpha=\angle Phs'$, $\beta=\angle hs'P$ and lastly $\gamma=\angle s'Ph$. Observe that since $\gamma=\pi-(\pi-\arctan(3))-\arctan(\frac{1}{2})=45°$, so irrespective of the position $s=(q_1,q_2)$ (only condition $q_2<1$), the x-coordinate of s' can be at most 3, whereas the y-coordinate of s' can be strictly less than 2 (designated as $A=(3,2)$ in Fig. 3.11(b)). It is because, $q_2+1<2$ (since $q_2<1$) and $q_1'=q_1+\frac{1}{tan\gamma}=q_1+1<3$ (since, $q_1<2$ as p_2 is placed at $(2,0)$ and $\angle Pp_2s=\alpha$ and $\alpha<\frac{\pi}{2}$, refer to Fig. 3.7). Now, if we draw a similar line Q_2'' parallel to Q_2' from A (where Q_2' is defined as in Case 2a in Section 3.3.2), then this line meets the x-axis at the point $C=(7,0)$ (refer to Fig. 3.11(b)). Further, we can conclude that $|s'T|<|AE|$ (where $E=(0,-1)$), as $3>q_1'>q_1\geq x_T\geq 0$ and $2>q_2+1>y_T\geq -1$. Also, we can see that $|hs'|\leq|CA|$ (as $\frac{|Fs'|}{|hs'|}=\frac{|F'A|}{|CA|}$, and $|Fs'|<|F'A|$, hence $|hs'|\leq|CA|$, where $F=(q_1',0)$ and $F'=(3,0)$). Note that $|Ph|\leq|PC|$, since each coordinate of p_3's location is less than or equal to the corresponding coordinate of point A, whereas $\angle PCA=\angle Phs'=\alpha<\frac{\pi}{2}$, so the lines $s'h$ and CA are parallel to each other, where s' lies inside the domain bounded by the left portion of the line $x=3$, and the lines $y=1$ and $y=2$. So, we have $|Ph|+|hs'|+|s'T|<|PC|+|CA|+|AE|=7+2(\sqrt{3}+\sqrt{5})$ (as $|PC|=7$, $|CA|=2\sqrt{5}$ and $|AE|=2\sqrt{3}$). So, in this case, the agent in order to reach the treasure traverses a path of length less than $7+2(\sqrt{3}+\sqrt{5})$. □

Lemmas 3.3.15, 3.3.16, 3.3.17, and 3.3.18 show the correctness and complexity of Algorithm 5 when the treasure is outside B.

Lemma 3.3.15. *When the treasure is outside B, the agent successfully learns the j-th bit of the binary string μ at cost $O(j)$.*

Proof. To obtain the j-th bit of μ the movement of the agent is as follows. When the treasure is present outside the square B, the Oracle places a pebble p_0 at P if the treasure is located on or to the right of the y-axis, or otherwise, there is no pebble placed at P (refer to line 6 of Algorithm 1). The movement of the agent from P is as follows:

- p_0 **found at** P: In this case, the Oracle places pebbles by calling NONSQUAREPLACEMENT$(1,0)$ or NONSQUAREPLACEMENT$(1,1)$ (refer to lines 6-12 of Algorithm 1). The agent moves at an angle π, i.e., along the negative x-axis (refer to line 1 of Algorithm 5). Further, it ignores the first j pebbles along the negative x-axis (refer to line 11 of Algorithm 5). It is because the initial value of t is 2 (as the first pebble to encounter is p_1), so $t = j+1$ when determining the j-th bit. After learning each bit, the value of t is incremented, and the invariant $t = j+1$ is maintained. Next, after ignoring the first j pebbles, continues to move until it either finds the treasure or encounters the $(j+1)$-th pebble, i.e., p_{b_j} or p_{t_2} placed at either $(-2j-1,0)$ or $(-2j-2,0)$ or $(-2j-6,0)$ or $(-2j-7,0)$ (refer to Algorithm 3). If the treasure is not found, the cost of reaching this pebble is at most $2j+7$. From the present location with pebble, the agent performs FINDBIT$(j+1,\pi)$ so it rotates at an angle $\pi-\theta_j$ and moves (refer to line 1 of Algorithm 6), where $\theta_j = \arctan\left(\frac{1}{2j}\right)$. Note that, if the agent from p_{b_j} rotates at an angle $\pi-\theta_j$ and moves, then based on the position of p_{b_j}, this will lead the agent either to the treasure, or p_2, or p_3. It is because, if the j-th bit is 1 (or 0), then $p_{b_j} = (-2j-1,0)$ (or $(-2j-2,0)$) and accordingly after this movement, the agent directly aligns along the line connecting p_{b_j} and p_2 (or p_3) with slope $\frac{-1}{2j}$ (=$\tan(\pi-\theta_j)$). By the same argument, if the agent is at p_{t_2}, and rotates in the same sequence of angles and moves and if the treasure is not reached, then it will eventually reach p_{t_1}. If a pebble is found, then this pebble is either p_2 placed at (-1,-1) or p_3 placed at (-2,-1) or p_{t_1} placed at either (-4,-1) or (-5,-1), respectively (refer to line 2 of Algorithm 3). So, the cost of this traversal from the $(j+1)$-th pebble to either p_2 or p_3 or p_{t_1} is at most $\sqrt{(2j+2)^2+1}$. From either of these pebbles, the agent is further instructed to rotate

along an angle β_j and move, where $\beta_j = \arctan\left(\frac{2j+1}{2j-1}\right)$ (refer to line 5 of Algorithm 6) until it encounters the treasure or encounters the pebble p_1, or p_{b_1} or reaches P with $O(1)$ cost. In each case, if the agent rotates from its current position at an angle β_j and moves, it directly aligns itself along the line connecting p_2 with P, or p_3 with p_1, or p_{t_1} with p_{b_1}, having slope $\tan(\beta_j - \theta_j) = \tan\left(\arctan\left(\frac{\frac{2j+1}{2j-1} - \frac{1}{2j}}{1+\frac{2j+1}{2j-1}\cdot\frac{1}{2j}}\right)\right) = 1$ (since $\arctan(a) - \arctan(b) = \arctan(\frac{a-b}{1+ab})$). This is exactly the slope of the lines p_2 with P, or p_3 with p_1, or p_{t_1} with p_{b_1}. Now we have the following cases:

- *If the treasure is found*: In this case, the agent has reached its goal, and the whole process terminates.
- *If a pebble is found*: In this case, the pebble found is either p_1 or p_{b_1}. In either of the cases, the agent is further instructed to rotate along an angle of $2\pi - \frac{\pi}{4}$ and move until it reaches P or pebble p_1 is found (refer to lines 10-16 of Algorithm 6). One of these points will be encountered, as after rotating by an angle of $2\pi - \frac{\pi}{4}$, the agent aligns itself along the x-axis, facing east. It is because, before rotating at an angle of $2\pi - \frac{\pi}{4}$, the agent is facing along a direction that makes a counter-clockwise angle of $\pi - \frac{\pi}{4}$ with respect to the negative x-axis. So, rotating $\pi - \frac{\pi}{4}$, aligns the agent on negative x-axis facing west. Further rotation of π after $\pi - \frac{\pi}{4}$, aligns the agent towards east, facing the origin P. Then, while moving east, it either encounters P or finds a pebble. Hence, we have two cases:
  - *If P is reached*: The agent gains the information that the j-th bit of μ is 0 (refer to lines 15-16 of Algorithm 6 and to lines 15-18 of Algorithm 5). So, the path traversed by the agent is $P \to p_{b_j} \to p_3 \to p_1 \to P$. Hence, the cost is at most $(2j+2) + \sqrt{(2j)^2+1} + O(1) = O(j)$.
  - *If a pebble is found*: In this case, the agent continues to move from p_1 until P is reached with $O(1)$ cost, in which case the agent gains the information that termination of the binary string is achieved, i.e., the $(j-1)$-th bit is the terminating bit of the binary string μ. The agent further moves on to execute algorithm FINDTREASURE() (refer to lines 15-16 of Algorithm 6, as well as lines 15-16 and 19-20 of Algorithm 5). So, the path traversed by

the agent is $P \to p_{t_2} \to p_{t_1} \to p_{b_1} \to p_1 \to P$. Hence, the at most cost is $(2j+7) + \sqrt{(2j+2)^2+1} + O(1) = O(j)$.

– *If P is reached*: In this case, the agent gains the information that the j-th bit of μ is 1 (refer to lines 8-9 of Algorithm 6 and lines 15-18 of Algorithm 5). So, the path traveled to gain this information is $P \to p_{b_j} \to p_2 \to P$. So, the cost is at most $(2j+1) + \sqrt{(2j)^2+1} + O(1) = O(j)$.

- **No pebble is found at** P: In this case, the Oracle places pebbles by calling NONSQUAREPLACEMENT$(2,0)$ or NONSQUAREPLACEMENT$(2,1)$ (refer to lines 17-22 of Algorithm 1). The agent moves at an angle 0, i.e., along the positive $x-$axis (refer to lines 1, 4 and 7 of Algorithm 5). In this case, the pebbles are placed in the following manner: for each pebble placed at (m,n), where $m \neq 0$ or $n \neq 0$ for the case where $x_T \geq 0$ (i.e., for the case where a pebble is found at P), the Oracle in this case, places the corresponding pebble at $(-m,-n)$. So, as per Lemma 3.3.2, with the same angular movements, the agent reaches the corresponding pebbles. Hence, for this case as well, the cost to obtain the j-th bit of the binary string is $O(j)$.

Hence, in each case, the cost of finding the j-th bit of μ is $O(j)$. □

Lemma 3.3.16. *Given k pebbles and the treasure located outside B, the agent successfully learns the binary string μ at cost $O(k^2)$.*

Proof. According to Lemma 3.3.15, the agent successfully determines the j-th bit of μ at $O(j)$ cost. Now as the binary string μ is of length at most k (specifically $k-10$), this implies the total cost to obtain μ is at most: $\sum_{j=1}^{k} O(j) = O(k^2)$. □

Lemma 3.3.17. *When the treasure is located outside B, the agent, after learning the binary string μ, successfully finds the treasure by executing* FINDTREASURE().

Proof. After the execution of the Algorithm 5, the agent performs Algorithm 7 with the already acquired binary string μ to finally reach the treasure $T = (x_T, y_T)$ (if not already reached).

The treasure is either located somewhere in the region $x \geq 0$ (i.e., at or to the right of the y-axis) or $x < 0$ (i.e., to the left of the y-axis) and accordingly, the Oracle divides the whole

left half or right half of the plane into 2^{k-10} sectors (refer to line 1 of Algorithm 1), where each sector S_i is bounded by half-lines L_i and L_{i+1} (where $i \in \{0,1,\ldots,2^{k-10}-1\}$) and the counter-clockwise angle between consecutive half lines is $\frac{\pi}{2^{k-10}}$. Suppose the treasure is located somewhere in sector S_{n_T}, where μ' ($\mu' = \mu \setminus \mu_1$) is the binary representation of n_T. The agent can correctly determine the values of μ_1 and n_T after executing the algorithm AGENTMOVEMENT (Algorithm 5). The whole aim of the Oracle is to align the agent either along the half-line L_{n_T} or L_{n_T+1}. The alignment of the agent along the half-lines L_{n_T} or L_{n_T+1} depends on the first bit value of μ, i.e., on μ_1 (refer to Algorithm 7), where $\mu = \mu_1 \cdot \mu_2 \cdots \mu_{k-9}$. Let F be the foot on the perpendicular on L_{n_T} or L_{n_T+1}, depending on the value of μ_1. Further, let us define $B = \{(x,y) | -1 \leq x \leq 1 \text{ and } -1 \leq y \leq 1\}$.

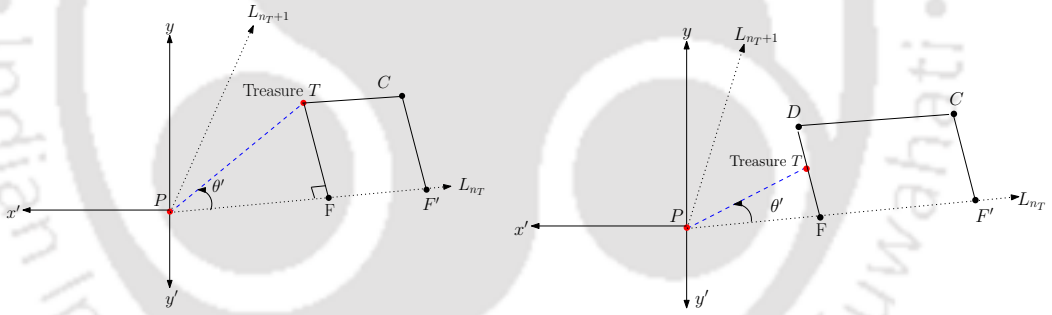
Now, as per the Algorithm 8, the agent is instructed to move either along L_{n_T} or L_{n_T+1}. Without loss of generality, suppose the agent is instructed to move along L_{n_T} (i.e., $\mu_1 = 1$), where $x_T \geq 0$. The other cases follow analogously, as the agent can either move along L_{n_T} or L_{n_T+1}, depending on the value of μ_1, irrespective of whether $x_T \geq 0$ or $x_T < 0$. In the case of the agent moving along L_{n_T+1}, the only change that occurs is the angular rotations performed by the agent after encountering each pebble until the treasure is found. To be specific, while the agent moves along L_{n_T+1}, it needs to rotate at an angle of $\frac{3\pi}{2}$ after it encounters any pebble until the treasure is found. This is unlike the case for moving along L_{n_T}, in which it is required to rotate at an angle of $\frac{\pi}{2}$ instead of $\frac{3\pi}{2}$. So, proving our lemma for L_{n_T}, analogously proves for L_{n_T+1}.

At the first step from P, the agent is instructed to rotate at an angle $\frac{3\pi}{2} + \frac{\pi \cdot n_T}{2^{k-10}}$ (as it is facing east from P, after the terminating iteration of encoding step), which aligns the agent along L_{n_T} (refer to line 1 of Algorithm 8). So, it starts moving along L_{n_T}. If T is on L_{n_T}, then the Oracle places no pebble on L_{n_T}, and the agent finds T. If T does not lie on L_{n_T}, then a pebble p_T is placed either at F or F' (for reference of these points refer Algorithm 4), on L_{n_T}. So, it eventually encounters this pebble p_T. Next, from p_T, the agent is instructed to rotate at an angle $\frac{\pi}{2}$ and move (refer to line 5 of Algorithm 8).

If p_T is at F, where F is the foot of the perpendicular to T on L_{n_T}, then after encountering p_T, the agent rotates at an angle $\frac{\pi}{2}$, which directly aligns it along the line FT. Hence, this counter-clockwise rotation from p_T at an angle $\frac{\pi}{2}$ directly aligns it along the line FT.

Therefore, while moving along this direction, it eventually finds the treasure (Fig. 3.12(a)).

If p_T is at F' (i.e., $F \in B$), then another pebble p_{T1} is placed at a point C towards T, where $\angle PF'C = \frac{\pi}{2}$ (refer to lines 12, 14 and 16 of Algorithm 4 and also Fig. 3.12(a) and 3.12(b)). Therefore, after the agent rotates at an angle of $\frac{\pi}{2}$ from F', it directly aligns itself along the line $F'C$. Next, moving along this line, the agent finds the next pebble p_{T1}. Note that the location of C is significant. Either it is the foot of the perpendicular from T on the perpendicular line from F', or if $|FT| < 1$, then $|F'C| = 1$ (such that $\angle PF'C = \frac{\pi}{2}$) and then another pebble p_{T2} is placed at D (refer to Fig. 3.12(b)), where $\angle F'CD = \frac{\pi}{2}$ and $|CD| = 2$ (refer to lines 13-14 of Algorithm 4). Next, from p_{T1}, again rotating at an angle of $\frac{\pi}{2}$, aligns the agent along the line CT (if $|FT| \geq 1$) or CD. In either case, the agent finds the treasure or encounters the pebble p_{T2} at D. Next, it further rotates at an angle of $\frac{\pi}{2}$, and this eventually aligns the agent along the line DT (it is because $FF'CD$ is a rectangle, where T lies on the line FD) and hence finds the treasure. □



(a) $F \notin B$ or $F \in B$ and $|FT| \geq 1$ (b) $F \in B$ and $|FT| < 1$

Figure 3.12: Traversal of the agent inside the sector when the first bit of μ is 1

Lemma 3.3.18. *When the treasure is located outside B, the agent after learning the binary string μ finds the treasure with cost at most $D(\sin\theta' + \cos\theta')$, where $0 \leq \theta' < \frac{\pi}{2^{k'}}$ and $k' = k - 10$.*

Proof. Suppose the treasure is located at a distance D from the initial point P inside the sector S_{n_T}. Let $\angle FPT = \angle F'PT = \theta'$. The point T must lie between the lines L_{n_T} and L_{n_T+1}, so we can conclude: $0 \leq \theta' < \frac{\pi}{2^{k'}}$, where $k' = k - 10$. Moreover $\angle TFP = \frac{\pi}{2}$ (as F is the foot of perpendicular of T on L_{n_T+1} if $\mu_1 = 0$, otherwise F is the foot of perpendicular of T on L_{n_T}).

The agent, after learning the binary string μ, executes the algorithms AGENTMOVEMENT and FINDTREASURE(), and successfully reaches the treasure by following either the path $PF \to FT$ (if $F \notin B$) or $PF' \to F'C \to CT$ (if $F \in B$ and $|FT| \geq 1$) or $PF' \to F'C \to CD \to DT$ (if $F \in B$ and $|FT| < 1$) from P (refer to Fig. 3.12). We analyze the cost of each of these cases.

1. $F \notin B$: In this case, the agent follows the path $PF \to FT$. We know $|PT| = D$, so $|PF| = D\cos\theta'$ and $|FT| = D\sin\theta'$. So, $|PF| + |FT| = D(\sin\theta' + \cos\theta')$.

2. $F \in B$ and $|FT| \geq 1$: In this case, the agent follows the path $PF' \to F'C \to CT$. Since, $F \in B$, so $|PF| \leq \sqrt{2}$, this implies $|PF'| \leq 2 + \sqrt{2}$ (since $|FF'| = 2$). Recall that $|FT| \geq 1$ and $FF'CT$ is a rectangle, where $|FT| = |F'C|$ and $|FF'| = |CT| = 2$. Since $\angle FPT = \theta'$, so $|FT| = D\sin\theta'$. Hence $|PF'| + |F'C| + |CT| \leq D\sin\theta' + (4 + \sqrt{2})$.

3. $F \in B$ and $|FT| < 1$: In this case, the agent follows the path $PF' \to F'C \to CD \to DT$. Now, by earlier argument, as $F \in B$, hence $|PF'| \leq 2 + \sqrt{2}$. Next, again as $FF'CD$ is a rectangle, where T lies on the line segment FD, moreover, $|FF'| = |CD| = 2$ and $|F'C| = |FD| = 1$, hence $|DT| < 1$. So, we have: $|PF'| + |F'C| + |CD| + |DT| < 6 + \sqrt{2}$. □

Combining Lemmas 3.3.16, 3.3.17, and 3.3.18, we have the final result in Theorem 3.3.19.

Theorem 3.3.19. *Given k pebbles, the agent starting from P successfully finds treasure with cost at most $O(k^2) + D(\sin\theta' + \cos\theta')$, where $0 \leq \theta' < \frac{\pi}{2^{k'}}$ and $k' = k - 10$.*

Remark 3.3.20. Consider the function $f(k) = O(k^2) + D(\sin\theta' + \cos\theta')$, where $0 \leq \theta' < \frac{\pi}{2^{k-10}}$. Note that θ' approaches 0 as k gets large, so $\sin\theta'$ approaches 0 and $\cos\theta'$ approaches 1. Thus, $f(k)$ goes to $o(D) + D$, as $k \to \infty$, which implies that $\frac{f(k)}{D} \to 1$.

3.4 Conclusion

We have shown that it is impossible to find the treasure at a finite cost with the help of a single pebble. Following this, it is shown that it is feasible to find the treasure with the help of two pebbles at a cost of $(\sqrt{2}+2)D+(\sqrt{2}+2)$, where D is the distance of the treasure from the initial position of the agent. Lastly, we have proposed an algorithm for a treasure hunt that finds the treasure in a Euclidean plane using $k \geq 11$ pebbles at cost $O(k^2)+D(\sin\theta'+\cos\theta')$, where $0 \leq \theta' < \frac{\pi}{2^{k-10}}$. Proving a matching lower bound remains an open problem to consider

in the future. Next, a natural question is to determine the optimal cost of the treasure hunt when $2 < k < 11$. Our model considers the assumption that the agent recognizes P whenever it visits it. This assumption helps us to formulate a faster treasure hunt algorithm, i.e., with $k \geq 11$ pebbles. It will be interesting to design a faster treasure hunt algorithm without this assumption. Note that if the agent has some visibility, the problem becomes very trivial even with only one pebble: place a pebble on the line from P to T within a distance of r from P, where r is the visibility radius of the agent. Starting from P, the agent sees the position of the pebble, moves to the pebble, and then continues until it reaches the treasure. However, the problem becomes challenging when the underlying plane has some obstacles. In that case, extending our algorithm would require the agent to efficiently determine the binary string μ while navigating around the obstacles, and the binary string would also need to instruct the agent how to efficiently navigate the obstacles when finding the treasure.

Chapter 4

Black Hole Search in a Dynamic Cactus

4.1 Introduction

In this chapter, we look into the black hole search problem. In general, the black hole search problem (or BHS problem) has been extensively investigated in static graphs, under various communication, scheduler models and in various initial agent configurations. In this chapter, we investigate this problem when the underlying graph is not static but dynamic. We consider our underlying graph to be a synchronous *evolving graph*. These evolving graphs can be thought of as a collection of static graphs, which evolve over time but maintain the underlying characteristic. We only focus on a specific dynamic graph, termed a dynamic cactus. So, to be precise, at each round, some edges can appear or disappear by the adversary, but the underlying characteristic of a cactus graph should be preserved. Before we started this study, the BHS problem had only been investigated in dynamic rings by Di Luna et al. [54, 56]. However, after our study of dynamic cactus, significant studies of this problem have been conducted in other dynamic structures as well, such as in dynamic tori [22] and in dynamic general graphs [85]. We, in this chapter, focus on two variants of dynamicity: first, when only one edge can be dynamic, and second, when k edges can be dynamic. The adversary selects the edge to be made dynamic (by

[4]This chapter has been published as: "Searching for a black hole in a Dynamic Cactus" in *Journal of Graph Algorithms and Applications (JGAA)*, 2025 and in *Proceedings of the 18th International Conference and Workshops on Algorithms and Computation (WALCOM 2024)*.

either reappearing it or disappearing it), and it does so by maintaining the underlying connectivity condition, that at any point, the underlying graph must be connected. Our aim in this chapter is to minimize the number of agents required to perform BHS in these dynamic settings, while also analyzing the complexity of such a BHS algorithm.

The prominent notations used in this chapter are listed in Table 4.1.

Notation	Meaning
$G_i = (V, E_i)$	Indicates the static graph at round i
$\mathcal{G}$	Indicates the dynamic graph
k	Defines the number of at most edges that can be dynamic at any round
n	The cardinality of the set V
$deg(u)$	Maximum degree of u in $\mathcal{G}$
Δ	Indicates the maximum degree in $\mathcal{G}$
A	Indicates the set of all agents
a_i	Indicates the agent with ID i
$Alen_{a_1}$	Variable to store the length of path a_1 has traversed from a certain position
$Apath_{a_1}$	Variable to store the sequence of ports of a path a_1 has traversed from a certain position
$Llen_{a_1}$	Variable that $Leader$ stores to measure the length a_1 has traversed from a certain position
$Lpath_{a_1}$	Variable that $Leader$ stores to track the sequence of ports that a_1 has traversed from a certain position

Table 4.1: Table indicates some of the prominent notations used in this chapter

4.1.1 Our Contribution

The following results are obtained when the cactus graph can have at most one dynamic edge at any round.

1. We establish the impossibility of finding the black hole in a dynamic cactus with 2 agents.

2. We have shown that any black hole search (BHS) algorithm with 3 agents requires at least $\Omega(n^{1.5})$ rounds and $\Omega(n^{1.5})$ moves, where n is the total number of nodes in the underlying dynamic cactus graph. Furthermore, we propose a BHS algorithm that operates with three agents in $O(n^2)$ rounds and $O(n^2)$ moves.

3. Next, with 4 agents we obtain an improved lower bound of $\Omega(n)$ rounds and $\Omega(n)$ moves.

Subsequently, when the cactus graph has at most k ($k > 1$) dynamic edges at any round,

we obtain the following results.

4. We establish the impossibility of finding the black hole with $k+1$ agents.

5. Next, we show that any BHS algorithm with $k+2$ agents requires $\Omega(n^{1.5})$ rounds and $\Omega(n^{1.5})$ moves, respectively.

6. With $2k+3$ agents we give an improved lower bound of $\Omega(n)$ rounds and $\Omega(n)$ moves, respectively.

7. Lastly with $2k+3$ agents, we establish an upper bound of $O(kn)$ rounds and $O(k^2n)$ moves.

Table 4.2 summarises the list of obtained results. In this chapter, all the pseudo-codes of the algorithms are presented in Appendix 4.6.

# DE	# Agents	Moves	Rounds	Reference
1	3	$\Omega(n^{1.5})$	$\Omega(n^{1.5})$	LB (Cor 4.3.2 & Thm 4.3.4)
	3	$O(n^2)$	$O(n^2)$	UB (Thm 4.4.9)
	4	$\Omega(n)$	$\Omega(n)$	LB (Thm 4.3.6)
k (>1)	$k+2$	$\Omega(n^{1.5})$	$\Omega(n^{1.5})$	LB (Cor 4.3.8 & Thm 4.3.9)
	$2k+3$	$\Omega(n)$	$\Omega(n)$	LB (Thm 4.3.10)
	$2k+3$	$O(k^2n)$	$O(kn)$	UB (Thm 4.4.14 & Thm 4.4.15)

Table 4.2: Summary of results, where *LB*, *UB* and *DE* represent Lower, Upper Bound and Dynamic Edge, respectively.

4.2 Model and Preliminaries

Dynamic Graph Model: We consider the dynamic graph to be an *evolving graph*. An *evolving graph* $\mathcal{G}$ is defined to be a sequence of static graphs, where $\mathcal{G} =< G_0, G_1, \ldots, G_r, \ldots >$, and $G_i = (V, E_i)$, such that $E_i \subseteq E_0$. E_0 defines the collection of edges present in G_0 at round 0, where we assume that no missing edge exists. We consider time to be discrete, so a *round* is defined to be a discrete time step. This means that $G_i = (V, E_i)$ is a static graph at round i (where $i \in \mathbb{Z}^+$). The set of vertices V remains fixed, i.e., does not change over time, but the edges can disappear (or, in other terms, go missing) and reappear at any round. Note that

we consider our dynamic graph $\mathcal{G}$ to be *1-interval connected*, which implies that irrespective of which edge (or edges) disappear at any round, our graph must remain connected. In this chapter, our dynamic graph $\mathcal{G}$ is considered to be a dynamic cactus. A *cactus graph* is defined to be a connected graph in which any two simple cycles have at most one node in common. The *footprint* of $\mathcal{G}$ is defined to be the initial cactus graph, i.e., $G_0 = (V, E_0)$. We denote $deg(u)$ as the maximum degree of $u \in \mathcal{G}$. The maximum degree of the graph $\mathcal{G}$ is denoted as Δ. The vertices (or nodes) in $\mathcal{G}$ are anonymous, i.e., unlabeled, although the edges are labeled. An edge incident to u is labeled via the port numbers ranging from $0, \ldots, deg(u)-1$. The ports are labeled uniformly in $\mathcal{G}$ in ascending order along the counter-clockwise direction, where a port with the port number i denotes the i-th incident edge at u in the counter-clockwise direction. Any edge $e = (u, v) \in \mathcal{G}$ is labeled by two ports (refer to the edge (v_{14}, v_{16}) in Fig. 4.1), one among them is incident to u (termed as *outgoing* port of u corresponding to the edge e) and the other incident to v (termed as *incoming* port of v corresponding to the edge e), they have no relation in common. Any number of agents can pass through an edge concurrently. Each node in $\mathcal{G}$ has local storage in the form of a *whiteboard*, the size of the whiteboard at a node $v \in V$ is $O(deg(v)(\log deg(v) + k \log k))$, where k is the maximum number of dynamic edges at any round. The whiteboard is essential to store the list of port numbers attached to a node. Any visiting agent can read and/or write some information corresponding to each port number. Fair mutual exclusion is applied to all incoming agents, restricting concurrent access to the whiteboard. The network $\mathcal{G}$ contains a malicious node termed as *black hole*, which can eliminate any incoming agent without leaving any trace of its existence. The remaining nodes, except the black hole, are termed as *safe nodes*.

Agent: We consider $A = \{a_1, \cdots, a_m\}$, to be the set of $m \leq n$ agents, which are initially co-located at a safe node, termed as $home$. Each agent has a distinct and visible ID of size $\lfloor \log m \rfloor$ bits taken from the set $[1, m]$. We define an agent to be a t-state automata (such that $t \geq \alpha n \Delta \log \Delta$, where $\alpha \in \mathbb{Z}^+$, n is total number of nodes in $\mathcal{G}$ and Δ is the maximum degree of $\mathcal{G}$), having a local storage of $O(n\Delta \log \Delta)$ bits of memory. An agent visiting a node can access the information written on the whiteboard, view the IDs of other agents present at the current node, and can communicate with them. Further, an agent, while traversing

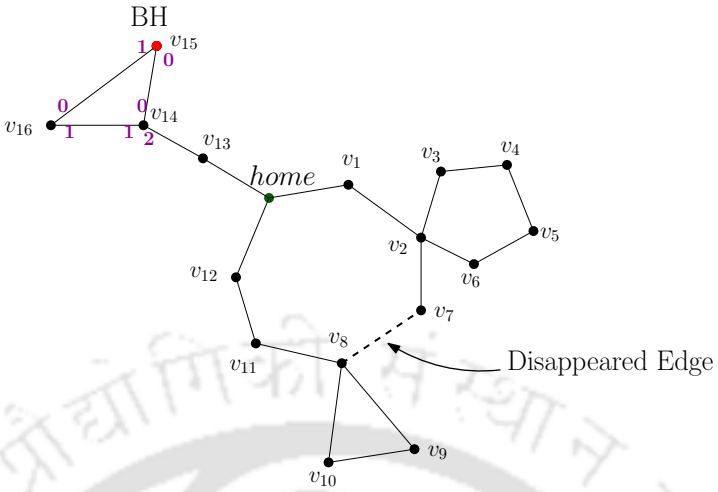


Figure 4.1: An example dynamic cactus graph with the edge (v_7, v_8) disappeared, where the port labeling is indicated on the adjacent edges of v_{14}, v_{15} and v_{16}. BH represents the black hole.

along an edge $e = (u, v)$, knows the incoming port (along which it left u), as well as the outgoing port (the port along which it entered v from u). These agents operate in *synchronous* rounds, where each agent gets activated at each round. After the agents get activated, they perform the "Look-Compute-Move" (LCM) cycle. The steps are defined as follows:

Look: During look phase, an agent takes a local *snapshot* of its current node. This snapshot includes the presence of other agents at the current node, along with their IDs, as well as the contents of the whiteboard present at the current node.

Compute: Based on the contents of its local memory, the snapshot obtained during the look phase and the information gathered from other agents, an agent decides whether to move from its current node. If it chooses to move, the algorithm outputs a direction dir, where if the current node is $u \in \mathcal{G}$, then $dir \in \{0, 1, \ldots, deg(u) - 1\}$. Otherwise, if $dir = nil$, the agent does not perform the Move phase and becomes inactive.

Move: In this phase, the agent chooses the port dir at the current node u, then it traverses along the corresponding edge to reach the adjacent node v, after which it becomes inactive. Moreover, while moving along dir, some information may also be updated on the whiteboard of u, if required.

An agent takes one unit of time, i.e., one round, to move from a node u to another node v following the edge $e = (u, v)$. Since the agents operate in synchronous rounds, each agent

gets activated at each round to perform one LCM cycle synchronously. So, the time taken by the algorithm is measured in terms of *rounds*. Another parameter is *move* complexity, which counts the total number of moves performed by the agents during the algorithm's execution.

We next define the problem definition of this chapter.

Definition 4.2.1 (Problem Definition). Given a dynamic cactus $\mathcal{G}$, an algorithm $\mathcal{A}$ for a set of co-located agents with distinct ID, solves the BHS problem if at least one agent survives and terminates. The terminating agent must either know the exact node of the black hole or have knowledge about the sequence of ports in the footprint of $\mathcal{G}$, such that visiting the last node following this sequence of ports will determine the location of the black hole.

In the following part, we define some specific movements that the agents perform while executing their algorithms.

Cautious Walk [60]: In this movement, it is ensured that while at least two agents move together, exactly one agent can get destroyed by the black hole, and the remaining agent not only survives but also can detect the location of the black hole, provided that the edge between them exists.

Suppose two agents, say a_1 and a_2 are located at u, then the lowest ID agent among them, say a_1, travels to an adjacent node v, yet to be explored by any agent. If a_1 finds v safe, it returns to u, while a_2 waits at u for a_1 to return. If a_1 returns to node u at the next round, both a_1 and a_2 move to v together. Otherwise, if at the next round, a_1 fails to return to u irrespective of the fact that the edge (u, v) exists, a_2 detects v to be the black hole node.

Pendulum Walk [56]: An agent that performs the pendulum walk (say, a_1) travels back and forth, increasing the number of hops after each movement, and always reports back to another agent (which may be referred to as a "witness" agent). More precisely, let us consider that two agents a_1 and a_2 are located at a node u. Now, a_1 (which performs the pendulum walk) decides to move one hop along the edge (u, v) and reaches the node v. If v is safe, a_1 returns to u, which helps a_2 understand that v is safe. Next, a_1 decides to move two hops instead of one, along the edges (u, v) and (v, w), thus reaching the node w. If w is safe, again a_1 returns back to u via v. In general, a_1 reports back to the witness agent after

exploring a new node, which increments the hop count by one.

Marking Walk: This walk is a special case of *cautious* walk. Here, the agent performs a similar movement as explained in *cautious walk*, but unlike *cautious* walk, no other agent is waiting for the exploring agent to return and then move together to the new node.

In this case, an agent a_1 (say) currently at a node u moves one hop to an adjacent unexplored node v, along an edge (u, v). If v is safe, it returns to u, marks the port (u, v) as safe, and in the next round moves to v. In this way, it moves.

4.3 Lower Bound Results

Here we study the lower bound on the number of agents, moves and rounds required to execute a BHS algorithm on a dynamic cactus. First, we study one dynamic edge case, then we study the k dynamic edge case.

4.3.1 Lower Bound Results on Single Dynamic Edge

In this section, we present the lower bound results when at most one edge can be dynamic at any round.

Theorem 4.3.1 (Impossibility for a single dynamic edge)**.** *Given a dynamic cactus $\mathcal{G}$ of size $n > 3$ with at most one dynamic edge at any round, let the agents know that the black hole is located in any of the three consecutive nodes $S = \{v_1, v_2, v_3\}$ inside a cycle of $\mathcal{G}$. Then, it is not possible for two agents to successfully determine the location of the black hole.*

The above theorem is a consequence of Lemma 1 in [56]. Note that the proof of Lemma 1 in [56] falls in line with our BHS algorithm definition. It is because, with 2 agents, the adversary can create a situation where both these agents cannot communicate among themselves since an agent in S is blocked on one side by a missing edge and on the other side by the black hole. Hence, the agent outside S can never determine when the other agent has been destroyed by the black hole or at which node in S it is destroyed. Observe that the proof technique does not require the use of whiteboards, but the result holds even if the nodes are equipped with a whiteboard.

Corollary 4.3.2 (Lower bound on agents for a single dynamic edge)**.** *Any BHS algorithm on a dynamic cactus graph with at most one dynamic edge requires at least three agents.*

Lemma 4.3.3 ([56])**.** *If an algorithm solves BHS with $O(n \cdot f(n))$ moves with three agents, then there exists an agent that explores a sequence of at least $\Omega(\frac{n}{f(n)})$ nodes such that:*

- *The agent does not communicate with any agent while exploring any node in the sequence.*
- *The agent visits at most $\frac{n}{4}$ nodes outside the sequence while exploring any node in the sequence.*

The proof of the above lemma does not incorporate the use of whiteboards, but this lemma holds even if the nodes are equipped with a whiteboard, as stated in [56].

In the following theorem, we give a lower bound on the move and round complexities required by any BHS algorithm operating with three agents on a dynamic cactus graph with at most one dynamic edge at any round.

Theorem 4.3.4. *Given a dynamic cactus $\mathcal{G}$, where there can be at most one dynamic edge at any round, any BHS algorithm operating with three agents requires $\Omega(n^{1.5})$ rounds and $\Omega(n^{1.5})$ moves.*

The above theorem is a consequence of [56, Theorem 6], which uses Lemma 4.3.3. The following observation gives a brief idea about the movement of the agents on a cycle inside a dynamic cactus. It states that when a single agent is trying to move along a cycle, the adversary can confine the agent on any single edge of the cycle. In the case of multiple agents trying to move along a cycle inside a cactus graph, if their movement is along one direction, i.e., either clockwise or counterclockwise, the adversary can prevent the agents from visiting further nodes inside the cycle.

Observation 4.3.5. Let $\mathcal{G}$ be a dynamic cactus with a cut U (where $|U| > 1$), such that the footprint of U is connected with $V \setminus U$ by the edges e_1 and e_2 in the clockwise and counterclockwise directions, respectively. In this setting, if we assume that all the agents at round r are present at the nodes in U, and that they attempt to cross to $V \setminus U$ only via the edge e_2

and not e_1, then in this scenario the adversary may prevent the agents from visiting a node outside U.

The above observation follows from [56, Observation 1], where a *cut* is defined to be a partition of the vertices of the graph into two disjoint subsets. The next theorem gives an improved lower bound on the move and round complexity when 4 agents execute a BHS algorithm instead of 3.

Theorem 4.3.6. *Any BHS algorithm with 4 agents requires* $\Omega(n)$ *rounds and* $\Omega(n)$ *moves on a dynamic cactus graph* $\mathcal{G}$*, in the presence of at most one dynamic edge at any round.*

Proof. Let a_1, a_2, a_3 and a_4 be the four agents that are assigned to detect the black hole in $\mathcal{G}$. Suppose these agents execute a BHS algorithm $\mathcal{H}$. Let us consider $\mathcal{H}$ terminates within $o(n)$ rounds in the presence of a single dynamic edge.

To contradict this claim, we consider an instance cactus graph $\mathcal{G}$ of n nodes, which consists of a single cycle, denoted as C'_1, where C'_1 has $n-1$ nodes (refer to Fig. 4.2). Let the black hole be somewhere in the cycle C'_1, (in Fig. 4.2, the node y_1 depicts the black hole), and suppose that without loss of generality, a_1 is the first agent to get destroyed by the black hole while moving clockwise in C'_1. Let Q be the set of $O(1)$ many consecutive counter-clockwise nodes to the black hole in C'_1 (where $|Q|$ or the cardinality of Q is at least 1) along which a_1 has written the exact location of the black hole or the sequence of ports that leads to the black hole before it got destroyed. As by hypothesis, each node contains a whiteboard, hence, it is possible for a_1 to write this information. This phenomenon implies that whenever some agent (except a_1) visits at least one node in Q, it understands the exact location of the black hole. Accordingly, $\mathcal{H}$ gets terminated (refer to Definition 4.2.1). Let e_q be the edge separating the black hole and the sector Q from the remaining nodes in C'_1. Now, as per Observation 4.3.5, if the remaining agents need to locate the black hole in C'_1, then at least one agent needs to traverse along C'_1 in a counter-clockwise direction, and at least one in a clockwise direction. So, the only possibility remains: while an agent always tries to visit a node in Q (it cannot do so until the adversary keeps the edge e_q missing), the remaining two agents can correctly locate the black hole location while traversing in a counter-clockwise direction along C'_1. This shows that in at least $|C'_1| - |Q|$ $(= n-1-O(1))$

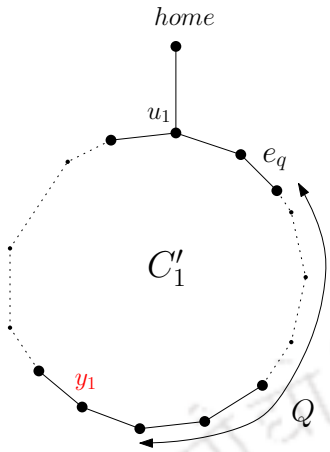


Figure 4.2: A cactus graph, with a cycle C'_1 with $n-1$ nodes, where the red node is the black hole location.

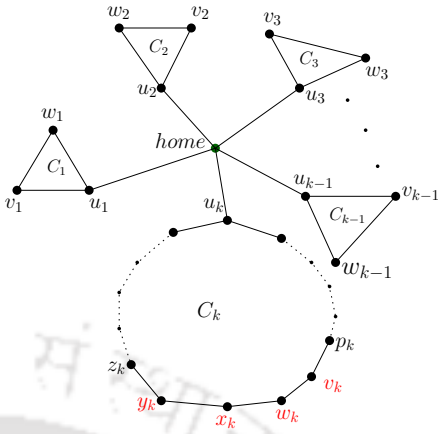


Figure 4.3: A cactus graph, with k different cycles, where the red nodes in C_k depicts possible locations of the black hole.

rounds, one among these three remaining agents detects the black hole location, which contradicts our claim that $\mathcal{H}$ terminates within $o(n)$ rounds. Moreover, at any round, a constant number of agents are moving, so to successfully locate the black hole location with four agents, any algorithm requires $\Omega(n)$ rounds and $\Omega(n)$ moves. □

4.3.2 Lower Bound Results for Multiple Dynamic Edges

In this section, we present the lower bound results when, at most, k ($k > 1$) edges are dynamic.

Theorem 4.3.7 (Impossibility for multiple dynamic edges)**.** *It is impossible for $k+1$ co-located agents to successfully locate the black hole position in a dynamic cactus $\mathcal{G}$ with at most k dynamic edges at any round.*

Proof. We prove the above statement by contradiction. Let us consider a dynamic cactus $\mathcal{G}$ (refer to Fig. 4.3) of n vertices, in which at most k edges are dynamic at any round. Let us consider $\mathcal{G}$ contains k cycles, denoted by C_i where $i \in \{1, 2, \cdots, k\}$, in which, except for the last cycle C_k, which is of length $n+2-3k$, every other cycle is of length 3. Let $\mathcal{H}$ be a BHS algorithm, which successfully terminates with a set of $k+1$ agents. Each agent is initially at *home*, and suppose after following the algorithm $\mathcal{H}$, the agents enter a configuration

where an agent a_i reaches a node v_i or w_i inside C_i ($i \in \{1,2,\cdots,k-1\}$), and the remaining two agents enter the cycle C_k. Suppose the black hole is located at any one among the four consecutive nodes $S = \{v_k, w_k, x_k, y_k\} \in C_k$, of which the agents have no idea. Since the adversary can disappear and reappear at most k edges at any round, hence, it can restrict each a_i inside C_i by alternating its position between v_i and w_i, by removing the edge (u_i, v_i) and (u_i, w_i) alternatively (where $i \in \{1,\cdots,k-1\}$). So, the remaining agents a_k and a_{k+1} have no choice but to explore C_k. They cannot explore a node in S together for the first time, as the adversary may place the black hole at that node, eliminating both of them, whereas the other agents have no idea of the location of the black hole as they are unable to come out of C_i ($i \in \{1,\cdots,k-1\}$). So let us consider an agent a_k (say) is the first to enter S, i.e., at a node v_k at some round r, or both agents enter a node in S at round r (suppose a_k visits v_k and a_{k+1} visits y_k). At this point, regardless of the position of a_{k+1}, the adversary removes the edge (v_k, p_k). Now, in any case, a_k cannot communicate with a_{k+1} even in the presence of a whiteboard. It is because on one side, there is a black hole, and on the other side, there is a missing edge. If v_k and w_k are safe nodes, a_k has no other option but to visit x_k at some round. In the meantime, each of the remaining agents a_i are stuck inside C_i ($i \in \{1,\cdots,k-1\}$), respectively.

If x_k is the black hole node, a_k gets destroyed. On the other hand, a_{k+1} has no other option but to eventually reach x_k, as it has no idea about a_k's destruction at x_k. So, whenever a_{k+1} reaches x_k, it also gets destroyed. Now, after finding that both of these agents have failed to return, the remaining agents can only guarantee that at least one among them is destroyed by the black hole. But they cannot guarantee which among the nodes x_k, w_k or v_k is indeed the black hole. It is because if x_k is safe, a_k has already been eliminated. So, whenever a_{k+1} reaches x_k, the adversary can reappear the edge (v_k, p_k) and disappear the edge (y_k, x_k). This restricts a_{k+1} to come out of S and communicate with other agents, with the help of a whiteboard, that x_k is safe. Hence, in any case, the remaining $k-1$ agents cannot terminate the algorithm, as they have no idea which of these three nodes is indeed the black hole node. This leads to a contradiction. □

Corollary 4.3.8 (Lower bound on agents for k dynamic edges)**.** *Any BHS algorithm operat-*

ing on a dynamic cactus with at most k dynamic edges at any round requires at least $k+2$ agents.

The following two theorems provide lower bounds on the complexity of any BHS algorithm with $k+2$ and $2k+3$ agents, respectively.

Theorem 4.3.9. *Any BHS algorithm operating on $\mathcal{G}$ with $k+2$ agents require $\Omega(n^{1.5})$ rounds and $\Omega(n^{1.5})$ moves, where $\mathcal{G}$ is a dynamic cactus with at most k dynamic edges at any round.*

Proof. We prove the above statement by contradiction. Let us suppose $\mathcal{H}$ be a BHS algorithm that works with $k+2$ agents within $o(n^{1.5})$ rounds. Now, consider the same instance graph $\mathcal{G}$ (refer to Fig. 4.3) of k-cycles, where $|C_i| = 3$ (for all, $1 \leq i \leq k-1$) and $|C_k| = n+2-3k$. The set of $k+2$ agents $A = \{a_1, a_2, \cdots, a_{k+2}\}$ are initially co-located at $home$. Suppose, while executing the algorithm $\mathcal{H}$, they enter a configuration in which a_i gets stuck inside C_i (for all, $1 \leq i \leq k-1$), whereas the remaining agents, i.e., a_k, a_{k+1} and a_{k+2} enter C_k. Now, by Theorem 6 in [56], a set of 3 agents require $\Omega((n+2-3k)^{1.5}) = \Omega(n^{1.5})$ rounds (since $k < \frac{n}{3}$) to correctly locate the black hole inside C_k. Note that in C_k, at most, one edge can be dynamic, similar to a dynamic ring of size $n+2-3k$. Hence, this leads to a contradiction to the fact that $\mathcal{H}$ locates the black hole in $o(n^{1.5})$ rounds. Moreover, a constant number of agents move while exploring C_k, whereas to enter this configuration starting from $home$, at least $2k$ moves are required. So, this shows that at least $\Omega(n^{1.5}+2k) = \Omega(n^{1.5})$ moves are required. This proves the theorem. □

Theorem 4.3.10. *Any BHS algorithm operating on a dynamic cactus $\mathcal{G}$ with $2k+3$ agents requires $\Omega(n)$ rounds and $\Omega(n)$ moves, where at any round at most k edges can be dynamic.*

Proof. We have proved the above statement by contradiction. Suppose a BHS algorithm $\mathcal{H}$ exists, which determines the location of a black hole in $o(n)$ rounds. Let $\mathcal{G}$ be the same graph (refer to Fig. 4.3) with k-cycles, where $|C_i| = 3$, for all $1 \leq i \leq k-1$ and $|C_k| = n+2-3k$, the agents are initially co-located at $home$. Now again, suppose the agents executing $\mathcal{H}$ enter a configuration where each a_i gets stuck in C_i, for all $1 \leq i \leq k-1$, then in this situation, the remaining $k+4$ agents try to explore C_k. Next, by Theorem 4.3.6, we know that it takes four agents among the $k+4$ agents to successfully locate the black hole in

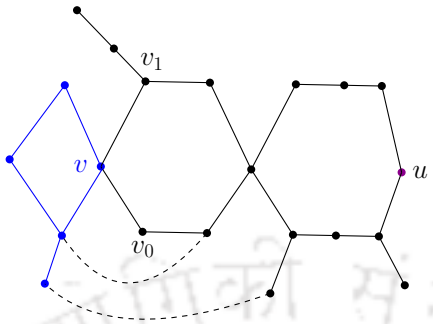


Figure 4.4: Represents a cactus graph, where blue vertices belong to *Half-2* and rest in *Half-1*. Dashed edges represent the edges from *Half-1* to *Half-2*.

$\Omega(n+2-3k) = \Omega(n)$ (where $k < \frac{n}{3}$) rounds. Hence, this leads to a contradiction. Moreover, the bound on the number of moves comes from at least $2k$ additional moves that are required to attain this configuration. On the other hand, a constant number of agents move to explore C_k. Hence, this shows that any BHS algorithm requires $\Omega(n)$ rounds and $\Omega(n+2k) = \Omega(n)$ moves. □

The next lemma follows from the structural property of a cactus graph.

Lemma 4.3.11. *Given two nodes u and v in a cactus graph G, there exists at most two nodes adjacent to v such that their removal disconnects u from v.*

Proof. To prove this claim, we have considered two cases: the node v is part of a cycle in G, and v is not part of a cycle.

1. The node v is part of a cycle C in G. Let v_0 and v_1 be the two adjacent nodes of v (refer to Fig. 4.4), such that there exists at least one path from u to v_0 and at least one path from u to v_1, which do not contain v. In this context, we have contradicted that after removing v_0 and v_1, the node v remains connected to u. We define *Half-1* to be the collection of nodes in G such that for each node w in *Half-1*, there exists at least one w to v_0 or w to v_1 path which does not pass through v. The remaining nodes belong to *Half-2*. Now, u belongs to *Half-1*. After the removal of v_0 and v_1, the cycle C gets disconnected. So, if any path exists from a node in *Half-1* to v, that path uses at least one edge that is not part of C. This violates the definition of the cactus graph, as in the original graph, i.e., in the presence of

C, there exists at least one edge, which is common between two cycles. This proves that the removal of v_0 and v_1 disconnects u from v.

2. The node v is not part of any cycle in G. Let v_0 be the adjacent node of v, where at least one path exists from u to v_0 that does not contain v. In this context, we contradict that after the removal of v_0, the node v still remains connected to u. We have defined *Half-1* as the set of nodes in G such that for each node w in *Half-1*, there exists at least one w to v_0 path which does not contain v. So, u belongs to *Half-1*. After removal of v_0, if still there exists a path from *Half-1* to v, then that implies there exists an alternate path to reach v, which does not pass through v_0. This contradicts the fact that in the original graph, v is not part of any cycle. □

The next corollary follows from Lemma 4.3.11.

Corollary 4.3.12. *Suppose a node v in a cactus graph G is part of a cycle, then there exists a set of three consecutive nodes $\{v_0, v, v_1\}$, such that any path from u to v in G must either pass through v_0 or v_1.*

4.4 Black Hole Search in Dynamic Cactus

In this section, we first present a BHS algorithm that works in the presence of at most one dynamic edge. Subsequently, we present another BHS algorithm that works in the presence of at most k (> 1) dynamic edges. Accordingly, we analyze the correctness and find the move and round complexities required by each algorithm.

4.4.1 Black Hole Search in Presence of Single Dynamic Edge

Here, we present a BHS algorithm that operates in the presence of at most one dynamic edge. Our algorithm requires three agents, namely $A = \{a_1, a_2, a_3\}$. In this section, we always call a_3 the $Leader$, and by agents, we only mean a_1 and a_2. Our BHS algorithm comprises two parts, one for the agents (i.e., a_1 and a_2) and the other for the $Leader$. The BHS algorithm executed by the agents is SINGLEEDGEBHSAGENT, whereas SINGLEEDGEBHSLEADER is for the $Leader$. Each of our algorithms intricately uses the whiteboard at each

node. Before discussing the algorithm idea, we first define all the contents of a whiteboard required while executing the BHS algorithm.

A whiteboard is maintained at each node $v \in \mathcal{G}$, where a list of information is used by both the agents and the $Leader$. Formally, for each port j of a node $v \in \mathcal{G}$, where $j \in \{0, \cdots, deg(v)-1\}$, an ordered tuple $(f(j), \texttt{Last.Leader})$ is stored, where the function f is defined as follows: $f : \{0, \ldots, deg(v)-1\} \longrightarrow \{\perp, 0, 1\}^*$,

$$f(j) = \begin{cases} \perp, & \text{if the port } j \text{ is yet to be visited by any agent} \\ 0 \circ A, & \text{if the set of agents in } A \text{ has visited } j \text{ but yet} \\ & \text{to explore the sub-graph originating from } j \\ 1, & \text{if the sub-graph originating through } j \text{ is fully explored by} \\ & \text{at least one agent and no agent is stuck along that sub-graph} \end{cases}$$

The symbol " $\circ$ " refers to the concatenation of two binary strings. We define A as the collection of agents that have visited the port j. For example, let us assume a_2 with ID 2 (i.e., 10) is the only agent to visit the port j, and it is yet to explore the sub-graph originating from j. In that case, the function $f(j)$ returns the value 010. The first bit represents that the sub-graph originating from the port j is yet to be fully explored by at least one agent. The remaining bits of $f(j)$ represent the ID of a_2.

The other entity, i.e., the `Last.Leader` at the node v stores the information about the last movement of the $Leader$ at v. To be precise, `Last.Leader` = 1 for the j-th port means that it is the port along which the $Leader$ must have moved from v, the last time it visited v. For the remaining ports at v, `Last.Leader` is set to be 0.

Before discussing SINGLEEDGEBHSAGENT and SINGLEEDGEBHSLEADER, we, in the following part, give a brief idea about the protocol an agent performs while executing their respective algorithms. Each agent (i.e., a_1 and a_2) executes the protocol of t-INCREASING-DFS [70] while exploring the graph $\mathcal{G}$. The t-INCREASING-DFS protocol helps the agent choose the next port. Additionally, this protocol enables the agent to return to the starting node. An agent stores each new port it visits from the starting node and stops whenever it

exceeds t-bits (refer to Section 4.2 for the value of t). While performing this protocol, the movement of the agents can be categorised into two main types: *explore* and *trace*.

(a): An agent performs *explore* when it explores a port that is yet to be visited by any other agent, i.e., if that port is j, then $f(j)$ is marked as $\perp$. An agent can only perform either *cautious* or *pendulum* walk on such ports (as instructed by the $Leader$) while trying to explore j.

(b): An agent a_1 (say) can perform *trace* when it visits a port, which a_2 has visited but not a_1. Additionally, the subgraph originating from this port has not yet been fully explored, i.e., it is still not marked as 1 on the whiteboard. An agent only performs *pendulum* walk along these ports.

Next, the task of the $Leader$ is explained as follows:

(a): It can instruct a_1 or a_2 to perform *cautious* or *pendulum* walk.

(b): It maintains certain variables for the agents a_1 and a_2, which help the $Leader$ to understand how far an agent a_1 (say) has traversed and along which path. These variables are defined to be $Alen_{a_1}$ and $Apath_{a_1}$ for a_1.

(c): It also maintains some more variables, which track how far the $Leader$ has traversed from a certain agent a_1 (say), and along which path. These variables are defined to be $Llen_{a_1}$ and $Lpath_{a_1}$ for a_1.

An agent can *fail to report* to the $Leader$ if either of these conditions holds:

(a): The agent is destroyed by the black hole.

(b): The agent, while traversing along a specific direction, encounters a missing edge.

We now provide a brief overview of our BHS algorithms.

Brief Idea of the Algorithms: Each agent starts from $home$, where we assume the ID of a_1 < ID of a_2 < ID of a_3. The agent a_3 is elected as the $Leader$ at $home$. After being elected $Leader$, a_3 executes the algorithm SINGLEEDGEBHSLEADER, whereas the agents a_1 and a_2 execute SINGLEEDGEBHSAGENT. On a high level, the $Leader$ instructs both agents to perform either one among the two walks: *cautious* or *pendulum* walk. Based on their movements, the $Leader$ stores the path and the distance these agents have traversed. It also stores the path that it itself traverses away from its previous position. Whenever the $Leader$ realises that any agent *fails to report*, it performs the following task. If it finds both

a_1 and a_2 *fails to report*, the $Leader$ understands that the black hole has destroyed at least one among them. Next, within a finite round, it either detects the exact black hole position or concludes the sequence of paths that it needs to traverse to eventually locate the black hole. Otherwise, if any one agent *fail to report*, in that case, the $Leader$ traverses towards that agent following the stored path. If the agent is found, that agent is again instructed to perform a particular movement. Otherwise, if the agent is not found and the $Leader$ encounters a missing edge along its traversal, the $Leader$ waits until the missing edge reappears. On the other hand, if it neither finds the agent nor any missing edge, the $Leader$ concludes the black hole position. Both algorithms terminate once the $Leader$ locates the black hole position or finds the exact sequence of ports to visit in order to locate the black hole. Next, we give a detailed description of our two algorithms.

Detailed Description of SINGLEEDGEBHSAGENT: This algorithm is executed by a_1 and a_2 autonomously. Initially, the $Leader$ (which is executing SINGLEEDGEBHSLEADER) instructs a_1 (since it is the lowest ID agent) to perform *cautious* walk and a_2 to perform *pendulum* walk. Consider that at the beginning, every port corresponding to each node in the network is marked $(\perp, 0)$, as any agent or $Leader$ is yet to visit these nodes. The agents choose a port based on the protocol of $t-$INCREASING-DFS (where $t \geq \alpha n\Delta \log\Delta$, and $\alpha \in \mathbb{Z}^+$), which helps them select the next port.

So an agent, a_1 (say), executes the following movement after being instructed to perform *cautious* walk. Since the *cautious* walk is performed with a following agent, in each scenario, that following agent is the $Leader$. Let us assume at round r (≥ 0), a_1 is with the $Leader$ at a node $u \in \mathcal{G}$. Then at round $r+1$, a_1 first checks whether any port exists at u, which is marked as $\perp$. If so, then a_1 chooses the lowest port among them, say that port is i, and moves along it to an adjacent node v (say). Let m be the incoming port at v along which a_1 reaches v from u. The agent not only moves to v from u through port i but also writes the data $0 \circ \{a_1\}$ with respect to port i at u and port m at v. This helps others gather the information that a_1 has already visited the nodes u and v, taking the ports i and m, respectively. This also helps others understand that the subgraph originating from port i or m is not yet fully explored, or that a_1 is stuck somewhere along the subgraph originating from port i or m.

Suppose v is safe and the edge (u,v) exists at round $r+2$, then a_1 returns to u from v to meet with the $Leader$ at round $r+2$. Next, at round $r+3$, if the edge (u,v) exists, a_1 accompanies the $Leader$ along with it to the next node v. But on the contrary, at round $r+1$, if a_1 is unable to find any port at u marked as $\perp$, then in this round itself, a_1 accompanies the $Leader$ and backtracks to an adjacent node, already visited by a_1 at any previous round. This is how a_1 executes *cautious* walk.

If an agent, a_1 (say), is instructed to perform *pendulum* walk, it performs the following movement. If we consider round r, a_1 is with $Leader$ at a node $u \in \mathcal{G}$, then at the next round, it checks if at least one port at u is marked as $\perp$. If so, then it chooses the lowest port among them and stores the port number in its internal memory. After which it moves along this port to an adjacent node v (say) while writing $0 \circ \{a_1\}$ both at the outgoing port of u (the port along which it traverses towards v) and at the incoming port of v (the port along which it reaches v from u). If there is no such port marked as $\perp$, but at least one port is marked as $0 \circ \{a_2\}$, a_1 chooses the lowest among these ports and stores the port number. Next, it moves to the adjacent node v (say) following this stored port. While moving, it updates the whiteboard information to $0 \circ \{a_1, a_2\}$, both at the outgoing port of u and the incoming port of v. In the next round, considering this adjacent node to be safe and the edge (u,v) to be present, a_1 returns to u and meets with $Leader$, conveying the path it has traversed in order to reach v. After which, at round $r+3$, again considering the edge (u,v) to be present, a_1 moves alone to v. At round $r+4$, from v, it reaches some new adjacent node w (based on the existence of such a port, not marked 1 or $0 \circ \{a_1\}$, and also the corresponding edge must not disappear), this information is updated by a_1. If w is safe, then at round $r+5$, it returns to v and at round $r+6$, it returns to the $Leader$ at u and conveys this extended path to the $Leader$, provided that all these in between edges have not gone missing. In this way, the agent executes the *pendulum* walk. Note that, irrespective of which walk an agent performs, whenever it encounters a missing edge along its chosen direction of movement, it immediately stops at the adjacent node of that missing edge until further instruction is provided to the agent or the missing edge reappears.

Detailed Description of SINGLEEDGEBHSLEADER: This algorithm is executed by a_3 once it is elected as the $Leader$. Initially co-located with other agents at $home$, it instructs the

lowest ID agent, i.e., a_1, to start performing *cautious* walk, and instructs a_2 to perform *pendulum* walk. The $Leader$ maintains certain variables for the agent currently performing *pendulum* walk. In this case, we have a_2 performing *pendulum* walk currently, hence the variables which the $Leader$ maintains are: $Alen_{a_2}$ and $Apath_{a_2}$. The variable $Apath_{a_2}$ keeps track of the sequence of ports that a_2 has explored while performing *pendulum* walk, whereas $Alen_{a_2}$ stores the cardinality of $Apath_{a_2}$. Each time a_2 returns to the $Leader$ after exploring a new node, the $Leader$ increments $Alen_{a_2}$ by 1, and updates $Apath_{a_2}$. Thereafter, if the $Leader$ moves away from the node where it last met with an agent, the $Leader$ keeps track of the path it has traversed away from that node. The variables which keeps track about the $Leader$'s movement are: $Llen_{a_2}$ and $Lpath_{a_2}$, for an agent a_2 (say) from which it has moved away. The variable `Last.Leader` is used by the $Leader$ to help an agent reach the current position of $Leader$. This entity is updated by the $Leader$ at the whiteboard of each node that it visits. To be accurate, if at round r, both $Leader$ and an agent a_2 (say) met at a node u, and after which $Leader$ decides to move away from u to a node v. In that case, the $Leader$ at round $r+1$ before reaching v, updates `Last.Leader` at the whiteboard of u to 1 for the port j (if the $Leader$ has taken the j-th port to reach v from u) and the rest of the ports at u to 0. Next, after reaching v, it updates $Lpath_{a_2}$ to $Lpath_{a_2} \cup (j,m)$ and increments $Llen_{a_2}$ by 1, where j and m indicates the port of the edge (u,v). Next, we discuss the scenarios that may occur for the $Leader$ while executing this algorithm.

Scenario-1: An agent, say a_2, *fails to report* when it is performing *pendulum* walk, while the other agent is performing *cautious* walk.

In this scenario, $Leader$ can understand a_2's failure to return, only when it finds a_2 does not report within $(2 \cdot (Alen_{a_2} + Llen_{a_2} + 1))$ rounds, since they last. After this realization, whenever a_1 meets the $Leader$, it instructs a_1 to change its movement from *cautious* to *pendulum* walk. After this, the $Leader$ starts moving towards a_2 and while moving, it updates the respective stored variables (so that a_1 can eventually find the $Leader$). The purpose of $Leader$'s movement towards a_2 is to check about a_2, i.e., whether it is stuck (or waiting) due to a missing edge or has been destroyed by the black hole. The $Leader$ takes help from the stored variables: $Lpath_{a_2}$ and $Apath_{a_2}$ to reach the last reported node of a_2.

While moving, the $Leader$ may face the following possibilities: $Leader$ may encounter a missing edge, or it may not find a_2, or it may find a_2 stuck (or waiting) for a missing edge to reappear. If the $Leader$ encounters a missing edge, then it waits until the missing edge reappears. Otherwise, if it does not find a_2 but encounters no missing edge, then it concludes that the adjacent node along which a_2 last traversed from the last reported node is the black hole, which terminates the algorithm. Lastly, the $Leader$ may find a_2 stuck (or waiting) for a missing edge. In this case, if the edge is still missing, then the $Leader$ sets a_2 free by instructing it to continue performing *pendulum* walk along alternate ports, whereas the $Leader$ waits for the missing edge to reappear. This shows that while the $Leader$ waits for the missing edge to reappear, both a_1 and a_2 perform *pendulum* walk. On the contrary, if the missing edge reappears, the moment $Leader$ finds a_2, then it asks a_2 to start performing *cautious* walk (note, earlier a_2 has been performing *pendulum* walk). So, a_2 performs *cautious* walk, but a_1 continues to perform *pendulum* walk.

Scenario-2: An agent, say a_1 fails to report to the $Leader$ when it is performing *cautious* walk, while the other agent is performing *pendulum* walk.

Before a_1 has failed to report, let us suppose both a_1 and $Leader$ have been at a node u, and then a_1 has visited an adjacent unexplored node v. If the failure for a_1's return occurs when the edge (u,v) still exists, then the $Leader$ being present at the adjacent node concludes that v is the black hole. On the other hand, if the failure of a_1's return is due to the edge (u,v), which has gone missing, then the $Leader$ waits at u until the edge reappears. In between all this, until $Leader$ instructs a_2 to change its movement, it continues to perform *pendulum* walk.

Scenario-3: Both a_1 and a_2 fails to report, while a_1 and a_2 are performing *cautious* walk and *pendulum* walk, respectively.

Without loss of generality, if a_1 fails to report before a_2, that failure of a_1's return is triggered by the missing edge between the $Leader$ and a_1. Otherwise, if the edge has existed and still a_1 fails to return, then $Leader$ must have already concluded the black hole node and terminated the algorithm. So, while the $Leader$ waits for the missing edge to reappear, it finds that after waiting for $(2\cdot(Alen_{a_2}+Llen_{a_2}+1))$ rounds since it last met a_2, a_2 is yet to return. As this is a sufficient number of rounds for a_2 to return, $Leader$ understands that

a_2 also fails to return. Subsequently, the $Leader$ concludes that a_2 must have been destroyed by the black hole and terminates the algorithm by concluding the last visited node of a_2 to be the black hole position. Note that the black hole can be found after eventually reaching the last reported node of a_2 following the sequences $Apath_{a_2}$ and $Lpath_{a_2}$, and then checking the whiteboard of that node. The whiteboard information indicates the last port visited by a_2, which is indeed the black hole position[1].

Scenario-4: Both a_1 and a_2 fail to report while they are performing *pendulum* walk.

Here, the $Leader$ moves towards the agent which has the value: $\min_{i\in\{1,2\}}(Alen_{a_i} + Llen_{a_i})$. Let that agent be a_1. So, the $Leader$ moves towards a_1 with the help of $Apath_{a_1}$ and $Lpath_{a_1}$, while updating `Last.Leader`, $Llen_{a_2}$ and $Lpath_{a_2}$, respectively. If, while traversing, the $Leader$ encounters a missing edge along the direction of its movement, it stops and waits for the missing edge to reappear. While waiting, if the $Leader$ finds that again a_2 fails to report within $\max_{i\in\{1,2\}}(2\cdot(Alen_{a_i} + Llen_{a_i} + 1))$ rounds, then it concludes that a_2 is destroyed by the black hole. This node can be located by following the instructions mentioned in the earlier case. On the contrary, if while traversing towards a_1, no missing edge is encountered, then the $Leader$ reaches to the last reported node of a_1 with the help of $Apath_{a_1}$ and $Lpath_{a_1}$, respectively. If a_1 is not found at this node, the $Leader$ concludes that a_1 is destroyed by the black hole. In addition to that, the node at which a_1 got destroyed is also detected by the $Leader$. The black hole node is the neighbouring node of the current position of $Leader$ with respect to the port j if a_1 visited the j-th port for the last time from the last reported node.

Next, we first define the purpose of the states that a_1 or a_2 or $Leader$ can attain while executing their respective algorithm.

An agent, i.e., a_1 or a_2 while executing SINGLEEDGEBHSAGENT can change to any of these following states:

Cautious: This state resembles the movements performed by an agent while executing *cautious* walk. An agent, say a_1, is in the state **Cautious**, first sets the variable $Move1 = 0$ and checks if a port exists whose f value is $\perp$. If such a port exists, it chooses the lowest

[1]The exact port should be the highest port along which $0 \circ A$ is written and $a_2 \in A$, this port is chosen by a_2 based on the protocol $t-$INCREASING-DFS.

port among them and traverses along it to an adjacent node. While traversing, it updates $Move1 = 1$ and writes $0 \circ \{a_1\}$ at the whiteboard of this edge's incoming and outgoing ports. Next, if the adjacent port is safe and the edge between the $Leader$ and a_1 exists, then it returns to the $Leader$. While backtracking, it sets $Move1 = 2$. In the next round, if the edge still exists, then it changes $Move1 = 0$ again and, together with the $Leader$, traverses to the adjacent node. Note that after backtracking to the previous node, if the $Leader$ cannot be found (i.e., the $Leader$ has already moved from its earlier position), then the agent follows `Last.Leader` at each node until it meets the $Leader$, and all this while, the $Move1$ value remains the same (i.e., 2). After it finds the $Leader$, it changes its $Move1$ to 0.

On the other hand, if no such port exists at the current node with f value $\perp$, then it traverses back to an already visited node along with the $Leader$ and with the same $Move1$ value (i.e., 0). Further, irrespective of any $Move1$ value, whenever the agent encounters a missing edge along its path, it waits at that adjacent node until the edge reappears or the $Leader$ instructs a specific movement.

Pendulum: This state resembles the movement of an agent performing *pendulum* walk. An agent, say a_1, is in the state **Pendulum**, first sets the variable $Move2 = 0$ and checks whether a port exists at the current node with f value $\perp$. If so, it chooses the minimum port among them. If not, then it checks if a port exists at the current node with f value $0 \circ \{a_2\}$. If such a port exists, it chooses the minimum port among them. Now, irrespective of which port is chosen, i.e., either marked as $\perp$ or $0 \circ \{a_2\}$, the agent, while traversing along this port, updates $Move2 = 1$ and updates $Apath_{a_1} = Apath_{a_1} \cup (j, m)$ (where j and m are the ports of the edge along which a_1 traverses). Thereafter, it moves towards the adjacent node while writing $0 \circ \{A \cup a_1\}$ on the whiteboard of both the ports j and m, respectively. Next, after exploring the new node, it backtracks the next round onwards until it meets the $Leader$. While backtracking, the agent follows the stored sequence $Apath_{a_1}$ to navigate its path towards $Leader$, and whenever that sequence exhausts and yet the $Leader$ is not found, then the agent uses `Last.Leader` at each node to find the $Leader$. At the same time, following the edges with the help of `Last.Leader`, a_1 also updates $Apath_{a_1}$ for each new port that it takes. Whenever it finds the $Leader$, it changes $Move2$ from 2 to 1. Next, a_1 again follows $Apath_{a_1}$ to reach its last reported node. If, along this movement, the

sequence $Apath_{a_1}$ is exhausted, that means a_1 has reached the last reported node. So, after reaching this node, it again sets $Move2 = 0$ and iterates this process.

On the other hand, if there does not exist any port at the current node, which is either marked as $\perp$ or $0 \circ \{a_2\}$, then in that case, a_1 sets $Move2 = 3$, and communicates this information with the $Leader$. After which, it backtracks to an already traversed node following the whiteboard and at each node that the $Leader$ visits, it checks whether there exists a port that is marked as $\perp$ or $0 \circ \{a_2\}$. If it finds such a port, it sets $Move2 = 4$ and updates $Apath_{a_i}$ to store the sequence of ports from $Leader$ to the current node, also updates $Alen_{a_i} = |Apath_{a_i}|$, where $|Apath_{a_i}|$ is the cardinality of $Apath_{a_i}$. Thereafter it moves towards the $Leader$, and after the $Leader$ is found, it communicates these $Move2$, $Alen_{a_i}$ and $Apath_{a_i}$ values to the $Leader$ and returns to the last node, following the sequence $Apath_{a_i}$ and sets $Move2 = 0$. Consecutively, the process mentioned for $Move2 = 0$ earlier, is iterated.

Currently, we discuss the states the $Leader$ can attain while executing the algorithm SINGLEEDGEBHSLEADER.

Initial: This state symbolizes the start of the algorithm SINGLEEDGEBHSLEADER. Initially the $Leader$ is accompanied by the agents a_1 and a_2 at $home$, where the $Leader$ initializes the variables: $Apath_{a_i} = Lpath_{a_i} = NULL$ and $Alen_{a_i} = Llen_{a_i} = 0$, for all $i \in \{1,2\}$. Next, it instructs the lowest ID agent, i.e., a_1, to change to state **Cautious** and a_2 to state **Pendulum**, and itself changes its state to **Assign**.

Assign: This state aims to initialize certain variables based on the fact that which agent (i.e., a_1 or a_2) is instructed to perform which walk. After all this initialization, the $Leader$ changes to state **Movement**.

Movement: This state discusses the updation of variables and direction of $Leader$'s movement (if at all it moves). Let a_1 be the agent, which returns to the $Leader$ along the port j while performing *cautious* walk. Now, if the $Leader$ is present at the same node at which it last met a_1, it decides to move with a_1 to the adjacent node from where a_1 has returned. While the $Leader$ moves, it updates $Lpath_{a_2} = Lpath_{a_2} \cup (j,m)$ (where (j,m) are the incoming and outgoing port of the edge chosen by the $Leader$ for its movement) and $Llen_{a_2} = Llen_{a_2} + 1$. It also updates `Last.Leader` to 1 for the port j and the remaining

ports to 0.

On the other hand, if an agent a_1 (say) returns along port j with $Move2 = 1$ while it performs *pendulum* walk, then the $Leader$ understands that a_1 has returned after exploring a new node. Accordingly, the $Leader$ gathers this information and updates $Apath_{a_1}$ and increments $Alen_{a_1}$ by 1. If a_1 returns with $Move2 = 3$, the $Leader$ understands that there is no new information to gather from a_1 and it does not update $Apath_{a_1}$ and $Alen_{a_1}$. Lastly, if a_1 returns with $Move2 = 4$, the $Leader$ understands that while backtracking, about the fact that the agent a_1 has found a port marked as $\perp$ or $0 \circ a_2$. So, at this point, the $Leader$ updates the variables $Apath_{a_1}$ and $Lpath_{a_1}$ from a_1, and instructs a_1 to reach the last reported node and then continues to perform *pendulum* walk along this available port, after changing to state **Pendulum**.

Missing-Edge-Leader: The $Leader$ changes to this state when it finds a missing edge. There are two possibilities: the current node contains only the $Leader$, or another agent is present.

If the $Leader$ is alone, it waits at the current node until the edge reappears. While waiting, if an agent fails to report, then the $Leader$ changes its state to **Fail-to-Report**. On the other hand, if another agent is present, then the $Leader$ changes to state **Assign** by assigning the agent to either change to state **Cautious** or **Pendulum**. It may be noted that the $Leader$ can only instruct an agent to change its state to **Cautious** if it finds the missing edge reappear, and in addition to that, the other agent is performing *pendulum* walk.

On the other hand, if the edge reappears and an agent reports, then that agent is instructed to continue its earlier walk or change its walk, whereas the $Leader$ changes its state to **Assign**. Otherwise, if no agent reports even after the edge has reappeared, then the $Leader$ continues to perform **Fail-to-Report** (if, before changing to state **Missing-Edge-Leader**, it has been in state **Fail-to-Report**). Otherwise, if it has been in a different state before changing to **Missing-Edge-Leader**, it changes to **Fail-to-Report**.

Fail-to-Report: This state symbolises the decision that the $Leader$ takes after it finds that an agent has failed to report. There are two possibilities: (1) an agent a_1 (say) fails to report while performing *cautious* walk, or (2) a_1 fails to report while performing *pendulum* walk. If a_1 fails to report while performing *cautious* walk and the edge between the $Leader$ and

that of a_1 is missing, then $Leader$ changes its state to **Missing-Edge-Leader**. Otherwise, if there is no missing edge between the $Leader$ and a_1, then $Leader$ concludes a_1 to be destroyed by the black hole, which terminates the algorithm. The second scenario occurs when a_1 fails to report while performing *pendulum* walk. In this scenario, after a_1 fails to report, if the other agent is currently performing *cautious* walk, then it is instructed to perform *pendulum* walk, and the $Leader$ moves towards a_1. If a_1 is found and a missing edge is encountered or not, the $Leader$ instructs a_1 to either perform *cautious* or *pendulum* walk, and it changes to either **Assign** or **Fail-to-Report**. If a_1 as well as a missing edge is not found, then the black hole location is concluded, which terminates the algorithm. Otherwise, if a missing edge is encountered while moving towards a_1, the $Leader$ changes to state **Fail-to-Report**.

The third scenario arises when the $Leader$ finds both the agents are not reporting, where we suppose a_1 is performing *pendulum* walk, and a_2 is performing *cautious* walk. In this scenario, if the edge between the $Leader$ and a_2 doesn't exist, then the $Leader$ concludes that the last visited node of a_1 is the black hole. Otherwise, the $Leader$ concludes a_2 is destroyed by the black hole.

The last scenario occurs when both agents a_1 and a_2 are performing *pendulum* walk, and they fail to report. This is understood by the $Leader$ after it waits for $\sum_{i=1}^{2}(2\cdot(Alen_{a_i}+Llen_{a_i}+1))$ rounds. So, the $Leader$ simply changes its state to **Fail-to-Report-Movement**.
Fail-to-Report-Movement: This state explains the phenomenon when both agents, i.e., a_1 and a_2, fail to report while performing *pendulum* walk. The $Leader$ moves towards the agent with $\min_{i\in\{1,2\}}(2\cdot(Alen_{a_i}+Llen_{a_i}+1))$ value. While moving, if it encounters a missing edge, it waits. While waiting, if the other agent (say, a_2) again fails to report (i.e. wait time is greater than $\max_{i\in\{1,2\}}(2\cdot(Alen_{a_i}+Llen_{a_i}+1))$), then the $Leader$ concludes that the last node visited by a_2 is the black hole. On the other hand, if within the waiting period a_2 reports, then the $Leader$ changes its state to **Fail-to-Report**.

If while the $Leader$ moves towards the agent with $\min_{i\in\{1,2\}}(2\cdot(Alen_{a_i}+Llen_{a_i}+1))$, and encounters no missing edge but finds the agent (say, that agent be a_1), in that case it instructs a_1 to change its state to **Pendulum** and initializes all the parameters associated with a_1 to 0 and $NULL$. The $Leader$, changes to state **Fail-to-Report**. But if no agent is

found, the *Leader* concludes that the last node visited by a_1 is the black hole position.

The pseudo codes of SINGLEEDGEBHSAGENT and SINGLEEDGEBHSLEADER are, thus, explained in Algorithm 9 and Algorithm 10 referred in Appendix 4.6.

Correctness and Complexity

In this section, we have proved the correctness of our algorithm, as well as shown the upper bound results in terms of move and round complexity.

Lemma 4.4.1. *Algorithms* SINGLEEDGEBHSAGENT *and* SINGLEEDGEBHSLEADER *on a dynamic cactus graph $\mathcal{G}$ with at most one edge being dynamic at any round, ensure that between a_1, a_2 and Leader, at most 2 among them can be stuck or waiting for a missing edge.*

Proof. To prove this claim, we have discussed the decisions that the *Leader* takes whenever it encounters a missing edge along its path. The *Leader* can either encounter a missing edge while it is moving with an agent a_1 (say), which is performing a *cautious* walk, or when it is moving towards an agent a_1 (say), which fails to report, while performing a *pendulum* walk. We have discussed each case that the *Leader* may encounter:

- If the *Leader* encounters a missing edge while it is moving with a_1, which is performing *cautious* walk, then either a_1 is with the *Leader*, or they are separated by this missing edge. In the first scenario, the *Leader* instructs a_1 to change its state from **Cautious** to **Pendulum**. This implies that a_1 starts performing *pendulum* walk around the remaining paths (those ports can be identified with the help of a whiteboard at each visiting node). In the meantime, the *Leader* waits until the edge reappears (refer to state **Missing-Edge-Leader** in Algorithm 10). On the other hand, if the *Leader* and a_1 are separated due to the missing edge, then *Leader* remains stationary until the missing edge reappears, or the other agent, i.e., a_2, also fails to report.

- If the *Leader* is moving towards a_1, which has been performing *pendulum* walk, then any of the following two situations can occur: either the *Leader* can find an agent stuck due to a missing edge, or it may encounter a missing edge not occupied by any agent. For the first situation, the agent can be found stuck or waiting, which

can be either a_1 or a_2. In that case, the $Leader$ instructs that agent to perform *pendulum* walk along any alternate available path, and the $Leader$ itself remains stationary. The $Leader$ remains stationary until the missing edge reappears or it finds some other agent has failed to report. In the second situation, whenever it encounters a node incident to a missing edge not occupied by any agent, it waits until the missing edge reappears or any other agent fails to report.

So, in either case, the $Leader$ does not allow more than one agent to occupy one end of the missing edge.

Now, we discuss the decisions an agent takes while encountering a missing edge.

- If an agent a_1 (say) encounters a missing edge along its movement and finds that another agent a_2 is already present for that missing edge, then a_1 chooses an alternate port and continues executing its algorithm, or if there does not exist any available port, it backtracks from the current node.
- If an agent a_1 (say) encounters a missing edge and finds the node adjacent to the missing edge to be vacant, then a_1 waits until the missing edge reappears or the $Leader$ instructs a_1 not to wait by interchanging the position with a_1.

Similarly, in this scenario as well, an agent also does not allow more than one agent to remain stuck or waiting for a missing edge. This ensures our claim that, between a_1, a_2, and $Leader$, at most two among them can be stuck or waiting due to a missing edge. □

Lemma 4.4.2. *The $Leader$ executing* SINGLEEDGEBHSLEADER *ensures that if both a_1 and a_2 are either stuck or waiting due to a missing edge, then the $Leader$ eventually instructs one among these agents to move, whereas it itself waits for the missing edge.*

Proof. Let r be the first round when both a_1 and a_2 are stuck or waiting at the nodes u and v, respectively, due to a missing edge (u, v). The movements of these agents before they get stuck (or waiting) lead us to the following cases.

- Both a_1 and a_2 have performed *pendulum* walk before each of them gets stuck.

This scenario of both agents performing *pendulum* walk can only arise when at a round $r' < r$, the $Leader$ gets stuck or is waiting at one end of the missing edge. The $Leader$, while it waits, must have instructed both agents to perform *pendulum* walk if not already performing. Now, at round r, while a_1 and a_2 are performing *pendulum* walk, they encounter a missing edge and again get stuck (or wait) due to the missing edge. This means that from round r onwards, the missing edge for which the $Leader$ has been waiting has reappeared (since there is at most one dynamic edge at any round). Let r'' (where $r < r''$) be the current round, at which $Leader$ finds that both agents fail to report. It can happen only when the $Leader$ finds no agent reporting even after waiting for at most $\sum_{i=1}^{2}(2 \cdot (Alen_{a_i} + Llen_{a_i} + 1))$ rounds since any of these agents last reported. This triggers the $Leader$ to move towards the agent with $\min_{i \in \{1,2\}}(2 \cdot (Alen_{a_i} + Llen_{a_i} + 1))$. Let that agent be a_1, so within $r'' + (2 \cdot (Alen_{a_1} + Llen_{a_1} + 1))$ rounds, the $Leader$ reaches the node occupied by a_1 and finds a_1. Next, it instructs a_1 at u to start *pendulum* walk.

- One agent has performed *cautious* walk, whereas the other agent has performed *pendulum* walk before they get stuck (or wait) for the missing edge at round r.

  Let a_1 be the agent who has performed *cautious* walk. So, at round r, whenever it has chosen to travel through the edge (u, v), it has found that the edge (u, v) is missing and hence gets stuck or waits for the missing edge to reappear. The $Leader$, being present at the same node also, must have found a_1 stuck (or waiting) at round r. Hence, in this situation, it must have instructed a_1 to leave u by changing to *pendulum* walk at round r itself.

Thus, in both these scenarios, it is proved that if both a_1 and a_2 occupy both sides of a missing edge, then the $Leader$ eventually instructs one among them to continue performing its movement, i.e., it makes the stuck or waiting agent free to move, whereas the $Leader$ itself remains stationary until either the missing edge reappears or it moves towards certain agent. Hence, this proves our claim. □

Lemma 4.4.3. *Neither the agents executing* SINGLEEDGEBHSAGENT *nor the Leader executing* SINGLEEDGEBHSLEADER *explore any cycle infinitely.*

Proof. We, first, have shown that an agent a_1 (without loss of generality) while executing Algorithm 9 can never explore any cycle infinitely. Let C_1 be a cycle in $\mathcal{G}$ containing a node u along which the agent a_1 starts exploring this cycle. Let us suppose that from u, a_1 moves to an adjacent node v along the port j. While moving, it updates $f(j) = f(j) \circ \{a_1\}$ (here, earlier $A = \Phi$ or $A = \{a_2\}$) at the whiteboard of u. Next, after exploring all the nodes of C_1, whenever a_1 returns to u and it attempts to choose the port j again, it finds that it has already visited the port j (based on the $f(j)$ value written on the whiteboard). Hence, it does not choose this port again, irrespective of whether a_1 is performing *cautious* walk or *pendulum* walk. This proves that an agent executing SINGLEEDGEBHSAGENT never explores any cycle infinitely.

Further, the $Leader$ can move with an agent while the accompanying agent is performing *cautious* walk, or it can move towards an agent who has failed to report while executing *pendulum* walk. In the first case, as the agent itself never explores any cycle infinitely, the $Leader$, accompanying it, can also never explore any cycle infinitely. In the second case, the $Leader$ moves towards the agent by following the same path that the agent a_1 (say) has already traversed, which is the length at most $\max_{i\in\{1,2\}}(2\cdot(Alen_{a_i} + Llen_{a_i} + 1))$. Hence, in this case, it is impossible for the $Leader$ to explore any cycle infinitely since the agent itself, executing Algorithm 9, cannot explore any cycle infinitely according to the earlier argument. This shows that the $Leader$ can also never explore any cycle infinitely. □

Lemma 4.4.4. *The algorithm* SINGLEEDGBHSAGENT *ensures that in the worst case, every node in $\mathcal{G}$ is explored by either a_1 or a_2 until any one among them gets destroyed by the black hole or the $Leader$ terminates the algorithm.*

Proof. As stated earlier, a_1, a_2 and $Leader$ are all $t-$state finite automata's, where $t \geq \alpha n\Delta\log\Delta$. An agent (either a_1 or a_2) finds eligible ports [2] at the current node, irrespective of whether they are executing *cautious* or *pendulum* walk. Among these eligible ports, the one the agent chooses is based on the protocol of $t-$INCREASING-DFS. It may be noted that this $t-$INCREASING-DFS protocol requires $O(n\Delta\log\Delta)$ bits of memory for an agent. It is because the agents, while executing Algorithm 9, not only explore each port in $\mathcal{G}$ but

[2] A port j is eligible for an agent a_1 (say) executing *cautious* walk, if $f(j) = \perp$, whereas if a_1 is executing *pendulum* walk, then an eligible port is one, which is either $f(j) = \perp$ or $f(j) = 0 \circ A \setminus \{a_1\}$.

also store the ports that they visit. The ports are stored via the sequence $Apath_{a_i}$ (where $i \in \{1,2\}$), where this sequence only gets incremented when an agent is performing *pendulum* walk. In addition to this, while only exploring a new port (not visited yet by a_i), the sequence $Apath_{a_i}$ is updated. As per the eligibility of a port, a port already visited by a_i is never explored again. Only during backtracking may it be visited, but during this time, the sequence $Apath_{a_i}$ is not updated. Each node in $\mathcal{G}$ can have at most Δ such ports, and in the worst case, an agent needs to store each such port. So this shows that $O(n\Delta \log \Delta)$ bits are sufficient for an agent to store each port in $\mathcal{G}$. This further implies that, the cardinality of $Apath_{a_i}$ is at most $O(n\Delta \log \Delta)$. Further, as stated in Theorem 6 and Corollary 7 of [70], an agent with $O(n \log \Delta)$ bits of memory is sufficient to explore any static graph with diameter n and maximum degree Δ. So, $O(n\Delta \log \Delta)$ bits are more than sufficient for an agent to explore $\mathcal{G}$. Additionally, Lemma 4.4.2 states that eventually, either $Leader$ or one of a_1 or a_2 remains stuck (or waits) due to a missing edge. Now, as $\mathcal{G}$ can have at most one dynamic edge at any round, the other agent among a_1 and a_2 can unobstructively explore the remaining graph. It can explore until it is destroyed by the black hole or the $Leader$ terminates the algorithm by either detecting the position of the black hole or the path that indicates the black hole location by visiting the last node. □

Lemma 4.4.5. *Algorithm* SINGLEEDGEBHSLEADER *ensures that the* $Leader$ *never gets destroyed by the black hole.*

Proof. The $Leader$ can move from its current node for either of these two cases: (1) it is accompanied by an agent that is performing *cautious* walk, (2) when it finds that a certain agent has failed to report. In the first case, it is observed that the $Leader$ only visits a node after it is ensured safe by the agent performing *cautious* walk. This means that it is only possible for the agent to get destroyed by the black hole, as it never explores an unexplored node while accompanying the agent, performing *cautious* walk.

In the second case, the $Leader$ moves from its current node only after it finds that an agent a_1 (say) has failed to report while performing *pendulum* walk. In this case, the $Leader$ moves towards a_1. The path taken by the $Leader$ to reach a_1 is already traversed by either a_1 or a_2 or both. It is because, the $Leader$ only follows the sequence $Apath_{a_1} \cup$

$Lpath_{a_1}$. Hence, the $Leader$ cannot be destroyed by the black hole. □

Lemma 4.4.6. *At least one among a_1 and a_2 executing* SINGLEEDGEBHSAGENT *gets destroyed by the black hole within $O(n^2)$ rounds.*

Proof. It may be recalled that the $Leader$ instructs one among a_1 or a_2 to perform *cautious* walk, or it instructs both a_1 and a_2 to perform *pendulum* walk. We have the following cases based on these movements, and we have proved our claim for each case.

- The case where one among a_1 and a_2 is instructed to perform *cautious* walk and the other to perform *pendulum* walk: let us suppose a_1 be the agent which is instructed to execute *cautious* walk, whereas a_2 is instructed to execute *pendulum* walk. In this situation, a_1 explores and moves to a new node in every 3 rounds (if not stuck or waiting due to a missing edge), whereas a_2 requires at most $2n$ rounds to explore a new node. Hence, in the worst case, if a_2 is not blocked at all, then it takes $O(n^2)$ rounds to get destroyed by the black hole.
- The case when both a_1 and a_2 are instructed to perform *pendulum* walk: as per our algorithm, this scenario can arise when the $Leader$ is stationary due to an adjacent missing edge. So, while the $Leader$ is still waiting for the missing edge to reappear, at least one among these two agents can unobstructively perform *pendulum* walk. In the worst case, the other agent may get stuck (or wait) on the other node of the same missing edge. Now, an agent performing *pendulum* walk requires at most $2n$ rounds to explore a new node, which shows that within $O(n^2)$ rounds, at least one agent gets destroyed by the black hole.

□

Lemma 4.4.7. *Let r be the round at which one among a_1 and a_2 while executing* SINGLEEDGEBHSAGENT*, is the first to get destroyed by the black hole, then the $Leader$ while executing* SINGLEEDGEBHSLEADER *terminates the algorithm within $r + O(n^2)$ rounds.*

Proof. Let us assume a_1 is the first agent to get destroyed by the black hole at round r. First, it is needed to be observed that $\max_{i\in\{1,2\}}(Alen_{a_i} + Llen_{a_i}) \leq n\Delta \approx n^2$, since $\Delta \leq n-1$. This

inequality holds since an agent never explores a port more than once. It can visit a port more than once while backtracking, but during backtracking, it never stores the port along which it backtracks. Hence, we have the following cases based on the movement strategies followed by a_1 and a_2 while executing Algorithm 9.

- Let a_1 is destroyed at round r while it is performing *cautious* walk.

  This means at round r, a_1 has visited an unexplored node, while the $Leader$ waits at the adjacent node for a_1 to report. As a_1 is performing *cautious* walk, a_2 must be performing *pendulum* walk. So, now we have two situations. Either the edge e between $Leader$ and a_1 exists, or it has gone missing. If it exists, then the $Leader$ at round $r+1$ finds that a_1 has failed to report. Hence, it identifies the black hole and terminates the algorithm. On the contrary, suppose the edge e is missing from round r onwards, as the underlying graph can have at most one such dynamic edge at any round; so while the $Leader$ waits for the missing edge to reappear, the agent a_2 can unobstructively continue *pendulum* walk until it gets destroyed by the black hole. It may be noted that *pendulum* walk requires at most $2n$ rounds to explore a new node. This implies that within $r+O(n^2)$ rounds, a_2 also gets destroyed by the black hole. If e is still missing, then the $Leader$ finds that a_2 fails to report, after waiting for $2\cdot(Alen_{a_2}+Llen_{a_2}+1)$ rounds since a_2 last met with $Leader$. This helps the $Leader$ to conclude that a_2 is destroyed by the black hole and accordingly terminates the algorithm. The $Leader$ terminates the algorithm because it knows the exact path to visit in order to locate the black hole. Otherwise, if e reappears (within $2\cdot(Alen_{a_2}+Llen_{a_2}+1)$ rounds since a_2 last met with $Leader$), then the $Leader$ finds that a_1 is not reporting. So, in the next round itself (as a_1 fails to report), the $Leader$ concludes that a_1 is destroyed by the black hole, which terminates the algorithm. This implies that within at most $r+O(n^2)+n^2 = r+O(n^2)$ rounds, the $Leader$ terminates the algorithm.

- Let a_1 be destroyed at round r by the black hole while performing *pendulum* walk.

  There are two possibilities for this case: (1) a_2 is performing *cautious* walk at round r, or (2) a_2 is performing *pendulum* walk at round r. The case (2), where a_2 is per-

forming *pendulum* walk, is discussed in the next case. Let us consider (for case (1)) that a_2 is performing a *cautious* walk. So anyhow the $Leader$ at round r' (where $r' \leq r + 2 \cdot (Alen_{a_1} + Llen_{a_1} + 1) \leq r + n^2$) finds that a_1 fails to report. This implies that at round r', if a_2 is with the $Leader$, then it instructs a_2 to change its movement from *cautious* to *pendulum*. Otherwise, if a_2 is not with the $Leader$, then the $Leader$ waits for 2 rounds (assuming there is no missing edge between the $Leader$ and a_2) for a_2 to report. Then, it instructs a_2 to change its movement from *cautious* to *pendulum*. Next, in that round, the $Leader$ moves towards a_1 following the sequence $Apath_{a_1} \cup Lpath_{a_1}$. While moving, if it does not encounter any missing edge, then at round r'' (where $r'' \leq r' + Alen_{a_1} + Llen_{a_1} \leq r + O(n^2)$), the $Leader$ identifies that a_1 is destroyed by the black hole and terminates the algorithm. On the contrary, while moving towards a_1, if the $Leader$ encounters a missing edge, it waits again for the missing edge to reappear. In the meantime, a_2 has already begun performing *pendulum* walk, from round r' onwards (or from $r' + 2$ onwards). So, it explores a new node in at most $2n$ rounds and then reports to the $Leader$. If a_2 also gets stuck (or waits) due to a missing edge, and fails to report at some round $r_1 \leq r + O(n^2)$, then the $Leader$ again moves towards a_2. Either the $Leader$ finds a_2, or a_1 again fails to report (because a_1 is already destroyed by the black hole at round r) and eventually, the $Leader$ terminates the algorithm. So, in worst case, the $Leader$ takes $r_1 + 2n^2 + O(n^2) = r + O(n^2)$ rounds to terminate the algorithm.

- Let a_1 be destroyed by the black hole at round r, while a_1 and a_2 are performing *pendulum* walk.

  Earlier, this case arose because the $Leader$ has been waiting for a missing edge to reappear. Otherwise, both agents are never instructed to perform *pendulum* walk while they are still reporting back to $Leader$. Now, since a_1 fails to report, and if a_2 is not obstructed at all, then a_2 can explore a new node in every at most $2n$ rounds and eventually fails to report, as it gets destroyed by the black hole. So, this concludes that within $r + O(n^2)$ rounds, a_2 also gets destroyed. Next, whenever the $Leader$ finds that a_2 has failed to report, then it moves towards $\min_{i \in \{1,2\}}(Alen_{a_i} +$

$Llen_{a_i}$) and either detects the black hole which terminates the algorithm or encounters a missing edge. If it again encounters a missing edge, then it further waits for $\max_{i\in\{1,2\}}(Alen_{a_i} + Llen_{a_i})$ rounds. Again, the $Leader$ finds that no agent is reporting, as both are destroyed. In this case, the $Leader$ detects the agent (based on the entity $\max_{i\in\{1,2\}}(Alen_{a_i} + Llen_{a_i})$) destroyed by the black hole and terminates the algorithm, as it knows the exact agent which has been destroyed and also it knows the path to follow in order to locate the black hole. In the worst case, this process requires $r + 2n^2 + O(n^2)$ rounds. This shows that the $Leader$ terminates the algorithm in at most $r + O(n^2)$ rounds.

The above cases cover all possibilities, and in each case, we have proved our claim. □

Lemma 4.4.8. *The $Leader$ executing* SINGLEEDGEBHSLEADER *correctly terminates the algorithm.*

Proof. It may be noted that, according to our Definition 4.2.1, the surviving agent can terminate the algorithm when it either knows the exact black hole node or knows the path it needs to visit to determine the black hole node. In our BHS algorithm, the $Leader$ is the one who can terminate the algorithm. The $Leader$ while executing SINGLEEDGEBHSLEADER encounters many scenarios. In this proof, we discuss all such scenarios and show that the $Leader$ correctly terminates the algorithm in each of them.

Case-1: [An agent (say, a_1) has been consumed by the black hole while executing *cautious* walk.] Let the black hole node be u. Since a_1 has been executing *cautious* walk, the $Leader$ is located at one of the neighbours of u, say at v. Now, if the edge (u, v) exists, then the $Leader$ finds that a_1 fails to report and concludes that a_1 is destroyed by the black hole. Since the $Leader$ knows the port by which a_1 has travelled to reach u, accordingly, the $Leader$ locates the black hole and terminates the algorithm. Otherwise, if the edge (u, v) is missing, at the moment when a_1 gets destroyed by the black hole. Then the $Leader$ waits for that edge to reappear, whereas the other agent, i.e., a_2, continues to perform *pendulum* walk and eventually gets destroyed. In that case, the $Leader$ understands a_2's failure to report, after waiting $2 \cdot (Alen_{a_2} + Llen_{a_2} + 1)$ rounds. After which, the $Leader$ concludes that a_2 is destroyed by the black hole and terminates the algorithm. The reason for termination

is that the $Leader$ currently knows the location of the black hole, as it can be determined by traversing the sequence of ports in $Apath_{a_2} \cup Lpath_{a_2}$ and then accessing the whiteboard. It is noted that this conclusion by the $Leader$ is indeed correct. It is because the $Leader$ at node v is occupying one end of the missing edge, whereas the other node is the black hole (which has earlier destroyed a_1). Now, as the underlying graph has at most one missing edge, so a_2, other than being destroyed by the black hole, must not have faced any obstruction to report back within $2 \cdot (Alen_{a_2} + Llen_{a_2} + 1)$ rounds.

Case-2: [An agent (say, a_1) has been destroyed by the black hole while executing *pendulum* walk.] It may be noted that a_1 has been performing *pendulum* walk before it gets destroyed by the black hole, and this can only happen if either a_2 has failed to report or the $Leader$ has encountered a missing edge. We have explained each of these possibilities in detail.

- a_2 *fails to report:* In this case, the $Leader$ instructs a_1 to change its movement from *cautious* to *pendulum*. Next, while the $Leader$ starts moving towards a_2 following $Apath_{a_2} \cup Lpath_{a_2}$, either it finds a_2, or it is not found, or the $Leader$ encounters a missing edge.

  Case-A: If the $Leader$ encounters a missing edge, it waits. In the meantime, a_1, while performing *pendulum* walk, gets destroyed by the black hole and fails to report. As a_1 has also failed to report, the $Leader$ moves towards a_1, leaving the node adjacent to the missing edge. If it faces no obstruction, it correctly locates the black hole and terminates the algorithm. But if it encounters a missing edge while moving towards a_1, it waits for a_2 and eventually a_2 reports to the $Leader$. It is because the earlier missing edge for which a_2 has failed to report has reappeared (since there can be at most missing edge at any round). In this case, a_2 continues to perform *pendulum* walk. While performing, a_2 eventually gets destroyed by the black hole and also fails to report. In this situation, $Leader$ again moves towards a_2 as described earlier, leaving the node adjacent to the missing edge and correctly locates the black hole. It is because while moving towards a_2, either a_2 is not found or the $Leader$ encounters a new missing edge. If the missing edge is encountered, then failure of a_1's return concludes that a_1 has been destroyed. Moreover, the $Leader$, having knowledge of the

path until the last reported node of a_1, knows the exact path to visit in order to locate the black hole and terminates the algorithm. Otherwise, not finding a_2 means that a_2 is destroyed by the black hole, which the $Leader$ understands, and terminates the algorithm.

Case-B: If a_2 is found, and whether there is a missing edge along the next direction of a_2 or not, the $Leader$ instructs a_2 either to perform *cautious* or *pendulum* walk. Now a_1 fails to report since it has been destroyed by the black hole. If a_2 has been performing *cautious* walk before a_1 fails to report, and when the $Leader$ understands that a_1 has failed to report, then it instructs a_2 to change to *pendulum* walk. Otherwise, the $Leader$ instructs a_2 to continue performing *pendulum* walk. Irrespective of which instruction the $Leader$ provides, it invariably moves towards a_1. If the $Leader$ does not find any missing edge corresponding to the last traversed port of a_1 at the last reported node, it correctly locates the black hole and terminates the algorithm. Otherwise, if the $Leader$ encounters a missing edge, it waits for the missing edge to reappear. When $Leader$ is waiting, a_2 is performing *pendulum* walk. After a few rounds, a_2 either reaches the other end of the missing edge or gets destroyed by the black hole. This means a_2 also eventually fails to report. So, the $Leader$ finds that both a_1 and a_2 are not reporting. Hence, it moves towards a_2 as the path towards a_1 is already blocked by a missing edge. If a_2 is found, then the $Leader$ correctly concludes that a_1 has been destroyed by the black hole (location of which can be found after visiting the sequence of ports $Apath_{a_1} \cup Lpath_{a_1}$ and then finding the last visited port at the whiteboard of last reported node), which terminates the algorithm. On the other hand, if the $Leader$ again encounters a missing edge while moving towards a_2, it waits for another $\max_{i\in\{1,2\}}(2\cdot(Alen_{a_i}+Llen_{a_i}+1))$ rounds. Thereafter, the $Leader$ concludes that a_1 has been destroyed by the black hole and terminates the algorithm, as now the $Leader$ knows the sequence of ports, i.e., $Apath_{a_1} \cup Lpath_{a_1}$ to reach the last reported node of a_1, and eventually visiting this node will determine the location of the black hole node.

Case-C: If a_2 is not found and there is no missing edge, the $Leader$ correctly con-

cludes that the black hole has destroyed a_2 and locates the position of the black hole and terminates the algorithm, as otherwise a_2 must have been found.

- *$Leader$ encounters a missing edge:* In this case, both a_1 and a_2 must execute *pendulum* walk as per SINGLEEDGEBHSLEADER. While performing their respective movement, if both a_1 and a_2 fail to report, then the $Leader$ moves towards the agent with $\min_{i\in\{1,2\}}(2\cdot(Alen_{a_i}+Llen_{a_i}+1))$ (let that agent be a_1) and eventually locates the black hole (refer to the earlier case) and terminates the algorithm. On the contrary, while moving towards a_1, if the $Leader$ encounters a missing edge, then this case is similar to the earlier Case-B, where the $Leader$ again terminates the algorithm.

□

Theorem 4.4.9. *The agents a_1 and a_2 executing* SINGLEEDGEBHSAGENT *and the $Leader$ executing* SINGLEEDGEBHSLEADER *correctly terminate the algorithm, in a dynamic cactus graph $\mathcal{G}$ with at most one dynamic edge at any round, within $O(n^2)$ moves and $O(n^2)$ rounds.*

Proof. Lemmas 4.4.6 and 4.4.7 state that our Algorithms 9 and 10 correctly terminate within $O(n^2)$ rounds, and since at each round, the $Leader$ and the agents can move at most once. Hence, there can be at most 3 moves in each round. This implies that the agents following Algorithm 9 and the $Leader$ following Algorithm 10 solve the BHS problem within $O(n^2)$ moves. □

4.4.2 Black Hole Search in Presence of Multiple Dynamic Edges

In this section, we discuss the case where the dynamic cactus graph $\mathcal{G}$ can have at most k ($k>1$ and $k\in\mathbb{Z}$) dynamic edges. In addition, irrespective of the fact that whichever edges are dynamic, the underlying constraint of the 1-interval connectivity property has to be preserved. Accordingly, we present a BHS algorithm, which is termed as MULTIEDGEBHS. This algorithm works with $2k+3$ agents and locates the black hole within $O(kn)$ rounds and $O(k^2n)$ moves.

Initially, a set of $A=\{a_1,a_2,\ldots,a_{2k+3}\}$ agents are co-located at $home$. Our algorithm requires each node to have a whiteboard with $O(deg(u)(\log deg(u)+k\log k))$ bits of stor-

age, where $u \in \mathcal{G}$. For each node $u \in \mathcal{G}$, the whiteboard stores the following information: for each port j of u, where $j \in \{0,1,\ldots,deg(u)-1\}$, an ordered tuple $(g_1(j), g_2(j))$ is maintained. The function g_1 is the same as the function f, defined in Section 4.4.1. The function g_2 is defined as follows, $g_2 : \{0,1,\ldots,deg(u)-1\} \rightarrow \{\perp, 0, 1\}$,

$$g_2(j) = \begin{cases} \perp, & \text{if an agent is yet to visit the port } j \\ 0, & \text{if no agent has returned to the node } u \text{ through the } j\text{-th port} \\ 1, & \text{if the node with respect to } j\text{-th port is safe} \end{cases}$$

Unlike our earlier BHS algorithm discussed for the single edge dynamic case, we don't require any $Leader$, i.e., each of the $2k+3$ agents autonomously executes MULTIEDGEBHS. Similar to Algorithm 9, each agent executes the protocol of $t-$INCREASING-DFS (where $t \geq \alpha n \Delta \log \Delta$, $\alpha \in \mathbb{Z}^+$). In this algorithm, an agent performs two types of action: (1) it explores an unexplored node, or (2) it walks along an already marked safe port. These two types of actions can be explained as follows:

1. *Explore:* In this action, an agent explores an unexplored port. While exploring, it either performs *cautious* walk or *marking* walk.

2. *Trace:* In this action, an agent only moves along safe ports. These ports are marked safe by an agent that has performed the action *explore*.

We now provide a brief overview of our algorithm, MULTIEDGEBHS.

Brief Idea of MULTIEDGEBHS: Each of the $2k+3$ agents start from $home$, where without loss of generality we order, ID of a_i < ID of a_{i+1} ($\forall\ i \in \{1,2,\ldots,2k+2\}$). On a high level, the task of each agent is to explore each port of $\mathcal{G}$. The exploration continues until the agent is destroyed by the black hole, encounters a missing edge along its path of movement, or the black hole is detected. If more than one agent is together, then to explore an unexplored port, the agents together perform a *cautious* walk. If only one agent is present, then to do the very same task, the single agent performs *marking* walk (refer to the definition of *marking* walk in Section 4.2). Next, while traversing $\mathcal{G}$, if an agent discovers a missing edge along its path of movement and no agent is waiting for that edge, this agent waits until the edge reappears. Otherwise, if some agent is already waiting for the same missing

edge to reappear, it simply ignores this port and moves to an alternate port or backtracks. Whenever an agent finds a port whose g_2 value is marked as 0 but the edge corresponding to this port is not missing, it waits for 2 rounds. If no agent arrives and marks g_2 value to 1, then the agent concludes this adjacent node to be the black hole and terminates the algorithm.

Detailed Description of MULTIEDGEBHS: At the beginning, the whiteboard entry corresponding to each port and at each node in $\mathcal{G}$ is marked as $(\perp,\perp)$. So, each agent initially co-located at $home$, sets the following variables: $Move1$, W_{time1}, $Move2$, W_{time2} and W_{time3} to 0. Next, they change their state to **Initial**. In this state, if an agent finds multiple agents are available[3] at the current node, they change their state to **Cautious**. Otherwise, if the agent finds no other agent is available, it changes to state **Marking**.

An agent in state **Cautious** first checks the ports corresponding to the current node. After checking, the following possibilities may arise: (1) a port can be marked as $(\perp,\perp)$, (2) a port can be marked as $(0\circ A,0)$, (3) a port can be marked as $(0\circ A,1)$, (4) a port can be marked as $(1,1)$. Now, based on these ports, we define all possible actions that an agent may execute.

1. If at least one port is found, which is marked as $(\perp,\perp)$ (i.e., all these ports are yet to be visited by any agent), then the following actions are performed. Next, the agent checks whether the $Move1$ value is 0. It may be noted that this $Move1 = 0$ signifies the fact that the agent is available for the next move. Suppose, among all such ports with marking $(\perp,\perp)$, let us assume j to be the lowest port. If the edge with respect to the port j exists, then the lowest ID agent (say, a_i) with $Move1 = 0$ traverses along this port. While traversing, it updates the whiteboard with the value $(0\circ a_i,0)$, at both ports of this edge (say those ports are j and m, respectively). After it reaches the adjacent node, it updates $Move1 = 2$. On the other hand, for all the agents not with the lowest ID, they set $W_{time2} = 0$ and change their state to **Cautious-Wait**. On the contrary, if the edge corresponding to the port j is missing, the lowest ID agent changes its state to **MissingCautious-Wait**, whereas the other agents with $Move1 = 0$, change to state **Initial**.

[3]The available agents are the ones that are neither stuck due to a missing edge nor waiting for other agents to report.

If the agent has a $Move1$ value of 1, then that means the agent has previously performed a *cautious* walk and has moved to a new node. This $Move1$ value also signifies that the agent is not the explorer agent among all the agents performing *cautious* walk. It is because the explorer agent, or the agent leading the cautious walk, never changes its $Move1$ value to 1. Currently, this agent checks if the explorer agent is also present at the current node; if so, it changes the $Move1$ value to 0 and moves into state **Cautious**.

If the agent has $Move1$ value 2, then this signifies that this agent is the explorer agent, performing *cautious* walk, and it has reached a node through a port marked as $(\perp, \perp)$. Currently, it is trying to return to the earlier node through the same edge in order to mark this port (or corresponding edge) safe. If the edge exists (i.e., it has not gone missing), the agent returns by marking the corresponding ports with respect to this edge to 1 (g_2 value) and sets $Move2 = 3$. Other agents agents with $Move1 = 0$ decides to move along the same port, through which the agent with $Move2 = 3$ just returned. Otherwise, if the edge does not exist, i.e., has gone missing, then the agent waits until the edge reappears.

Finally, if the agent has $Move1$ value 3, it implies that the agent has returned to the earlier node (say, u) after reaching a node (say, v) through an unexplored port. Currently, it is trying to reach v again. If it manages to reach v through the same port, then it sets $Move1 = 0$ and changes to state **Initial**. Otherwise, if the edge (u, v) is missing, then it waits with the same $Move1$ value until the edge reappears.

2. If none of the ports has marking $(\perp, \perp)$, then the agent checks for ports with marking $(0 \circ A, 0)$. If such a port exists, but the edge with respect to the lowest port with this marking is missing, then the lowest ID agent with $Move1 = 0$ waits if no agent is already waiting for this edge. All the other agents with the same $Move1$ value change to state **Initial**. On the contrary, if the edge is not missing and no agent is waiting, then all agents with $Move1 = 0$ wait for two rounds. If some agent returns, i.e., g_2 value of this port changes to 1, they all change their state to **Initial**. Otherwise, they conclude that the adjacent node contains the black hole and terminate the algorithm. On the other hand, if $Move1 = 1$, then as discussed in the earlier case, if it finds another agent with $Move1 = 0$, it updates $Move1 = 0$. Otherwise, if $Move1 = 2$ or $Move1 = 3$, then the agent will execute the same instructions as discussed in the earlier case.

3. If none of the ports is marked as $(\perp,\perp)$ or $(0\circ A,0)$, but there exists at least one port that is marked as $(0\circ A,1)$, then the following actions are performed by the agents. If the agent has a $Move1$ value of 0, and the edge with respect to the lowest such port (i.e., the port marked as $(0\circ A,1)$) is missing, and no other agent is waiting for this edge, the lowest ID agent with $Move1 = 0$ waits. The other agents with the same $Move1$ value change to state **Initial**. Otherwise, if another agent is waiting for this missing edge, the agents with $Move1 = 0$ attempt to change the port. If no ports are available, they backtrack and change to the **Initial** state. On the contrary, if an edge with marking $(0\circ A,1)$ is available, they check the set A with respect to this port. If the set A contains the ID of an agent that is also trying to traverse through this edge and has $Move1$ value 0, then all these agents with $Move1 = 0$, who are trying to visit this edge, either choose a different port or backtrack and then change to state **Initial**. Otherwise, if A does not contain any ID of the agents with $Move1 = 0$ and trying to visit this port, each agent with $Move1 = 0$ moves along this port while updating $(0\circ A,1)$ to $0\circ(A\cup\{a_i\})$ (if, $\{a_i\}$ denotes the collection of all those agents) and then changes to state **Initial**. For the remaining $Move1$ values (i.e., 1, 2 and 3), the same instruction is followed as explained in earlier cases.

4. If all ports are marked with $(1,1)$ and the agent has $Move1 = 0$, then it backtracks to an already traversed node and changes its state to **Initial**. For the remaining $Move1$ values, the actions follow from earlier cases.

An agent is in the state **MissingCautious-Wait** because it has been the explorer among the set of agents executing *cautious* walk, and currently, it is waiting. The purpose of its waiting is as follows: it has tried to traverse along a port that is marked as $(\perp,\perp)$, but while traversing, it found that edge to be missing. So, currently, it waits for the missing edge to reappear, and whenever the edge reappears, it will change to state **Initial**.

Next, an agent is in state **Cautious-Wait** because it is one of the follower agents performing a *cautious* walk, where the explorer agent has traversed along the port, say j, and has yet to return. If the edge exists, this agent waits for at most two rounds, and the following conclusions are established.

(a): If the explorer agent returns and the edge still exists, it moves to this new node. While moving, it updates the g_1 value, $Apath_{a_i}$ sequence and sets $Move1 = 1$. Thereafter, it

changes to state **Initial**. On the other hand, if the edge does not exist after the explorer agent returns, it changes to state **Initial**.

(b): If the explorer agent does not return, the agent concludes that the adjacent node is the black hole.

Otherwise, if the edge does not exist, and the current agent is of the lowest ID among all agents in this state, it waits. Otherwise, it changes to state **Initial** and chooses a different port, or if no ports are available, it backtracks and changes to state **Initial**.

An agent in the state **Marking** can encounter the following types of ports: (1) a port marked as $(\perp,\perp)$, (2) a port marked as $(0\circ A,0)$, (3) a port marked as $(0\circ A,1)$, (4) a port marked as $(1,1)$.

1. An agent finds at least one port marked as $(\perp,\perp)$ and, if that agent is with $Move2=0$, then the lowest port among them is chosen. Let that port be j. Now, if the edge with respect to port j exists, then the agent moves along this edge while updating $(\perp,\perp)$ to $(0\circ\{a_i\},0)$ (where a_i be the agent). It should be noted that this update is applied to both ports of this edge, i.e., both the outgoing and incoming ports. The agent a_i also changes $Move2$ value to 1 from 0. Otherwise, if the edge with respect to port j is missing and no other agent is waiting for this edge, a_i waits. Or, if another agent is already waiting, a_i chooses another port and continues to execute this state. If none of the ports is available, then a_i backtracks and changes to state **Initial**.

Otherwise, if a_i is with $Move2=1$, that means it has already traversed along an unexplored port, and currently, it is waiting to return to the previous node and mark the corresponding edge safe. In this situation, if the edge exists, it moves back to the previous node, sets $Move2=0$, and tries to return. If, after returning, the edge has gone missing, then it waits until the missing edge reappears. After moving back to the new node with $Move2=0$, it changes to state **Initial**.

2. The agent finds no ports with marking $(\perp,\perp)$, but at least one port exists that is marked as $(0\circ A,0)$. If the agent (say, a_i) has $Move2=0$, then it waits for 2 rounds if no other agent is already waiting. After waiting, if no agent returns, then a_i locates the black hole and terminates the algorithm. Otherwise, if some agent returns along this port and marks g_2 value to 1, then a_i changes its state to **Initial**. If, on the other hand, some agent is already

waiting, then a_i chooses a different port, and if no such port exists to choose from, then a_i backtracks and changes to state **Initial**. On the contrary, if a_i has $Move2 = 1$, then it follows the same instruction as discussed in the earlier case.

3. The agent (say, a_i) finds no ports with marking $(\perp, \perp)$ or $(0 \circ A, 0)$, but finds at least one port with marking $(0 \circ A, 1)$. If $Move2 = 0$ and the set A with respect to this port does not contain the ID of a_i, then a_i moves along this port while updating g_2 to $(0 \circ (A \cup \{a_i\}))$ with respect to both ports of this edge, provided that the edge is not missing. After reaching the adjacent node, it changes to state **Initial**. Otherwise, if A contains the ID of a_i, then a_i chooses a different port. If no ports are available, it changes to the **Initial** state.

Otherwise, if the port that is marked with $(0 \circ A, 1)$ is missing, then a_i waits if no other agent is already waiting. Otherwise, if $Move2 = 1$, then a_i follows similar instructions defined in earlier cases.

4. All the ports are marked as $(1, 1)$. If $Move2 = 0$ and the edge exists, then a_i backtracks to an already traversed port and changes to state **Initial**. Otherwise, if the edge is missing and no other agents are waiting, then a_i waits. If $Move2 = 1$, then the same instruction is followed as defined earlier.

It may be noted that, irrespective of which state an agent is currently executing, whenever the agent chooses to traverse a port not explored by it previously, it not only updates the g_1 and g_2 values in the whiteboard but also stores this port. This port is updated in the set $Apath_{a_i}$, which contains the sequence of ports that the agent has traversed. The pseudo code of MULTIEDGEBHS is explained in Algorithm 11 referred to Appendix 4.6.

Correctness and Complexity

In this section, we analyse the correctness and complexity of our algorithm MULTIEDGEBHS.

Lemma 4.4.10. *Our algorithm* MULTIEDGEBHS *ensures that at most 2 agents can be stuck or waiting due to a missing edge at any round, where the underlying graph $\mathcal{G}$ is a dynamic cactus graph with at most k dynamic edges at any round.*

Proof. An agent executing MULTIEDGEBHS can encounter a missing edge along its path while it is performing *cautious* walk or *marking* walk. Irrespective of which walk it per-

forms, whenever it encounters a missing edge, it either waits for it or ignores it. First, the agent checks whether there already exists an agent waiting for that particular port (this can be understood as the agents can communicate among themselves whenever they are present at the same node). If some agent is already waiting, it ignores this missing edge. Otherwise, if no agent is waiting and if the agent is of the lowest ID among all agents performing *cautious* or *marking* walk, then it waits for the missing edge to reappear (refer to the case when $Move1 = 0$ in state **Cautious** and $Move2 = 0$ in state **Marking**, and encounters a missing edge in Algorithm 11). Hence, this shows that either end of a missing edge can be occupied by at most 2 agents. In contrast, the remaining agents that encounter them ignore this missing edge by choosing a different port or by backtracking from the current node. □

Lemma 4.4.11. *Our algorithm* MULTIEDGEBHS *ensures that at most 2 agents can be destroyed by the black hole.*

Proof. Let v be the black hole node in $\mathcal{G}$ and suppose it is part of some cycle in $\mathcal{G}$, where the two other adjacent nodes in that cycle are v_0 and v_1. As per Corollary 4.3.12, any path originating from u (where $u \in$ *Half-1*) either passes through v_0 or v_1, to reach v. So, any agent that first explores the edges (v_0, v) and (v_1, v) cannot mark these edges to be safe, as they get destroyed by the black hole the moment they reach v. As we consider, v is part of a cycle, so the adversary can remove at most one edge in this cycle at any round; otherwise, the underlying graph will get disconnected. So, while executing Algorithm 11, a round r must exist at which at least one agent is at the node v_0, trying to explore (v_0, v), and at least one other agent is at the node v_1, trying to explore (v_1, v). So, at both these nodes, they find the g_2 value of the ports corresponding to the edges (v_0, v) and (v_1, v), marked as 0, whereas at least one of these edges exists. So, after waiting for at most two rounds, either of these agents at v_0 or v_1 can successfully detect the black hole location without destroying any additional agents. Hence, this shows that our algorithm ensures that at most 2 agents can be destroyed by the black hole. □

Lemma 4.4.12. *Our algorithm* MULTIEDGEBHS *ensures that no agent explores any cycle infinitely.*

Proof. Consider C be any cycle in $\mathcal{G}$ and u be the node in C through which an agent a_i while executing Algorithm 11 starts exploring the cycle C. So, a_i may be performing *cautious* walk or *marking* walk. If a_i is performing *cautious* walk, then the edge $(u,v) \in C$ is chosen if it is marked as $(\perp,\perp)$ or $(0 \circ A, 1)$, where the set A does not contain the IDs of the agents performing *cautious* walk. So, in this case, an edge already traversed by any agent performing *cautious* walk is never chosen again. On the other hand, if a_i is performing *marking* walk, then it also chooses a port marked as $(\perp,\perp)$ or $(0 \circ A, 1)$, where A does not contain the ID of the current agent. So, in this case as well, an edge already traversed by this agent is never explored again. Hence, no agent explores a cycle infinitely. □

Lemma 4.4.13. MULTIEDGEBHS *ensures that any agent that is not stuck or waiting for a missing edge can explore the remaining graph until it gets destroyed by the black hole or detects it.*

Proof. As we have stated earlier, each agent is a t-state finite automata, where $t \geq \alpha n\Delta \log \Delta$. An agent, while executing *cautious* or *marking* walk, executes the underlying protocol of $t-$INCREASING-DFS. Now, as there are at most k dynamic edges at any round, by Lemma 4.4.10, at most $2k$ among $2k+3$ agents can be stuck or waiting for these dynamic edges. This implies that at any round, there exist at least 3 agents that have a static graph to explore. Moreover, as per Theorem 6 and Corollary 7 in [70], $O(n \log \Delta)$ bits of memory are required to explore any static graph of diameter n and maximum degree Δ. So, the agents executing MULTIEDGEBHS have more than sufficient internal memory to explore any static graph (as they have $O(n\Delta \log \Delta)$ bits of internal memory). Further, it may be noted that $O(n\Delta \log \Delta)$ bits of memory is sufficient for any agent executing Algorithm 11, because, in this case as well, the agents store the ports of each newly explored edge in $Apath_{a_i}$ (there can be at most $n\Delta$ many such edges). So, each remaining agent can explore the remaining graph until it is either destroyed by the black hole or detects it. □

Theorem 4.4.14. *Our algorithm* MULTIEDGEBHS *ensures that it requires at most* $2k+3$ *agents to successfully locate the black hole position on a dynamic cactus graph* $\mathcal{G}$ *with at most k dynamic edges at any round.*

Proof. By Lemma 4.4.10, we have shown that at most 2 agents can be stuck or waiting due to a dynamic edge. Now, at any round, there can be at most k dynamic edges, which implies that at most $2k$ agents can be stuck or waiting due to these dynamic edges. Moreover, by Lemma 4.4.11, it is shown that at most 2 agents can get destroyed by a black hole. Also, Lemma 4.4.13 ensures that any agent, not stuck or waiting, can explore the graph until it either detects the black hole or gets destroyed by it. Hence, the remaining agent among $2k+3$ agents can traverse along the graph unobstructively. Whenever it finds a node adjacent to a black hole, i.e., along which a port is marked unsafe (i.e. g_2 value with respect to a port is 0) on the whiteboard, it waits for at most two rounds. If no agent returns and updates g_2 to 1, then this remaining agent correctly concludes that the node with respect to the unsafe port is the black hole. □

Theorem 4.4.15. *A set of $2k+3$ agents executing* MULTIEDGEBHS, *locates the black hole in a dynamic cactus graph $\mathcal{G}$ within $O(kn)$ rounds and $O(k^2n)$ moves, where at any round, $\mathcal{G}$ can have at most k dynamic edges.*

Proof. Let us suppose the dynamic cactus graph $\mathcal{G}$ contains k many cycles. Let these cycles are denoted by C_i, where we define $|C_i| = l_i$ for all $i \in \{1,2,\cdots,k\}$ and $l_i \geq 3$, $l_i \in \mathbb{N}$. Since $\mathcal{G}$ contains k cycles, this means at most k edges can be dynamic at any round. This shows that, as per Theorem 4.4.14, to execute MULTIEDGEBHS on $\mathcal{G}$, $2k+3$ agents are required.

We first analyse the round complexity that our algorithm requires in order to explore the cycle $C_i \in \mathcal{G}$. Let us suppose α many agents (where $\alpha > 5$) are currently at a node $u \in C_i$, where C_i is yet to be explored. Now, as there are multiple agents at u, they start performing *cautious* walk, with the lowest ID agent (say a_{j1}) becoming the explorer, i.e., the first agent to explore an unexplored node. So, they start their movement, and without loss of generality, we assume that their movement is along the clockwise direction. While moving, suppose they encounter a missing edge $(v,v') \in C_i$. In the worst scenario, this missing edge separates the explorer from the remaining $\alpha-1$ agents. Now, as per our algorithm, the lowest ID agent (say, a_{j2}) among $\alpha-1$ agents must wait for the return of a_{j1} after the edge (v,v') reappears. So, this implies the edge (v,v') separates a_{j1} and a_{j2} from the remaining $\alpha-2$ agents. In this situation, the remaining agents following Algorithm 11 must find an

available port (i.e., a port which is either marked as $(\perp, \perp)$, or $(0 \circ A, 1)$, where A does not contain the ID of these $\alpha - 2$ agents). To find this available port, they may need to backtrack as well if no ports are available at the current node (i.e., from v' in this case). Let us assume that these $\alpha - 2$ agents need to eventually backtrack to u, which we have previously considered to be the starting node of C_i. Next, from u, there exists an unexplored port in the counter-clockwise direction. This can lead these $\alpha - 2$ agents to start performing *cautious* walk in a counter-clockwise direction, with the lowest ID agent among them (say, a_{j3}) being the explorer.

In the meantime, when these $\alpha - 2$ agents have already moved from v' towards u, the adversary reappears (v, v'), which eventually makes a_{j1} and a_{j2} to reunite. After reuniting, they continue exploring in a clockwise direction. On the other hand, suppose after the adversary reappears (v, v'), it again removes an edge (w, w') at some other round. Now, this edge (w, w') can belong on the path of exploration of these $\alpha - 2$ agents, which again, by the earlier argument can separate two agents, namely a_{j3} and the lowest ID agent among the remaining $\alpha - 3$ agents (a_{j4}, say). Once more, the remaining $\alpha - 4$ agents (i.e., except a_{j1}, a_{j2}, a_{j3} and a_{j4}) start finding an available port. So, in the quest to find a black hole, a situation can arise again where these $\alpha - 4$ agents need to backtrack up to v' until they find an available port. This shows that in order to separate 4 agents from a group of α agents, our algorithm requires $O(l_i)$ rounds (more precisely, at most $4 \cdot l_i$ rounds). Going by the same argument, if $\alpha = 2k + 3$, then to separate them by reappearing and disappearing edges in C_i, our algorithm requires at most $O(\alpha \cdot l_i) = O(k \cdot l_i)$ rounds. It may be noted that this separation can be performed for each k such cycles. So the total number of rounds required to detect the black hole is: $O(k \cdot \sum_{i=1}^{k} l_i) = O(kn)$, it is because $O(\sum_{i=1}^{k} l_i) = O(n)$ (this comes from a well known result, that the number of at most edges in a cactus graph of size n is $\lfloor \frac{3(n-1)}{2} \rfloor$, refer to page 160 in [124]). Moreover, note that, at each round, at most $2k+3$ agents move along a single edge. Hence the worst case move complexity is $O(k^2 n)$. □

Remark 4.4.16. There is a fundamental difference between the algorithm presented in Section 4.4.1 and that of the one presented in Section 4.4.2. It is because the algorithm presented for the single dynamic edge case is optimal in terms of agents. So, there exist cer-

tain scenarios where both a_1 and a_2 get destroyed by the black hole. In these scenarios, the $Leader$, even knowing which agent has been destroyed by the black hole and also the exact path to traverse in order to locate the black hole, cannot do so. The reason is that after 2 agents are destroyed, only the $Leader$ remains alive, and the adversary still has the power to remove any one edge, so what it can do is that whenever the $Leader$ tries to follow the path to the black hole, the adversary blocks it. If it tries to visit through a separate path, the adversary reappears the earlier edge and disappears a new edge such that it is again blocked. This shows that the $Leader$ can never reach the last reported node of one of the agents, which has been destroyed by the black hole, and identify the exact black hole node. So, even if the $Leader$ understands the path to visit in order to locate the black hole, the algorithm still terminates in that case.

The algorithm, explained for the multiple dynamic edges case, does not face these issues. Since a sufficient number of agents are present, an agent can know the black hole position only if it reaches one of the adjacent nodes through which another agent has already been destroyed. So, the terminating agent knows the exact black hole node.

4.5 Conclusion

In this chapter, we have studied the BHS problem in a dynamic cactus for two types of dynamicity. We have proposed algorithms, lower bounds and upper bound complexities in terms of the number of agents, rounds and moves for each case of dynamicities. First, we have studied at most one dynamic edge case, where we have shown that, with 2 agents, it is impossible to find the black hole, and correspondingly designed a BHS algorithm for 3 agents. Our algorithm is tight in terms of the number of agents. Second, we studied the case when at most k edges are dynamic. In this case, we also propose a BHS algorithm with $2k+3$ agents. Further, we have proposed that it is impossible to find the black hole with $k+1$ agents in this scenario.

However, as per the problem definition in this chapter, it is only sufficient for the agent to either identify exactly one port leading to the black hole or to know at least one port leading to the black hole by traversing its stored sequence of ports in the footprint graph.

This termination condition does not guarantee that the terminating agent knows exactly the position of the black hole in the underlying graph. Hence, an immediate future work is to design such BHS strategies where the terminating agent must know the exact position of the black hole in the underlying dynamic graph. This strategy will trivially render the destruction of more agents, but it will be an interesting direction to be looked upon. Another interesting future work may be to design an optimal algorithm in terms of the number of agents when the underlying graph has at most k dynamic edges, in the same setting. Further, obtaining an optimal algorithm in terms of complexity in both cases of dynamicity is another obvious future direction.

4.6 Appendix

In this section, we present the following pseudo-code.

- SINGLEEDGEBHSAGENT, refer to Algorithm 9, defined in Section 4.4.1.
- SINGLEEDGEBHSLEADER, refer to Algorithm 10, defined in Section 4.4.1.
- MULTIEDGEBHS, refer to Algorithm 11, defined in Section 4.4.2.

Algorithm 9: SINGLEEDGEBHSAGENT(a_i)

Input: $\{n, home\}$
States: {**Cautious**, **Pendulum**}
Initialize $Move1 = 0$, $Move2 = 0$, $Apath_{a_i} = NULL$. // $Move1$ variable is used for state Cautious and $Move2$ variable is used for state Pendulum.
In State: **Cautious**
if *a missing edge is encountered* **then**
    Wait
else
    if $Move1 = 0$ // $Move1 = 0$ implies the agent is ready to explore
    then
        if *current node has at least one port marked as* $\perp$ **then**
            Choose $\min_{j \in deg(current-node)}\{j\}$ such that $f(j) = \perp$
            Traverse along this port. Update $f(j) = 0 \circ \{a_i\}$ at the current node and $f(m) = 0 \circ \{a_i\}$ at the adjacent port. // j, m are the outgoing and incoming port of the edge connecting current node and adjacent node, respectively.
            Update $Move1 = 1$
        else
            Backtrack while accompanying the $Leader$, along a path previously traversed.
    else if $Move1 = 1$ // $Move1 = 1$ implies the agent has reached an unexplored node.
    then
        if *a missing edge appears* **then**
            Wait, until the missing edge reappears.
        else
            Move along the port m and set $Move1 = 2$. // $Move1 = 2$ means that the agent has started backtracking after exploring a new node.
    else
        if *current node does not have* $Leader$ **then**
            Follow `Last.Leader`.
        else
            If the agent has followed `Last.Leader` then change $Move1$ to 0. Otherwise, wait for the instruction of the $Leader$. If no instruction given, then accompany the $Leader$ through port j and change $Move1$ to 0.
In State: **Pendulum**
if *a missing edge is encountered along its direction of movement and the node is vacant* **then**
    Wait.
else
    if $Move2 = 0$// $Move2 = 0$ implies that the agent is ready to explore
    then
        if *the current node has at least one port marked as* $\perp$ **then**
            Choose $\min_{j \in deg(current-node)}\{j\}$ such that $f(j) = \perp$
            Traverse along this port and update $f(j) = 0 \circ \{a_i\}$ at the current node and $f(m) = 0 \circ \{a_i\}$ at the adjacent port. // j, m are the outgoing and incoming port of the edge connecting current node and adjacent node, respectively.
            Update $Move2 = 1$ and $Apath_{a_i} = Apath_{a_i} \cup (j, m)$.
        else if *the current node has at least one port marked as* $f(j) = 0 \circ (A \setminus \{a_i\})$ **then**
            Choose $\min_{j \in deg(current-node)}\{j\}$ such that $f(j) = 0 \circ (A \setminus \{a_i\})$
            Traverse along this port and update $f(j) = 0 \circ \{A \cup a_i\}$ at the current node as well as at the adjacent port.
            Update $Move2 = 1$ and $Apath_{a_i} = Apath_{a_i} \cup (j, m)$. // $Move2 = 1$ implies the agent has already performed a new exploration.
        else
            Set $Move2 = 3$ // $Move2 = 3$ implies that no port to explore at current node
    else if $Move2 = 1$ **then**
        if *current node has the* $Leader$ **then**
            Communicate the $Move2$ value with the $Leader$.
            Set $Move2 = 2$ and choose the first port in $Apath_{a_i}$ and follow it.
        else
            if *the sequence* $Apath_{a_i}$ *is exhausted* **then**
                Backtrack following the `Last.Leader` for every new node traversed, update $Apath_{a_i} = (j, m) \cup Apath_{a_i}$ // (j, m) is the incoming and outgoing port of the edge along which a_i has traversed following Last.Leader
            else
                Backtrack following $Apath_{a_i}$.
    else if $Move2 = 2$ **then**
        if *the current port chosen is the last port of* $Apath_{a_i}$ **then**
            Set $Move2 = 0$.
        else
            Continue following the sequence $Apath_{a_i}$.
    else if $Move2 = 3$ **then**
        if *at the current node at least one port exists marked as* $\perp$ *or* $0 \circ (A \setminus \{a_i\})$ **then**
            Set $Move2 = 4$.
            Update $Apath_{a_i}$ to store the sequence of path from the last position of the $Leader$ to the current node and also set $Alen_{a_i} = |Apath_{a_i}|$, where $|Apath_{a_i}|$ implies the cardinality of $Apath_{a_i}$.
        else
            if *current node has the* $Leader$ **then**
                Communicate $Move2$ value to the $Leader$ and continue backtrack.
            else
                Backtrack to an already traversed node.
    else if $Move2 = 4$ **then**
        if *current node has the* $Leader$ **then**
            Communicate $Move2$, $Alen_{a_i}$ and $Apath_{a_i}$ with the $Leader$.
            Move until it reaches the last node, following the sequence $Apath_{a_i}$ and after reaching set $Move2 = 0$.
        else
            Keep finding the $Leader$, following $Apath_{a_i}$ or `Last.Leader`.

Algorithm 10: SINGLEEDGEBHSLEADER

Input={$n, home$}
States: {**Initial**, **Assign**, **Movement**, **Missing-Edge-Leader**, **Fail-to-Report**, **Fail-to-Report-Movement**}
In State: **Initial**
Set $Alen_{a_i} = Llen_{a_i} = 0$ and $Apath_{a_i} = Lpath_{a_i} = NULL$, where $i \in \{1,2\}$ and initialize $W_{time}1 = 0$.
Instruct a_1 to change to state **Cautious** and a_2 to change to state **Pendulum** in SINGLEEDGEBHSAGENT and change to state **Assign**.
In State: **Assign**
if a_m *is instructed to change to state* ***Cautious*** **then**
  Update $Alen_{a_m} = Llen_{a_m} = 0$ and $Apath_{a_m} = Lpath_{a_m} = NULL$. Change to state **Movement**.// a_m `can be either` a_1 `or` a_2
else
  Update $Alen_{a_m} = Llen_{a_m} = 0$ and $Apath_{a_m} = Lpath_{a_m} = NULL$. Change to state **Movement**.
In State: **Movement**
if *a missing edge is encountered* **then**
  Change to state **Missing-Edge-Leader**.
else
  if a_m *performing cautious walk returns along port* j *and the* $Leader$ *has not moved from the node at which* a_m *last reported* **then**
    Traverse along the port j.
    Update `Last.Leader` at port j to 1 and other to 0. Also update $Llen_{a_n} = Llen_{a_n} + 1$ and $Lpath = Lpath \cup (j,m)$ // a_m `and` a_n `are agents among` a_1 `and` a_2 `and` (j,m) `is the outgoing and incoming port of the edge taken by the` $Leader$
  else if a_m *performing pendulum walk, returns along port* j *with* $Move2 = 1$ **then**
    Update $Alen_{a_m} = Alen_{a_m} + 1$ and gather the updated $Apath_{a_m}$ from a_m.
  else if a_m *performing pendulum walk, returns along port* j *with* $Move2 = 3$ **then**
    Do not update $Alen_{a_m}$ and $Apath_{a_m}$
  else if a_m *performing pendulum walk, returns along port* j *with* $Move2 = 4$ **then**
    Update $Apath_{a_m}$, $Alen_{a_m}$ and instruct a_m to follow $Apath_{a_m}$ till the last node and then continue in state **Pendulum**.
In State: **Missing-Edge-Leader**
if *current node is adjacent to a missing edge, and it is vacant* **then**
  Wait at the current node and change to state **Fail-to-Report** if an agent fails to report.
else
  if *the edge is yet to reappear* **then**
    Wait and instruct the agent at current node to perform state **Pendulum**. Change to state **Assign** and set $W time1 = 0$.
  else
    Instruct the agent at current node to perform state **Cautious**. Change to state **Assign** and set $W time1 = 0$.
if *the* $Leader$ *is waiting for the missing edge, and it reappears* **then**
  if *an agent reports* **then**
    Instruct the agent to continue earlier walk. Change to state **Assign** and set $W time1 = 0$.
  else
    If previous state was **Fail-to-Report**, then continue performing **Fail-to-Report**. Else change to state **Fail-to-Report**.
In State: **Fail-to-Report**
if a_m *fails to report within 2 rounds while performing cautious walk* **then**
  if *there exists an adjacent missing edge* **then**
    Change to state **Missing-Edge-Leader**.
  else
    Conclude the node visited by a_m is the black hole node and terminate the algorithm.
else if a_m *does not report within* $2 \cdot (Alen_{a_m} + Llen_{a_m} + 1)$ *rounds, while performing pendulum walk* **then**
  if $W time1 > 2$ **then**
    if *current node has* a_n **then**
      if a_n *is performing cautious walk* **then**
        Instruct a_n to change to **Pendulum**. Set $Apath_{a_n} = Lpath_{a_n} = NULL$ and $Alen_{a_n} = Llen_{a_n} = 0$.
        Move towards a_m following $Apath_{a_m}$, while updating $Llen_{a_n} = Llen_{a_n} + 1$ and $Lpath_{a_n} = Lpath_{a_n} \cup (j,m)$.
      else
        Remain at the current node, instruct a_n to continue state **Pendulum**. Update the parameters according to state **Movement**.
    else
      if *the edge exists and there exists port in* $Apath_{a_m}$ *to be traversed* **then**
        Move towards a_m following $Apath_{a_m}$, while updating $Llen_{a_n} = Llen_{a_n} + 1$ and $Lpath_{a_n} = Lpath_{a_n} \cup (j,m)$.
      else if *the edge does not exist but there exists port in* $Apath_{a_m}$ *to be traversed* **then**
        Change to state **Missing-Edge-Leader**
      else if *the edge does not exist and there is no port in* $Apath_{a_m}$ *to be traversed* **then**
        Change to state **Missing-Edge-Leader**
      else if a_m *is found* **then**
        If no missing edge is found, then instruct a_m to perform state **Cautious** and change to state **Assign**. Otherwise change to state **Missing-Edge-Leader**.
      else
        Conclude a_m is destroyed by the black hole.
  else
    $W time1 = W time1 + 1$.
else if a_m *performing cautious walk, does not return within 2 round, and,* a_n *performing pendulum walk, does not return within* $2 \cdot (Alen_{a_n} + Llen_{a_n} + 1)$ *rounds* **then**
  if *the edge between* a_m *and* $Leader$ *is missing* **then**
    Conclude the last visited node of a_n as the black hole and terminate the algorithm.
  else
    Conclude the adjacent node visited by a_m is the black hole and terminate the algorithm.
else if *both* a_m *and* a_n *performing pendulum walk does not return within* $\sum_{i=1}^{2} (2 \cdot (Alen_{a_i} + Llen_{a_i} + 1))$ *round* **then**
  Change to state **Fail-to-Report-Movement**.
In state: **Fail-to-Report-Movement**
if *a missing edge is encountered* **then**
  Wait at the current node.
  if *wait time greater than* $\max_{i \in \{1,2\}} (2 \cdot (Alen_{a_i} + Llen_{a_i} + 1))$ **then**
    if *no agent reports* **then**
      Conclude that the last visited node of the agent with $\max_{i \in \{1,2\}} (2 \cdot (Alen_{a_i} + Llen_{a_i} + 1))$ is the black hole.
    else
      Change to state **Fail-to-Report**.
else
  Move towards the agent with $\min_{i \in \{1,2\}} (2 \cdot (Alen_{a_i} + Llen_{a_i} + 1))$.
  if *the agent is found* **then**
    Set $Alen_{a_m} = Llen_{a_m} = 0$, $Apath_{a_m} = Lpath_{a_m} = NULL$ and instruct a_m to change to state **Pendulum**. // a_m `be the agent found.`
    Change to state **Fail-to-Report**.
  else
    Conclude the last visited node of the agent with $\min_{i \in \{1,2\}} (2 \cdot (Alen_{a_i} + Llen_{a_i} + 1))$ is the black hole.

Algorithm 11: MULTIEDGEBHS(a_i)

```
1  Input = {n, home}
2  States: {Initial,Cautious, Marking, MissingCautious-Wait, Cautious-Wait}
3  Initially set Move1 = W_time1 = Move2 = W_time3 = 0
4  In State: Initial
5  if the current node has more than one agent then
6      Change to state Cautious.
7  else
8      Change to state Marking.
9  In State: Cautious.
10 if a port exists at the current node marked as (⊥, ⊥) then
11     if Move1 = 0 then
12         if the edge with respect to lowest such port exists then
13             if a_i is the lowest ID agent at the current node, in state Cautious then
14                 Set Move1 = 2 and traverse along this port, while marking
                   (0 ∘ {a_i}, 0) at both the ports of the edge to this adjacent
                   node, update Apath_{a_i} = Apath_{a_i} ∪ (j, m).// j−th
                   port is the lowest such outgoing port at the
                   current node, m−th port is the incoming port
                   of the adjacent node
15             else
16                 Set W_time2 = 0 and change to state Cautious-Wait.
17         else
18             If no agent is waiting and a_i is the lowest ID agent, then it changes
               to state MissingCautious-Wait whereas other agents change to
               state Initial. Otherwise, if an agent is waiting for this edge, then
               choose a different port, if no port exists to choose, then backtrack
               and change to state Initial.
19     else if Move1 = 1 then
20         if there exists an agent with Move1 = 0 then
21             Set Move1 = 0 and change to state Cautious. Also set W_time2 = 0.
22     else if Move1 = 2 then
23         If the edge with respect to port m exists, then return to previous node
           while updating g_2(m) = 1 and g_2(j) = 1, and set Move1 = 3. Otherwise,
           if the edge is missing, then wait.
24     else
25         Move along port j and then set Move1 = 0 and change state to Initial.
           Otherwise, if edge is missing then wait.
26 else if a port j exists marked with (0 ∘ A, 0) exists then
27     if Move1 = 0 then
28         if the edge is missing and no agent is waiting then
29             If a_i is the lowest ID agent with Move1 = 0, then wait. Otherwise,
               change to state Initial.
30         else if the edge exists and no agent is waiting then
31             if W_time1 > 1 then
32                 If g_2 with respect to this node is marked 1, then change to
                   state Initial and also set W_time1 = 0. Otherwise, conclude
                   the adjacent node to be the black hole.
33             else
34                 W_time1 = W_time1 + 1.
35         else
36             Choose a different port. Otherwise, if no port exists to choose from,
               then backtrack and change to state Initial.
37     else if Move1 = 1 then
38         if there exists an agent with Move1 = 0 then
39             Set Move1 = 0 and change to state Cautious also set W_time2 = 0.
40     else if Move1 = 2 then
41         If the edge with respect to port m exists, then return to previous node,
           while updating g_2(m) = 1 and g_2(j) = 1, and set Move1 = 3. Otherwise,
           if the edge is missing, then wait.
42     else
43         Move along port j and then set Move1 = 0 and change state to Initial.
           Otherwise, if edge is missing, then wait.
44 else if a port j exists marked as (0 ∘ A, 1) then
45     if Move1 = 0 then
46         if the set A contains ID of an agent present at current node which is not
           waiting for any missing edge then
47             Choose a different port or backtrack and change to state Initial.
48         else
49             if the edge is missing and no agent is waiting for this edge then
50                 If a_i is the lowest ID agent, then wait. Otherwise change to
                   state Initial.
51             else if the edge exists but no agent is waiting for this edge then
52                 Move along this port and update g_1(j) = g_1(m) = 0 ∘ (A ∪ {a_i})
                   and Apath_{a_i} = Apath_{a_i} ∪ (j, m), set Move1 = 0 then
                   change to state Initial.
53             else
54                 Choose a different port and if no port exists to choose then
                   backtrack and change to state Initial.
55     else if Move1 = 1 then
56         If another agent with Move1 = 0 is present, then set Move1 = 0 and change
           to state Cautious also set W_time2 = 0.
57     else if Move1 = 2 then
58         If the edge with respect to port m exists, then return to previous node,
           while updating g_2(m) = 1 and g_2(j) = 1, and set Move1 = 3. Otherwise,
           wait.
59     else
60         Move along port j and then set Move1 = 0 and change state to Initial.
           Otherwise, if the edge is missing then wait.
61 else
62     if Move1 = 0 then
63         Backtrack to an already traversed node, and change to state Initial.
64     else if Move1 = 1 then
65         If there exists another agent with Move1 = 0, then set Move1 = 0 and
           change to state Cautious and set W_time2 = 0.
66     else if Move1 = 2 then
67         If the edge with respect to port m exists, then return to previous node,
           while updating g_2(m) = 1 and g_2(j) = 1, and set Move1 = 3. Otherwise,
           wait.
68     else
69         Move along port j and then set Move1 = 0 and change state to Initial.
           Otherwise, if the edge is missing, then wait.
70 In State: MissingCautious-Wait
71 if the edge is missing then
72     Wait.
73 else
74     Change state to Initial
75 In State: Cautious-Wait
76 if W_time2 > 1 then
77     if the edge exists then
78         if there is an agent which returned along port j with Move2 = 3 then
79             if the edge exists then
80                 Move along this port update g_1(j) = g_1(m) = 0 ∘ (A ∪ {a_i}) and
                   Apath_{a_i} = Apath_{a_i} ∪ (j, m), set Move1 = 1 and change
                   to state Initial.
81             else
82                 Change to state Initial.
83         else
84             Conclude the adjacent node to be black hole.
85     else
86         If the agent is of lowest ID among all the agents which are in state
           Cautious-Wait, then wait. Otherwise change to state Initial.
87 else
88     W_time2 = W_time2 + 1
```

```
In State: Marking
if a port exists at current node marked as (⊥,⊥) then
    if Move2 = 0 then
        if the edge with respect to lowest such port exists then
            Set Move2 = 1 and traverse that port, while marking (0∘{a_i},0) at
              both the outgoing port j and incoming port m, update
              Apath_{a_i} = Apath_{a_i} ∪ (j,m).
        else
            If no agent is already waiting, then wait. Otherwise, choose a
              different port and if no port exists to choose from, then backtrack
              and change to state Initial.
    else if Move2 = 1 then
        If the edge with respect to port m exists then return to previous node, while
          updating g2(m) = g2(j) = 1, otherwise wait. If returned then in the next
          round move along port j and set Move2 = 0, change to state Initial.
          Otherwise if the edge is missing, then wait.
else if a port exists at the current node marked with (0∘A,0) then
    if Move2 = 0 then
        if the edge exists then
            if W_time3 > 1 then
                If g2 with respect to this port is marked 1, then change to
                  state Initial and set W_time3 = 0. Otherwise, conclude the
                  node with respect to this port is the black hole.
            else
                If no other agent is waiting, then set W_time3 = W_time3 + 1.
        else
            Wait, if no other agent is already waiting, and a_i is the lowest ID
              among all agents with Move2 = 0. Otherwise, choose a different
              port and if no port exists then backtrack and change to state
              Initial.
    else if Move2 = 1 then
        If the edge with respect to port m exists, then return to previous node,
          while updating g2(m) = g2(j) = 1, otherwise wait. If returned then in the
          next round move along port j and set Move2 = 0, change to state Initial.
          Otherwise if missing edge then wait.
else if a port exists at the current node marked with (0∘A,1) then
    if Move2 = 0 then
        if the edge exists then
            if A does not contain a_i then
                Move the lowest among such port and update
                  g2(j) = g2(m) = 0∘(A∪{a_i}),
                  Apath_{a_i} = Apath_{a_i} ∪ (j,m) and change to state Initial.
            else
                Choose a different port if exists or backtrack and change to
                  state Initial.
        else
            Wait, if no other agent is already waiting, and a_i is the lowest ID
              among all agents with Move2 = 0. Otherwise, choose a different
              port and if no port exists then backtrack and change to state
              Initial.
    else if Move2 = 1 then
        If the edge with respect to port m exists, then return to previous node,
          while updating g2(m) = g2(j) = 1, otherwise wait. If returned then in the
          next round move along port j and set Move2 = 0, change to state Initial.
          Otherwise if the edge is missing, then wait.
else
    if Move2 = 0 then
        if the edge exists then
            Backtrack to an already traversed port, and change to state Initial
        else
            Wait, if no other agent is already waiting, and a_i is the lowest ID
              among all agents with Move2 = 0. Otherwise, change to state
              Initial.
    else if Move2 = 1 then
        If the edge with respect to port m exists, then return to previous node,
          while updating g2(m) = g2(j) = 1, otherwise wait. If returned along port
          j, then in the next round move along port j and set Move2 = 0, change
          to state Initial. Otherwise if missing edge then wait.
```

Chapter 5

Perpetual Exploration of a Ring with a Byzantine Black Hole

5.1 Introduction

In this chapter, we introduce a more powerful version of the black hole, termed as *Byzantine black hole* (BBH) in comparison to the classical black hole studied in the previous chapter. The malicious node behaviour of the BBH is far more complex than that of a classical black hole. So, in order to detect the BBH, just exploration of the network may not be sufficient; there may be a need for perpetual exploration. BBH is an extension of the earlier studied *gray hole* by Královič et al. [88], Miklík [101] and Bampas et al. [13]. Definitions of all these different variations of black hole can be found in Section 1.1.2 of Chapter 1.

Unlike in Chapter 4, here we consider our underlying topology not to be dynamic but a static ring. The idea behind considering this particular topology lies behind the fact that all the previous studies of a variation of black hole by Královič et al. [88], Miklík [101] and Bampas et al. [13] considered only static rings. In addition to that, these studies only focused on the fact that the agents are initially co-located, with each node contains a whiteboard, and these agents are controlled by an asynchronous scheduler. We, during our study, found that not only are the results obtained by them not optimal, but a natural question came to

[5]This chapter has been published as: "Perpetual Exploration of a Ring in Presence of Byzantine Black Hole" in *Proceedings of the 28th International Conference on Principles of Distributed Systems (OPODIS 2024)*.

our mind: what about other initial configurations (the agents may start in a scattered configuration, i.e., the agents can have more than one initial starting position), or what about other models of communication? Hence, in this chapter, we investigated the problem of perpetual exploration in the presence of a BBH, under all possible initial configurations (i.e., the agents may be co-located or they can start from scattered positions), and in all fundamental models of communication (i.e., face-to-face, pebble and whiteboard). The agents are controlled by a synchronous scheduler, unlike previous studies in the literature of an asynchronous scheduler. Changing the scheduler also gave better bounds, as can be found in this chapter. Almost all the results we obtained are optimal, and there are certain interesting open questions that lie ahead for further investigation.

A list of all the prominent notations used in this chapter is explained in Table 5.1.

Notation	**Meaning**
$R = (V, E)$	Ring with vertex set V and edge set E
n	Cardinality of V
v_i	$V = \{v_0, v_1, \ldots, v_{n-1}\}$ indicates nodes in the ring network
v_b	Indicates the BBH node in R
$A = \{a_0, \ldots, a_{k-1}\}$	Indicates the k agents
$home$	Starting position of all co-located agents
h_i	Starting position of agent a_i in scattered configuration
SUS	Set of all suspicious nodes
|SUS|	Cardinality of the set SUS
SUS_1, SUS_2	Two possible locations of BBH identified by the agents
p_i	Indicates the i-th pebble
$Seg(a_i)$	Set of nodes in segment of a_i
$(home, ID(a_i))$	Home type message of a_i at h_i
$(visited, ID(a_i))$	Visited type message of a_i at last node of $Seg(a_i)$
SC_i	Indicates scenario i, where v_i is the BBH location

Table 5.1: Main notations used in Chapter 5

5.1.1 Our Contribution

In this chapter, we investigate the perpetual exploration problem, by a team of synchronous mobile agents, of a ring R of size n, in the presence of a BBH. First, we consider the case when the agents are initially co-located. We obtain the following results.

1. For the *pebble* model of communication, we obtain that 3 agents are necessary and sufficient to perpetually explore R.

2. For the *face-to-face* model of communication, 4 agents are sufficient to perpetually explore R. In addition, 3 agents are necessary to solve this problem under face-to-face model of communication.

3. For the *whiteboard* model of communication, we obtain that 3 agents are necessary and sufficient to solve this problem.

Next, we consider the case when the agents are initially scattered, and in this context, we obtain the following results:

1. For the *Pebble* model of communication, we show that 4 agents are necessary and sufficient to explore R perpetually.

2. For the *Whiteboard* model of communication, we obtain an improved bound of 3 agents (in comparison to *Pebble* model of communication in the scattered configuration), which is necessary and sufficient to explore the ring R perpetually.

In the following Table 5.2, we have summarized the results.

Table 5.2: Summary of our results

		Whiteboard	**Pebble**	**Face-to-Face**
Co-located	Upper Bound	3	3	4
	Lower Bound	3	3	3
Scattered	Upper Bound	3	4	—
	Lower Bound	3	4	—

5.2 Models and Preliminaries

We consider the underlying topology to be an oriented ring, defined by $R = (V, E)$, where $V = \{v_0, v_1, \cdots, v_{n-1}\}$ and each node v_i (for some $0 \leq i \leq n-1$) is unlabeled and has two ports connecting the adjacent nodes $v_{i-1 \pmod n}$ and $v_{i+1 \pmod n}$, where they are consistently labeled as *left* and *right*. A set of k agents operates on R, denoted by the set $A =$

$\{a_0, a_1, \cdots, a_{k-1}\}$. These agents have two types of initial configuration, first, each of the k agents is *co-located* at a node, termed as $home$, and, secondly, the agents can start from several distinct nodes, for which we call the initial configuration to be *scattered*. Each agent has the knowledge of the underlying topology R and possesses some computational capabilities, thus requiring $O(\log n)$ bits of internal memory. The agents have unique IDs of size $O(\log k)$ bits taken from the set $[0, k^c]$, for some constant c; they are autonomous and execute the same algorithm. In this chapter, we consider the communication model to be any of the three distinct models, i.e., *face-to-face*, *pebble*, and *whiteboard*, by which agents can communicate with each other. In *face-to-face* model, the agents can communicate with another agent only while they are co-located. In *Pebble* model, the agents carry a movable token. These are used by agents to mark special nodes, allowing other agents to distinguish them. Finally in *whiteboard* model, each node in R contains a $O(\log n)$ bits of memory, that can be used to store and maintain information.

The agents operate in synchronous rounds, and in each round, every agent becomes active and takes a local snapshot of its surroundings. For an agent at a node v in some round r, the snapshot contains the two adjacent ports of v, already stored data on the memory of v (if any, only in case of whiteboard model of communication), the number of pebbles located at v (if any, only in case of pebble model of communication), contents from its local memory, and IDs of other agents on v. Based on the outcome of this snapshot, an agent executes some action. This action includes a *communication* step and a *move* step. In a communication step, an agent communicates implicitly or explicitly with other agents according to the communication models discussed above. In the move step, the agent can move to a neighbouring node by following a port incident to v. If an agent at a node v in round r decides to move during the move step, then in round $r+1$, it resides on a neighbour node of v. Each of these actions is atomic, so an agent cannot find another agent concurrently passing through the same edge. It can only interact with another agent (based on the communication model) only when it reaches another node.

In this chapter, we consider the underlying graph contains a single BBH, while the remaining nodes are normal nodes, termed as *safe nodes*. It is assumed that the starting positions of the agents must be safe nodes. The BBH node is unknown to the agents. We

assume that if the adversary decides to activate the black hole nature of the BBH, it does so at the beginning of its corresponding round, and the node retains the nature till the end of the round. We have also considered that the BBH can always choose to destroy any incoming agent and agents present on the node during its black hole nature, and also choose to destroy any information present on that node. The goal of our study in this chapter is to perform perpetual exploration of the ring R using the agents, in the presence of a BBH. Next, we formally define our problem:

Definition 5.2.1 (PERPEXPLORATION-BBH)**.** Given a ring network R with n nodes, where one node v_b is a BBH and with a set of agents A positioned on R, the PERPEXPLORATION-BBH asks the agent in A to move in such a way that each node of R except v_b is visited by at least one agent infinitely often.

5.3 Impossibility Results

In this section, we prove all the impossibility results, in regard to solving PERPEXPLORATION-BBH. Firstly, we find the lower bound on the minimum number of agents required to solve PERPEXPLORATION-BBH.

Theorem 5.3.1. *Two agents in a ring R of size n cannot solve* PERPEXPLORATION-BBH *even in the presence of a whiteboard, if the number of possible consecutive BBH positions is at least 3.*

Proof. Let v_1, v_2 and v_3 be three possible consecutive BBH positions in a ring R. Let the two agents be denoted by a_1 and a_2, which are sufficient to solve PERPEXPLORATION-BBH in R. It means that there exists an algorithm $\mathcal{A}$, such that these two agents solve PERPEXPLORATION-BBH by executing $\mathcal{A}$. Without loss of generality, let a_1 explore v_2, and it moves from v_1 to v_2 for the first time at round t. This means, a_1 must be present at v_1 at round $t-1$. Next, let us consider two copies of the ring R, namely R_1 and R_2. In R_1, v_1 is the BBH, whereas in R_2, v_2 is the BBH. Let the adversary in R_1 destroy a_1 at round $t-1$ and in R_2 destroy a_1 at round t. We claim that, at rounds t and $t-1$, a_2 cannot be on either v_1 or v_2. It may be noted that, at t-th round if a_2 is at v_2 or at $(t-1)$-th round a_2 is at v_1, then

both a_1 and a_2 gets destroyed at rounds $t-1$ and t in rings R_1 and R_2, respectively. Now, consider the case if at the t-th round a_2 is at v_1, this implies a_2 must have traversed from a node u_0 (adjacent to v_1 and not v_2) to v_1 at round $t-1$ or it traversed from v_2, since a_2 cannot be present at v_1 at round $t-1$.

We first show that a_2 cannot be at u_0 at round $t-1$. As the whiteboard content at nodes except v_1 and v_2 is the same in both R_1 and R_2, due to the same execution of a_1 up to round $t-1$ and the same execution of a_2 up to round t. This means, at round t whenever a_2 visits v_1, a_2 cannot distinguish between R_1 and R_2. Hence, the problem transforms to solving PERPEXPLORATION-BBH with a single agent in the presence of two possible BBH positions, which is impossible to solve. This proves that at round $(t-1)$, a_2 cannot be on u_0. Next, we prove that a_2 cannot be on v_2 at round $t-1$. Suppose in R_2, the adversary activates the BBH at round $t-1$, destroying a_2 at v_2. Now it may be observed that, the effect of this destruction of a_2 does not affect the inputs of a_1 at round $t-1$ on v_1, so inadvertently a_1 moves to v_2 at round t (as per the execution of $\mathcal{A}$), thus destroying both a_1 and a_2, eventually within round t. This implies that a_2 must not remain in u_0 or v_2 at round $t-1$. So at round t since all whiteboard contents are the same for R_1 and R_2, except at nodes v_1 and v_2 whose content only differs from round $t-1$ onwards, and this cannot be known by a_2 as it is not present at these nodes during these rounds, i.e., $t-1$ and t. Hence, this information cannot help a_2 distinguish between R_1 and R_2. So, again, the problem transforms to solving PERPEXPLORATION-BBH with a single agent in the presence of two possible BBH positions, which is impossible. This contradicts the fact that $\mathcal{A}$ solves PERPEXPLORATION-BBH with 2 agents, on a ring R, the possible consecutive BBH positions are at least 3. □

Corollary 5.3.2. *Three agents are necessary to solve* PERPEXPLORATION-BBH *on a ring R with n nodes, where each node of R has a whiteboard.*

The above corollary follows from Theorem 5.3.1. It may be noted that the lower bound of 3 agents also holds for agents with the pebble model of communication, as the pebble model of communication can easily be simulated to the whiteboard model of communication.

Corollary 5.3.3. *Two agents, each equipped with $O(\log n)$ pebbles can not solve the problem* PERPEXPLORATION-BBH *on a ring R with n nodes.*

The next theorem improves the lower bound on the number of agents when the agents are scattered and have a pebble model of communication.

Theorem 5.3.4. *Three scattered agents, each equipped with a pebble, cannot solve the problem* PERPEXPLORATION-BBH *on a ring R with n nodes.*

Proof. Let a_1, a_2 and a_3 be three agents each equipped with a pebble, where the initial positions of these agents be the three nodes h_1, h_2 and h_3, respectively. The distance between h_i and h_j is the same for all $i, j \in \{1,2,3\}$ and $i \neq j$, and it is considered that h_j is the nearest node of h_i in the clockwise direction. We further consider the fact that these distances between h_i and h_j are sufficiently large. Let $\mathcal{A}$ be an algorithm that solves PERPEXPLORATION-BBH in this setting. Now, without loss of generality we assume that, after the agents start executing $\mathcal{A}$, a_1 be the first agent to explore the third node (i.e., along the path $h_1 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3$, if v_1, v_2 and v_3 be those nodes) either in the clockwise or counter-clockwise direction. As per the execution of $\mathcal{A}$, let a_1 reach v_3 at round t (for some $t > 0$). Let us consider three scenarios, SC_1, SC_2 and SC_3, where in SC_i, v_i be the BBH, for $i \in \{1,2,3\}$. We first claim that a_1 cannot carry its pebble during any execution of $\mathcal{A}$, because if it does, then in SC_1 it would be destroyed along with a_1 at v_1. In addition to this, since the distance between two consecutive h_i (for all $i \in \{1,2,3\}$) is sufficiently large, so other agents will have no idea that any agent is already destroyed. This scenario is equivalent to solving PERPEXPLORATION-BBH with 2 agents having a pebble each, and as n is sufficiently large, it is impossible as per Corollary 5.3.3.

Suppose the adversary decides to activate the BBH whenever a_1 reaches a node among v_1, v_2 and v_3 (depending on the scenarios, i.e., SC_1, SC_2 or SC_3) for the first time. In this situation, for all three scenarios, i.e., SC_1, SC_2 and SC_3, from round t onwards, the remaining alive agents a_2 and a_3 will have no knowledge about where a_1 is destroyed, even if they know that a_1 is destroyed. It is because the distance between two consecutive h_i is sufficiently large, so their exploration region does not intersect till round t. In addition, the time-out strategy (or waiting for others) will not work, as for each SC_i, every agent gets the

same timeout output. This shows that, at round t onwards, the number of possible positions of the BBH is greater than or equal to 3, and all these possible positions are also consecutive. So the situation at round t is similar to the problem of solving PERPEXPLORATION-BBH on a ring R with two agents having a total of 3 pebbles, where the number of possible consecutive positions of the BBH is greater than or equal to 3. Since we have assumed $\mathcal{A}$ solves the problem, thus $\mathcal{A}$ can also solve the problem of PERPEXPLORATION-BBH with two agents, where the number of possible consecutive positions of the BBH is greater than or equal to 3. But due to Corollary 5.3.3, it is impossible. Hence, there can never exist any algorithm that solves PERPEXPLORATION-BBH with 3 scattered agents, each of which is equipped with a pebble. □

Corollary 5.3.5. *Four scattered agents each equipped with a pebble is necessary to solve the problem* PERPEXPLORATION-BBH*, on a ring R with n nodes.*

5.4 Perpetual Exploration with Co-located Agents

In this section, we assume that the agents start from a common node, also called *home*. With this assumption, we investigate the number of agents sufficient to solve the problem PERPEXPLORATION-BBH under different communication models.

5.4.1 Pebble Model of Communication

Here we study the PERPEXPLORATION-BBH problem in the presence of a BBH, explicitly for the pebble model of communication.

Algorithm Description

We present an algorithm PERPEXPLORE-COLOC-PBL that solves PERPEXPLORATION-BBH in the presence of 3 co-located agents with the help of an additional pebble.

Given a ring R with a BBH node v_b, and a set of agents, $A = \{a_1, a_2, a_3\}$ (we assume, ID of a_1 < ID of a_2 < ID of a_3) are initially co-located at *home*. The agents only know the following details: the total number of nodes in R (i.e., n), and the fact that *home* is safe.

This implies that the remaining $n-1$ nodes in R are suspicious for each agent. We call the set of all suspicious nodes SUS, where |SUS| denotes the cardinality of SUS.

High level idea of Algorithm: Initially, two agents a_1 and a_2 explore the ring R perpetually, while the third agent, i.e., a_3, waits at $home$. To describe the algorithm, we label the nodes of the ring R starting from $home$ in the clockwise direction as $v_0, v_1, \ldots v_{n-1}$. The agents a_1 and a_2, while perpetually exploring R, execute the following rules.

Rule-I: [All agents are at $home$ along with the pebble at some round, say $r \geq 0$]. In this case, a_1 moves clockwise to node v_1 and waits until round $r+1$ for a_2 to arrive. At round $(r+1)$, the agent a_2, leaves the pebble at $home$ and moves to v_1 to meet a_1. So, at round $(r+2)$ the agents a_1 and a_2 are together at v_1, and the pebble is at $home$, i.e., at v_0. From this configuration onwards, the agents a_1 and a_2 follow Rule-II, while a_3 waits at home.

Rule-II: [a_1 and a_2 are at v_i $(i \neq 0)$ and the pebble is at v_{i-1} at some round $r > 0$]. At round r, a_1 moves to the next node v_{i+1} in the clockwise direction and waits for 3 rounds for a_2, i.e., until round $(r+3)$. In the meantime, at round $(r+1)$, a_2 leaves v_i, moves one step in a counter-clockwise direction to v_{i-1}, and collects the pebble. Next, in round $(r+2)$, a_2 again moves to v_i along with the pebble. Subsequently, at round $(r+3)$, a_2 again leaves the pebble at v_i and moves in a clockwise direction, to meet a_1 (which is waiting at v_{i+1} for a_2 to arrive). Note that, when a_2 meets a_1 at v_{i+1}, the pebble it was carrying is left by a_2 at the nearest counter-clockwise node, i.e., at v_i (for better reference, see Figure 5.1). Also note that the same configuration translates from v_i to v_{i+1}. These agents follow Rule-II again if $i+1 \leq n-1$. For $i+1 = n$, let the agents follow Rule-II from r'-th round, so at $(r'+3)$-th round, a_2 joins a_1 and a_3 at $home$, whereas the pebble is at v_{n-1}. Next, in round $(r'+4)$, a_1 and a_2 follows Rule-III, while a_3 waits at $home$.

Rule-III: [a_1, a_2 and a_3 all are at $home$ and the pebble is at v_{n-1} at some round $r > 0$]. The agent a_1 waits at $home$ till round $(r+1)$, whereas at round r, the agent a_2 moves to v_{n-1}, collects the pebble and returns to $home$ at round $(r+1)$. Note that at round $r+2$, all agents and the pebble are located at $home$. So, from this round onwards, a_1 and a_2 start to follow Rule-I. So, if the BBH does not destroy any agent or the pebble, then R gets perpetually explored.

Observation 5.4.1. Let r be the round, at which a_1, a_2 and a_3, along with the pebble, are

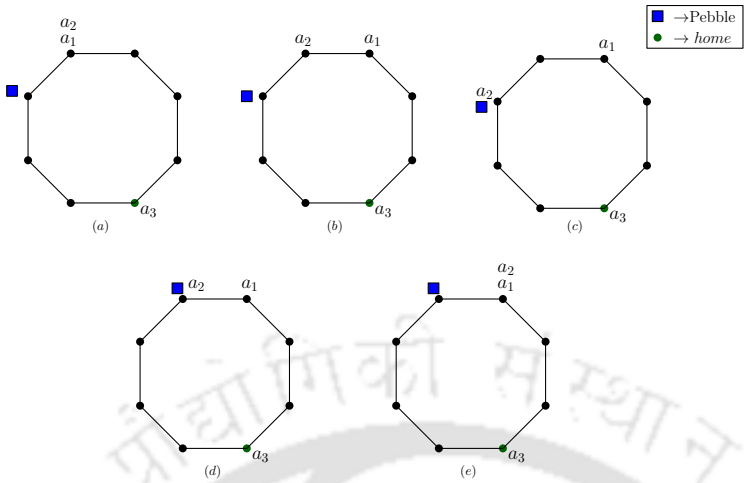


Figure 5.1: An execution of PERPEXPLORE-COLOC-PBL, starting from the configuration where a_1 and a_2 are together on a vertex, to the configuration where a_1 and a_2 are on the adjacent clockwise vertex.

at *home*. Then again, this configuration reoccurs at round $r+4n-1$ if the BBH does not intervene.

Intervention by the BBH: While a_1 and a_2 execute the above rules, if the BBH intervenes by destroying either the pebble or any agent(s), then a certain anomaly arises, which we discuss in the following cases.

Case-1: Let a_1 be the only agent, which is the first to be destroyed by the BBH at some round r (>0). This node at which a_1 gets destroyed is v_b (where $b \neq 0$). This follows that a_1 and a_2 must have been following Rule-I (if $b=1$) or Rule-II. So, within round $r+3$, anyway a_2 must also visit v_b. When a_2 visits v_b, there can be two situations: either it stays alive and finds that a_1 is not present, or it also gets destroyed at v_b by the BBH. For the first situation, a_2 knows the exact location of v_b, and it performs perpetual exploration, avoiding v_b. We define r' ($<r$) as the last round at which all agents and the pebble were at *home*, before the round r, i.e., at which the agents a_1 and a_2 start executing Rule-I. So, for the second situation at round $(r'+4n)$, a_3 finds none of the agents a_1, a_2 are present at *home*. This triggers a_3 to move clockwise. Note that the pebble which was left by a_2 during the execution of Rule-II, is at v_{b-1}. While exploring clockwise, a_3 finds the pebble at v_{b-1}, and from this phenomenon, it interprets the next node (i.e., v_b) to be the BBH. Hence, a_3 detects the location of the BBH and perpetually explores R, while avoiding BBH (i.e., v_b). A

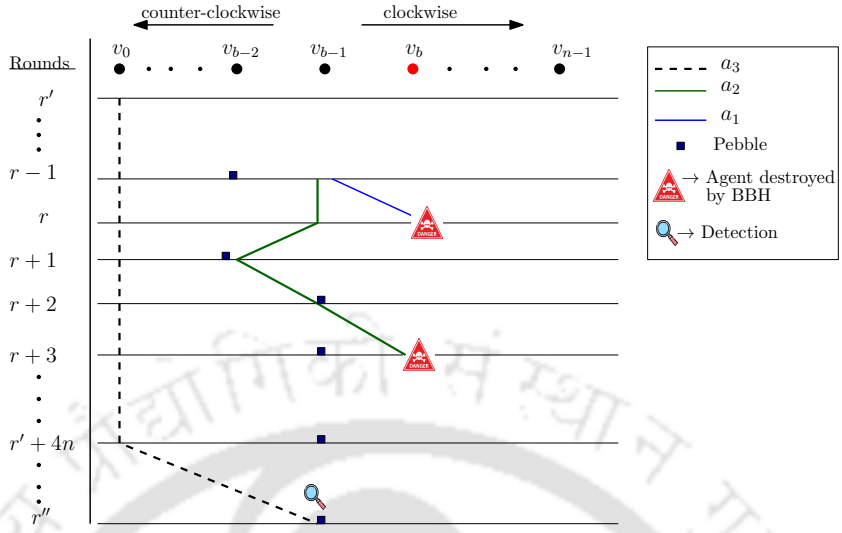


Figure 5.2: Represents the step wise time diagram of PERPEXPLORE-COLOC-PBL where at round r and $r+3$, agents a_1 and a_2 are destroyed at v_b. Starting from round $r'+4n$, a_3 moves clockwise and encounters the pebble at round r'' ($< r'+5n$), and detects the BBH.

detailed illustration of this case, where both a_1 and a_2 are destroyed by v_b, and eventually a_3 detects the BBH location, is defined via a time diagram in Figure 5.2.

Case-2: Let a_2 be the only agent, which is the first to be destroyed by the BBH at some round r. So, when a_2 is destroyed, a_1 must be at v_i where $i = b+1$ or $b+2$. Since, a_2 is destroyed at round r, so within round $r+3$, a_1 finds that a_2 fails to visit v_i. This phenomenon helps a_1 to interpret that, a_2 must have been destroyed at v_{i-1} or v_{i-2}. Note that i cannot be equal to 1 as a_1 can only be at v_1 by executing Rule-I, where Rule-I starts with all agents and the pebble being present at $home$. This implies that i cannot be equal to 1. Also note that, if $i = 2$ then a_2 can only be destroyed at v_1, as $v_0 = home$. In this case, a_1 can exactly locate the BBH and explore the ring R perpetually, avoiding v_1 (i.e., v_b).

Now for $i \neq 1,2$, there can be two sub-cases, $i = 0$ and $i > 2$. Let r' be the last round before r when all three agents and the pebble were at $home$, and from which a_1 and a_2 start executing Rule-I.

For $i = 0$, a_1 reaches v_0 at round $r'+4n-6$, and as per Rule-III it waits for a_2 till $r'+4n-3$ round. Absence of a_2's arrival at v_0 at round $r'+4n-2$ triggers both a_1 and a_3 to conclude, v_{n-1} and v_{n-2} to be the two possible locations of the BBH. They identify v_{n-1}

as $\texttt{SUS}_1$ and v_{n-2} as $\texttt{SUS}_2$. Accordingly, from $r'+4n-2$ round onwards, a_1 visits $\texttt{SUS}_1$ in counter-clockwise, and a_2 visits $\texttt{SUS}_2$ in clockwise. After visiting, they return to $home$, and wait until both of them are together at $home$. They repeat this process. Note that if no further agents are destroyed, perpetual exploration of R is achieved, as all nodes are perpetually visited by a_1 and a_3. Since these agents know n, a_1 knows exactly how many rounds to wait for a_3 to return after visiting $\texttt{SUS}_2$, which is at most $2n$ rounds. Failure of meeting either of a_1 or a_3 triggers the other agent to identify $\texttt{SUS}_1$ or $\texttt{SUS}_2$ to be the BBH, and accordingly perform perpetual exploration avoiding v_b.

On the other hand, consider the situation where a_2 gets destroyed along with the pebble either at round $r'+4n-2$ or at $r'+4n-1$. This situation can be understood by a_1 at $home$, when it finds the absence of a_2 at round $r'+4n$. Accordingly, a_1 detects v_{n-1} to be the BBH and performs perpetual exploration of R, avoiding v_{n-1} (i.e., v_b). Before explaining the second subcase, we have the following observation.

Observation 5.4.2. For $i \neq 0,1$, let $\hat{r}$ be the first round when a_1 and a_2 both reach v_i together. Let r' be the last round before $\hat{r}$ at which a_1 and a_2 start executing Rule-I, from $home$. Then $\hat{r} = r'+4i-3$.

For the second sub-case, i.e., $i \neq 0,1,2$, a_1 finds the absence of a_2 within round $r'+4i-2$, where r' is the last round before $r'+4i-2$ at which a_1 and a_2 start executing Rule-I. This triggers a_1 to move clockwise until it reaches $home$. Note that, a_1 must reach $home$ within round $r'+n+3i-3 \leq r'+4n-6$, since $i \leq n-1$ (since, from round $r'+4i-2$, a_1 moves clockwise for $n-i-1$ distance to reach $home$). At round $r'+n+3i-2$, a_1 moves counter-clockwise from $home$, and it continues to move until it reaches v_i, and waits for a_3. The agent a_3, on the other hand, finds that at round $r'+n+3i-1$, a_1 is absent at $home$, though it was present at the previous round. Note that $r'+n+3i-1 \leq r'+4n-4$, this triggers a_3 to move counter-clockwise until it finds a_1. The agent a_3 got triggered because, except for the case where a_1 detects an anomaly from a node v_j (where $j \neq 0,1,2$), in all other cases when a_1, after reaching $home$ at round $r'+4n-6$, must wait at $home$ for at least 3 more rounds, i.e., till round $r'+4n-3$. In Figure 5.3, an explicit time diagram of this case, starting from the round at which a_2 gets destroyed at v_b, until the round at which a_1 and a_3 gather at v_i

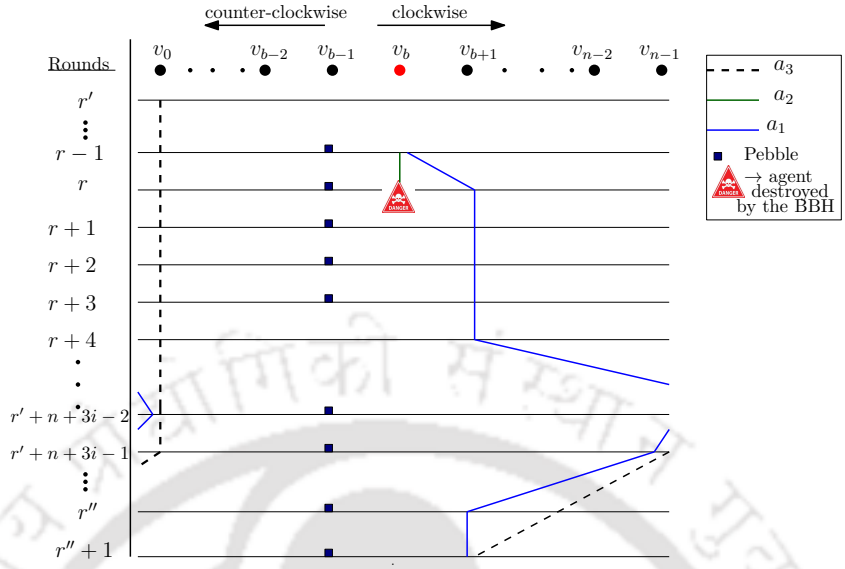


Figure 5.3: Represents the step-wise time diagram of PERPEXPLORE-COLOC-PBL where at round r, a_2 gets destroyed at v_b. a_1 detects the anomaly due to absence of a_2 at round $r+4$, and moves clockwise until v_0. It reaches v_0 at round $r'+n+3i-3$. From $r'+n+3i-2$, it starts moving counter clockwise until it reaches v_{b+1}. a_3 also after finding absence of a_1 at round $r'+n+3i-1$, starts moving counter-clockwise until it meets a_1, say at round $r''+1$.

(= v_{b+1} in this case), is explained.

After a_1 and a_3 meets, they recognize $\mathtt{SUS}_1$ as v_{i-1} and $\mathtt{SUS}_2$ as v_{i-2}, and operates similarly as described in the first sub-case. Accordingly, we can conclude by a similar argument that, at least one agent perpetually explores R, avoiding v_b.

Case-3: The first agents to be destroyed by the BBH are both a_1 and a_2. This case has been explicitly explained in the description of *Case-1*.

Case-4: Pebble is the first item to be destroyed by the BBH and not any agent. Let r be the round at which the pebble is destroyed, at a node v_b. So, within $(r+2)-$th round a_2 must visit v_b to collect the pebble as per Rule-II or Rule-III. At $(r+3)-$th round a_2 is already destroyed or a_2 is alive. If a_2 is alive, then it identifies the current node to be the BBH, due to the absence of the pebble and performs perpetual exploration, avoiding v_b. If a_2 gets destroyed, then it cannot return to meet a_1, and this is similar to the *Case-2*, where at least one agent among a_1 and a_3 perpetually explores R, avoiding the BBH.

From the above discussion, we have the following theorem.

Theorem 5.4.3. *A team of 3 co-located synchronous agents are sufficient to solve the prob-*

lem PERPEXPLORATION-BBH *on a ring R with n nodes under the pebble model of communication, in the presence of one pebble initially co-located with the agents.*

The necessary condition of the above theorem follows from Corollary 5.3.3.

5.4.2 Face-to-Face Model of Communication

In this section, we consider the face-to-face model of communication and prove the following theorem.

Theorem 5.4.4. *A team of 4 co-located and synchronous agents is sufficient to solve the problem* PERPEXPLORATION-BBH *on a ring R with n nodes under the face-to-face model of communication.*

In order to prove Theorem 5.4.4, we discuss an algorithm with 4 co-located agents in this model of communication. Initially, the 3 lowest ID agents, say, a_1, a_2 and a_3 are chosen, where the fourth lowest ID agent, say a_4, is associated with the second lowest ID agent, i.e., a_2. The idea of the algorithm resembles that of algorithm PERPEXPLORE-COLOC-PBL. In this case, as there is no existence of pebbles, the agent a_4 is used to mimic the pebble used in Section 5.4.1. In general, as per algorithm PERPEXPLORE-COLOC-PBL, when we say that an agent a_i carries a pebble p, in this case we mean that the agent a_i communicates a message *carry* with its associated agent $a_{i'}$ such that both these agents simultaneously move together until further new instruction is communicated. On the contrary as per algorithm PERPEXPLORE-COLOC-PBL, when we say that an agent a_i drops (or left) a pebble p at a node v, then in this case we mean that a_i communicates a message *drop* with $a_{i'}$, which in turn instructs $a_{i'}$ not to move further and remain stationary at the node v until further instruction is communicated. Hence, with this terminology, a team of 4 agents can execute the algorithm PERPEXPLORE-COLOC-PBL, and the correctness also follows similarly. So, this proves Theorem 5.4.4. In addition, the necessary statement of Theorem 5.4.4 is a consequence of Theorem 5.3.1, which follows from the fact that the face-to-face model can be simulated to the whiteboard model.

5.4.3 Whiteboard Model of Communication

The algorithm PERPEXPLORE-COLOC-PBL can be simulated in a whiteboard model of communication. A pebble on a node can be simulated by a bit of information, which is marked on the whiteboard of that node. Note that collecting a pebble from the node is simulated by erasing the same bit of information on that node and dropping the pebble can be simulated by marking the node with a bit of information as well on the whiteboard of that node. From this, we get the following result for the whiteboard model of communication.

Theorem 5.4.5. *A team of 3 synchronous co-located agents are sufficient to solve the problem* PERPEXPLORATION-BBH *on a ring R if each node is equipped with a whiteboard having constant memory.*

The sufficient condition follows from Corollary 5.3.2. Previously in [14], for an asynchronous scheduler, the tight bound for the number of agents to solve this problem was 4. Now from Theorem 5.4.5, we get a trade-off between reducing the optimal number of agents required vs. the scheduler, in the presence of a more generalized version of gray hole as well, i.e., the Byzantine black hole.

5.5 Perpetual Exploration with Scattered Agents

In this section, we discuss the problem of PERPEXPLORATION-BBH, when the agents are initially scattered on more than one node, where each starting node is assumed to be safe. We investigate this problem under different communication models. First, we discuss the pebble model of communication, and subsequently, we discuss the whiteboard model of communication.

5.5.1 Pebble Model of Communication

We discuss the problem when the agents communicate explicitly with the help of pebbles.

Algorithm Description

Here we discuss the model where the agents are placed arbitrarily along the nodes of the ring R (note that each such node must be a safe node), where each agent has a movable token (we call it a pebble), which it can carry along with it, and acts as a mode of inter-agent communication. Moreover, the agent can gather the IDs of other agents that are currently at the same node at the same round. With this context, we show that 4 scattered agents (namely, $A = \{a_0, a_1, a_2, a_3\}$) with a pebble each are sufficient to solve PERPEXPLORATION-BBH on R, using our algorithm PERPEXPLORE-SCAT-PBL. In this algorithm, we assumed that 4 agents are scattered at 4 different nodes of R. To accommodate the other cases, where 4 agents are scattered at less than or equal to 3 nodes, a slight modification of PERPEXPLORE-SCAT-PBL is required, which is explained in Remark 5.5.1. Next, we describe the high-level idea of the algorithm.

High level idea of Algorithm: Let the starting position of a_0 be the first, after which a_1, a_2, and a_3 have their respective starting positions in clockwise increasing order. Let h_i is denoted to be the starting position of a_i, for all $i \in \{0,1,2,3\}$ (they are indeed nodes of the form v_j for some $j \in \{0,1,\ldots,n-1\}$, but just to indicate them as *home* of a_i, we term it as h_i). By $Seg(a_i)$ (or *segment* of a_i) we mean the clockwise arc of nodes, starting from h_i and ending at $h_{(i+1) \pmod 4}$. If no agents are destroyed by the BBH node v_b, then the exploration of R is carried on iteratively by the agents as follows. Let an iteration start with round r. From round r onwards, agent a_i moves clockwise along $Seg(a_i)$ leaving its pebble at h_i until it reaches $h_{(i+1) \pmod 4}$, i.e., end of $Seg(a_i)$. An agent can distinguish the node $h_{(i+1) \pmod 4}$ by seeing the pebble left there by the agent $a_{(i+1) \pmod 4}$. When a_i reaches $h_{(i+1) \pmod 4}$ traversing $Seg(a_i)$ for the first time, it knows the length of $Seg(a_i)$ and stores the length in its own memory. For further traversals, it does not depend on seeing pebbles at the end nodes of the segment, as it can use the length of that segment. Irrespective of which, after a_i reaches $h_{(i+1) \pmod 4}$, it waits there till round $r + n - 1$. This waiting time ensures that the other agents have reached the endpoints of their corresponding segments during their clockwise movement. At round $r + n$, a_i collects the pebble (if it exists) from $h_{(i+1) \pmod 4}$ (the pebble left by $a_{(i+1) \pmod 4}$) and starts moving counter-clockwise along

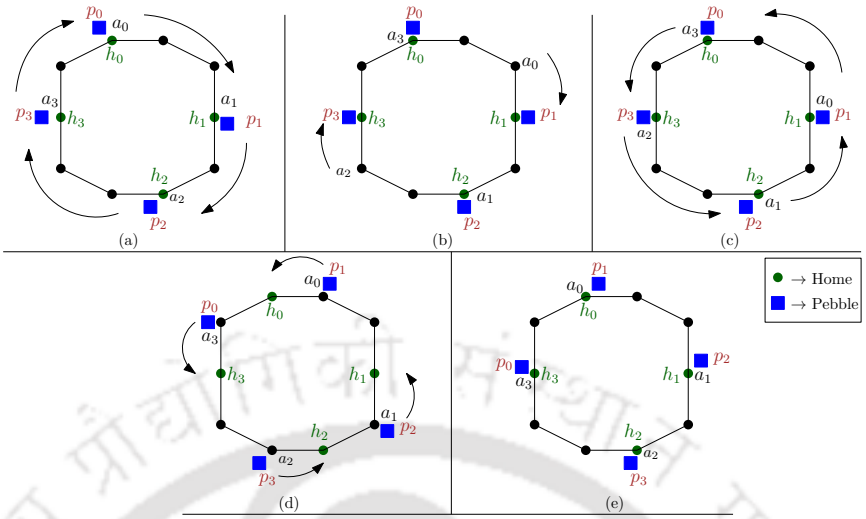


Figure 5.4: **(a)** h_i's are *home*. Agent a_i is on h_i initially with a pebble. We name the pebble initially at h_i as p_i, but in reality, they are anonymous. **(a-b)** Each agent a_i moves clockwise until the next *home* (i.e., $h_{(i+1) \pmod 4}$) without carrying any pebble. The agents already reached (here a_2 and a_4) wait for others. **(c-e)** All agents are on their clockwise nearest *home*. They start moving counter-clockwise together with the pebble present at their current location towards their initial *home*. The agents that have already reached their *home* wait for others to reach their *home*.

with the collected pebble until it reaches its own *home*, i.e., h_i. Also note that, if a_i does not find any pebble to collect at $h_{(i+1) \pmod 4}$, it starts moving counter-clockwise without any pebble. This case can only happen if $a_{(i+1) \pmod 4}$ is destroyed at the BBH node v_b, while it was returning back after collecting the pebble from $h_{(i+2) \pmod 4}$ in the previous iteration. After a_i reaches h_i during its counter-clockwise movement, it again waits till round $r+3n$ before it repeats the whole process in the next iteration. This waiting time is devised in such a way that if an agent during this iteration detects any anomaly due to intervention by the BBH, it has enough time to gather with the other agents. Note that, since $\cup_{i=0}^{3} Seg(a_i) = R$, if no agents are destroyed, in each iteration the agents collectively visit each node of R. Fig. 5.4 illustrates the execution of an iteration.

Intervention by the BBH: The exploration can be hampered only if an agent is destroyed at the BBH node v_b. Since $Seg(a_i) \cap Seg(a_j)$ is either empty or consists of a safe node, and in addition, a segment is explored by only one agent, so at most one agent can be destroyed by the BBH during an iteration. Without loss of generality, let $v_b \in Seg(a_j)$, for some $j \in \{0,1,2,3\}$. Let us assume a_j is the first agent to be destroyed at v_b at the i-th

iteration. Based on this, there are two cases.

Case-I: a_j is destroyed while moving clockwise in iteration i. Let r_i be the round at which i-th iteration started. For this case, a_j fails to collect the pebble from $h_{(j+1) \pmod 4}$ left by $a_{(j+1) \pmod 4}$. So, when $a_{(j+1) \pmod 4}$, returns to $h_{(j+1) \pmod 4}$ after moving counter-clockwise along with the pebble it has collected from $h_{(j+2) \pmod 4}$, it finds two pebble at $h_{(j+1) \pmod 4}$ within $r_i + 2n - 1$ round. This is considered by $a_{(j+1) \pmod 4}$ as an anomaly, and it learns that v_b must be in the segment, in counter-clockwise direction starting from $h_{(j+1) \pmod 4}$. In this scenario, $a_{(j+1) \pmod 4}$ waits till round $r_i + 2n - 1$, so that other agents return to their corresponding *home* in the meantime. Next, from round $r_i + 2n$, the agent $a_{(j+1) \pmod 4}$ starts moving clockwise with two pebbles. The aim of this move is to meet and gather with the other alive agents. Note that the other alive agents wait at their corresponding *home* till round $r_i + 3n$, from round $r_i + 2n - 1$ onwards. It is because, at their respective *home* they do not detect any anomaly. Since $a_{(j+1) \pmod 4}$ starts moving clockwise with the pebbles from round $r_i + 2n$, it can cover every node within the round $r_i + 3n - 1$. The agent $a_{(j+1) \pmod 4}$ first meets $a_{(j+2) \pmod 4}$ at $h_{(j+2) \pmod 4}$ while it moves clockwise after detecting anomaly. Then both of them move together, while $a_{(j+2) \pmod 4}$ carries the pebble, which it was earlier carrying back to *home*. They move until they meet with $a_{(j+3) \pmod 4}$ at $h_{(j+3) \pmod 4}$. Note that at this moment 3 agents are at a node (which is $h_{(j+3) \pmod 4}$), with 4 pebbles (one carried by $a_{(j+3) \pmod 4}$, two carried by $a_{(j+2) \pmod 4}$ and one with $a_{(j+3) \pmod 4}$). In this case, they execute the algorithm PERPEXPLORE-COLOC-PBL and achieve PERPEXPLORATION-BBH of the ring R. Figure 5.5 represents the time diagram indicating the movement of the agent starting from r_i, then destruction of a_j within $r_i + n$ and finally all 3 alive agents with 4 pebbles, gathering at round r'_i ($\leq r_i + 3n$).

Case-II: a_j is destroyed while moving counter-clockwise in iteration i. Let r_i be the round at which i-th iteration has started. Since a_j is destroyed while moving counter-clockwise, it implies that it got destroyed after round $r_i + n - 1$, and it was carrying the pebble it collected from $h_{(j+1) \pmod 4}$. Now, when all alive agents return to their corresponding *home* during iteration i, i.e., within round $r_i + 2n - 1$, none of them finds any anomaly. Accordingly, each agent waits till round $r_i + 3n$, and starts $(i+1)$-th iteration from $r_i + 3n + 1$ round, which we term as r_{i+1}. In this iteration, all agents except a_j (since it is destroyed

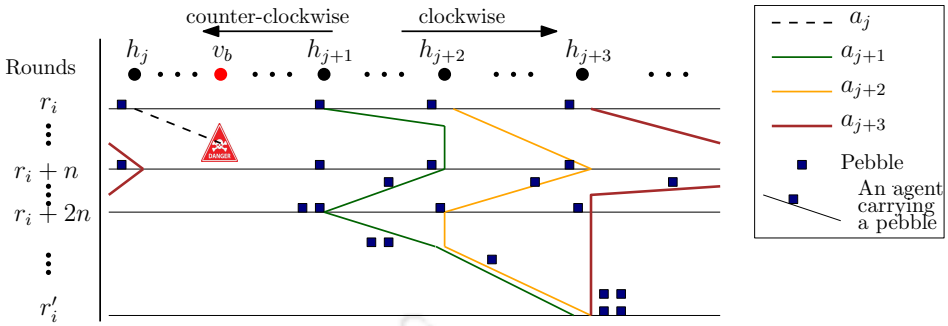


Figure 5.5: Represents the i-th iteration time diagram of PERPEXPLORE-SCAT-PBL where at round r_i, a_j gets destroyed at v_b in $Seg(a_j)$, while moving clockwise. a_{j+1} detects anomaly and starts moving with 2 pebbles from $r_i + 2n$ onwards. Meets with a_{j+2} and a_{j+3}, each accompanying a pebble, gathers at round r'_i ($\leq r_i + 3n$).

at the i-th iteration) move clockwise to the end points of their corresponding segments, after leaving the pebble at their corresponding *home*. Note that at this point within round $r_{i+1} + n - 1$, each alive agent except $a_{(j+3) \pmod 4}$ finds a pebble at their clockwise *home* of their respective segment. According to our algorithm, $a_{(j+3) \pmod 4}$ does not consider the absence of a pebble at its clockwise adjacent *home*, as an anomaly. As per our algorithm, these agents wait there till $r_{i+1} + n - 1$ round and collects pebble (for $a_{(j+3) \pmod 4}$ no pebble exists) and moves back to their corresponding *home* ($a_{(j+3) \pmod 4}$ moves back without the pebble) again within $r_{i+1} + 2n - 1$ round. Now, since a_j was destroyed earlier, the pebble left by $a_{(j+1) \pmod 4}$ was picked by no agents and thus, while $a_{(j+1) \pmod 4}$ returns back to $h_{(j+1) \pmod 4}$ it finds two pebbles and does the same execution as explained in *Case-I*. But in this case, the pebble that a_j was carrying may not be destroyed at v_b (as the adversary has the choice to destroy it). So, after gathering, when the remaining alive agents from $h_{(j+3) \pmod 4}$ start executing PERPEXPLORE-COLOC-PBL, the leading agent (or the lowest ID agent among them), detects the BBH to be its current node, whenever it finds a pebble at a node other than $h_{(j+3) \pmod 4}$. Accordingly, after this, it starts exploring R perpetually, avoiding v_b.

The basic idea of this algorithm is to gather three agents with at least 3 pebbles in the expense of one agent and then execute PERPEXPLORE-COLOC-PBL (ref. Remark 5.5.2) to solve the PERPEXPLROATION-BBH problem. Figure 5.6 demonstrates via a time diagram, starting from the i-th iteration at which a_j and also the pebble gets destroyed while moving

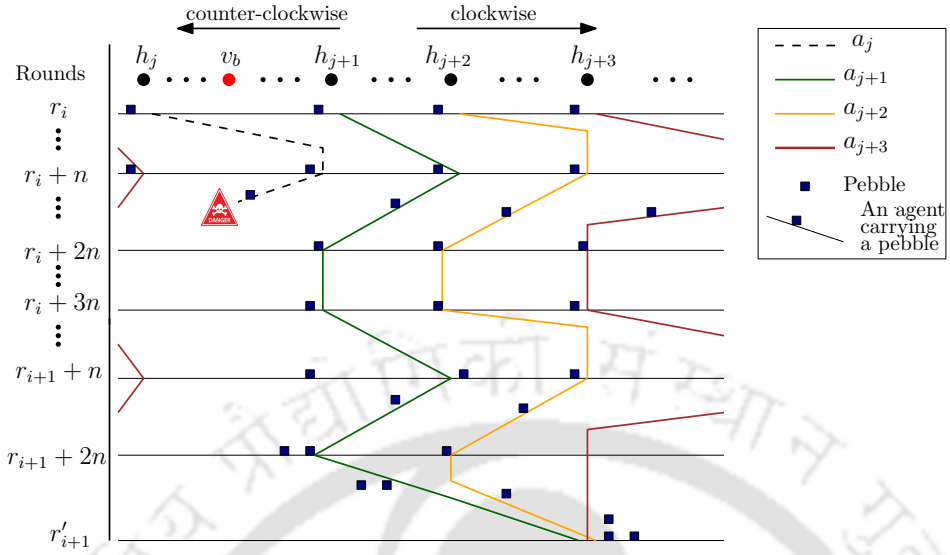


Figure 5.6: Represents the i-th iteration time diagram of PERPEXPLORE-SCAT-PBL, where the agent a_j gets destroyed at v_b with the pebble, while moving counter-clockwise. a_{j+1} detects the anomaly within $r_{i+1}+2n-1$ due to the presence of 2 pebbles at h_{j+1}. Starts moving clockwise and gathers with remaining alive agents within round r'_{i+1} ($< r_{i+1}+3n$) at h_{j+3}.

counter-clockwise, uptill the round at which the remaining 3 alive agents gather with the 3 remaining pebbles at $h_{(j+3) \pmod 4}$.

In the following remark (ref. Remark 5.5.1), we discuss the modification of algorithm PERPEXPLORE-SCAT-PBL so that it includes the initial configurations where four agents are distributed among less than four nodes.

Remark 5.5.1. In the discussion of our algorithm PERPEXPLORE-SCAT-PBL, we have considered that 4 agents are scattered along 4 distinct nodes. Here we describe how our algorithm PERPEXPLORE-SCAT-PBL works for the remaining cases (i.e., when 4 agents are scattered among 3 or 2 nodes initially) by slight modification. Observe that in these remaining cases, there exists at least one node where initially the multiplicity is greater than 1. Let there exist a node with multiplicity 3; in this case, these agents directly start executing PERPEXPLORE-COLOC-PBL. While executing the algorithm, if they encounter the fourth agent (along with the pebble) somewhere along R, they just ignore this agent and the pebble (note that the IDs of all three initially co-located agents are already collected by each other) while executing their current algorithm. On the other hand, if initially there exists a node with

multiplicity greater than 1 and less than 3, then in that case, only the lowest ID agent (say a_1) at the current node performs the iteration, whereas the other agent at the current node (say a_2) waits till $r_i + 2n - 1$ round (for some i-th iteration). The agent a_2 then checks for an anomaly in the next round (i.e., if the current node has more pebbles than the current number of agents). If such an anomaly exists after $r_i + 2n - 1$ rounds, then that implies that a_1 has already detected the anomaly. It is because, a_1, while it returned back to its $home$ within $r_i + 2n - 1$ rounds, carrying a pebble which it collected from the nearest clockwise $home$, it encounters the first anomaly, i.e., the number of pebbles is more than the number of agents. In that case, a_1 waits till $r_i + 2n - 1$ round and after which it starts to move clockwise, and on the other hand, a_2 also starts moving clockwise. This guarantees that if the anomaly is detected at a multiplicity, then both a_1 and a_2 move together to gather with the third agent, which is waiting till round $r_i + 3n$. Now, if the anomaly is not detected at the multiplicity, then both a_1 and a_2 wait till $r_i + 3n$ round at the same node. After which a_1 further starts the $(i+1)$-th iteration from round $r_i + 3n + 1$ $(= r_{i+1})$ onwards and a_2 waits till $r_{i+1} + 2n - 1$. If some other agent detects an anomaly at the i-th iteration, it must meet a_1 and a_2 at their $home$ at some round r'_i $(< r_i + 3n)$. In that case, they find that there are 3 agents at their $home$ and at least 3 pebbles, and all of them start executing the algorithm PERPEXPLORE-COLOC-PBL.

Remark 5.5.2. As mentioned in PERPEXPLORE-COLOC-PBL requires, 3 agents and 1 pebble to perform PERPEXPLORATION-BBH, but in this case, we instruct the agents to perform PERPEXPLORE-COLOC-PBL whenever three agents and at least three pebbles are able to gather at a node. These agents perform PERPEXPLORE-COLOC-PBL with a slight modification that excess pebbles remain at their gathered node, which they term as $home$.

The following theorem follows from the above description of the algorithm PERPEXPLORE-SCAT-PBL and ensures the correctness of the algorithm.

Theorem 5.5.3. *Algorithm* PERPEXPLORE-SCAT-PBL *solves the* PERPEXPLORATION-BBH *problem on a ring R with 4 synchronous and scattered agents under the pebble model of communication, where each agents are equipped with a pebble.*

The necessary condition of the above theorem follows from Corollary 5.3.5.

5.5.2 Whiteboard Model of Communication

In this section, we propose an algorithm PERPEXPLORE-SCAT-WHITEBOARD that solves the problem PERPEXPLORATION-BBH with 3 agents, where the agents are initially scattered, and they communicate via the whiteboard present at each node of R. This algorithm is designed for the case when all the agents are starting at different nodes. Note that this algorithm can be easily modified a bit to include the case where initially three agents are scattered at two distinct nodes.

High Level Idea of Algorithm: Let a_0, a_1 and a_2 be three agents starting from the nodes h_0, h_1 and h_2, respectively, where these nodes are in a clockwise order. Let $Seg(a_i)$ (also called 'Segment of a_i') be the clockwise arc starting from h_i and ending at $h_{(i+1) \pmod 3}$. If no agents are destroyed by the BBH node v_b, then the exploration of R is carried out iteratively by the agents in the following manner. Note that, at the start of the iteration, each agent is at their corresponding $home$. Let an iteration start at round r. At round r, an agent, say a_i, first erases all previously stored data (if any) at its own $home$, i.e., h_i. Subsequently, in this round itself, it writes the message $(home, ID(a_i))$ at h_i and starts moving clockwise. This type of message is called a *home* type message. It indicates that h_i is the $home$ of a_i. The agent a_i continues to move clockwise, until it reaches the node $h_{(i+1) \pmod 3}$, where this node also signifies the end of $Seg(a_i)$, as $h_{(i+1) \pmod 3}$ is the $home$ of $a_{(i+1) \pmod 3}$. Moreover, when a_i moves clockwise along $Seg(a_i)$, it marks each node of $Seg(a_i)$ by writing *right*, while erasing any previous such markings (if at all exists), excluding h_i and $h_{(i+1) \pmod 3}$. On reaching the node $h_{(i+1) \pmod 3}$, the agent a_i writes the message $(visited, ID(a_i))$ at $h_{(i+1) \pmod 3}$ after removing the earlier *home* type message left by $a_{(i+1) \pmod 3}$. This type of message is termed as *visited* type message. It also indicates that an agent has visited the last node (in clockwise direction) of its own segment. Moreover, a_i also waits until $(r+2n-1)$ round at the node $h_{(i+1) \pmod 3}$. This waiting time is designed to ensure that, if any agent encounters any information regarding the segment containing the BBH, it can get enough time to meet all the other alive agents that are waiting till round $r+2n-1$. Starting from round $r+2n$, a_i starts to move in a counter-clockwise direction from $h_{(i+1) \pmod 3}$. While moving counter-clockwise, a_i erases previously written *right*

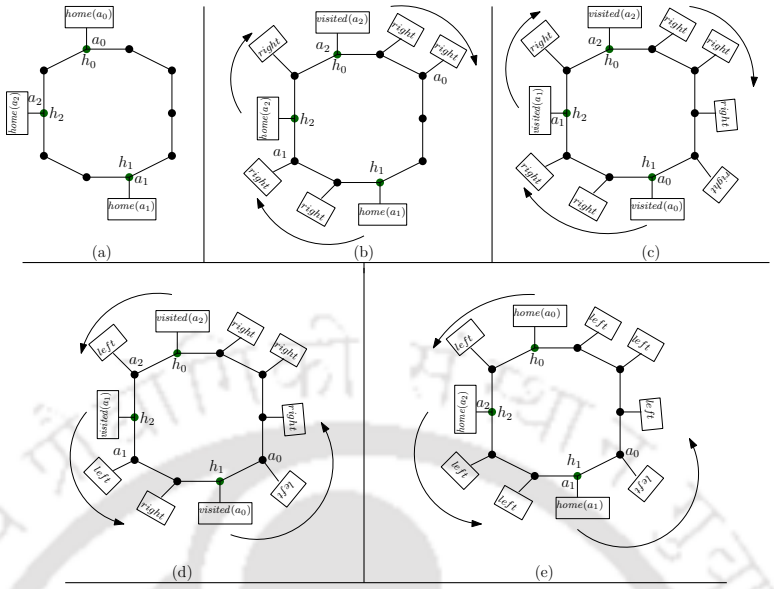


Figure 5.7: **(a)** Agent a_i is on h_i initially, where it writes a *home* type message. **(b-c)** Each agent a_i moves clockwise while writing a *right* message at each node, except at h_i and $h_{(i+1) \pmod 3}$, and stops at $h_{(i+1) \pmod 3}$. At $h_{(i+1) \pmod 3}$, a_i writes *visited* type message. **(d-e)** Each agent a_i starts returning to h_i by moving counter-clockwise by erasing *right* message and writing *left* message. At h_i, a_i erase *visited* message and write *home* message.

marking (which it marked while moving along the clockwise direction) from each node of $Seg(a_i)$ and writes *left* there upon arriving, except at the nodes h_i and $h_{(i+1) \pmod 3}$. This counter-clockwise movement continues until a_i reaches its own $home$, i.e., h_i. It finds at h_i a *visited* type message left there by $a_{(i-1) \pmod 3}$ in the current iteration. On reaching h_i, a_i again waits till round $r + 4n$. This waiting time is again designed to ensure that, if an agent during its iteration detects any anomaly due to the intervention by the BBH, it has enough time to meet and gather with other alive agents, who are waiting till round $r + 4n$. Note that since $\cup_{i=0}^{2} Seg(a_i) = R$, if no agents are destroyed by the BBH, in each iteration the agents collectively visit each node of R. The execution of an iteration is explained in Fig. 5.7.

Intervention by the BBH: The exploration can only be hampered if an agent is destroyed by the BBH at v_b. Since $Seg(a_i) \cap Seg(a_j)$ is either empty or consists of a safe node, and also, since a segment is explored by only one agent, this implies that at most one agent can be destroyed by the BBH during an iteration. Without loss of generality, let $v_b \in Seg(a_j)$, for some $j \in \{0,1,2\}$. Let us assume that a_j be the first agent to be destroyed at v_b at the

i-th iteration. Now, there are two cases.

Case-I: a_j is destroyed while moving clockwise in the i-th iteration. Let r_i be the round at which i-th iteration has started. This means, a_j has failed to reach $h_{(j+1) \pmod 3}$ within round r_i+n-1, and also has failed to write *visited* type message there. So, when $a_{(j+1) \pmod 3}$ returns to its *home* (i.e., $h_{(j+1) \pmod 3}$) within round r_i+3n-1, it finds no *visited* type message, as it should have, if a_j was not destroyed. This helps $a_{(j+1) \pmod 3}$ to interpret that the adjacent counter-clockwise segment of its own segment has the BBH, and an agent must have been destroyed there while moving clockwise. This information triggers it to move clockwise with the aim of gathering with other alive agents, i.e., only $a_{(j+2) \pmod 3}$ in this case, from round r_i+3n. If $a_{(j+1) \pmod 3}$ reached $h_{(j+1) \pmod 3}$ before r_i+3n-1, and detects the above-mentioned anomaly, it waits till round r_i+3n-1 and then starts the move from round r_i+3n. This waiting time ensures that while it starts moving with the aim of gathering with other agents, all of them are waiting at their corresponding *home*. Note that, when $a_{(j+1) \pmod 3}$ starts moving, $a_{(j+2) \pmod 3}$ is waiting until round r_i+4n. This waiting time is sufficient for $a_{(j+1) \pmod 3}$ to meet with $a_{(j+2) \pmod 3}$. After they meet, $a_{(j+1) \pmod 3}$ shares the information about its direction of movement in the whiteboard of $h_{(j+2) \pmod 3}$. With this, they again start moving clockwise together until they reach h_j. The agents can distinguish h_j by the *visited* type message written there by $a_{(j+2) \pmod 3}$ within round r_i+2n-1. Note that, in the clockwise direction, each node after h_j up to the previous node of v_b was marked *right* by a_j before it was destroyed. So, from h_j, the agents $a_{(j+1) \pmod 3}$ and $a_{(j+2) \pmod 3}$, identify themselves as a_L and a_H, respectively, and follow Rule-I. Figure 5.8 demonstrates the step-wise time diagram of i-th iteration, at which a_j gets destroyed, and subsequently remaining alive agents gather at h_j at round r'_i ($\leq r_i+4n-1$).

Rule-I: In this case, suppose they are at a node v_0, from this node the two agents a_L and a_H execute the following execution. a_L moves clockwise to the next node, say v_1 while the other agent waits at v_0. If at v_1, a_L sees the *right* marking, it interprets v_1 as safe. So, it comes back to v_0. At v_0, seeing that a_L has returned, a_H also interprets v_1 to be safe. So, in the next round, both a_L and a_H move to v_1 together and repeat the process from v_1 again. When a_L reaches v_b, it sees no *right* marking there. a_L if alive, interprets this by determining the current node to be the BBH. So, it starts perpetual exploration of R, except

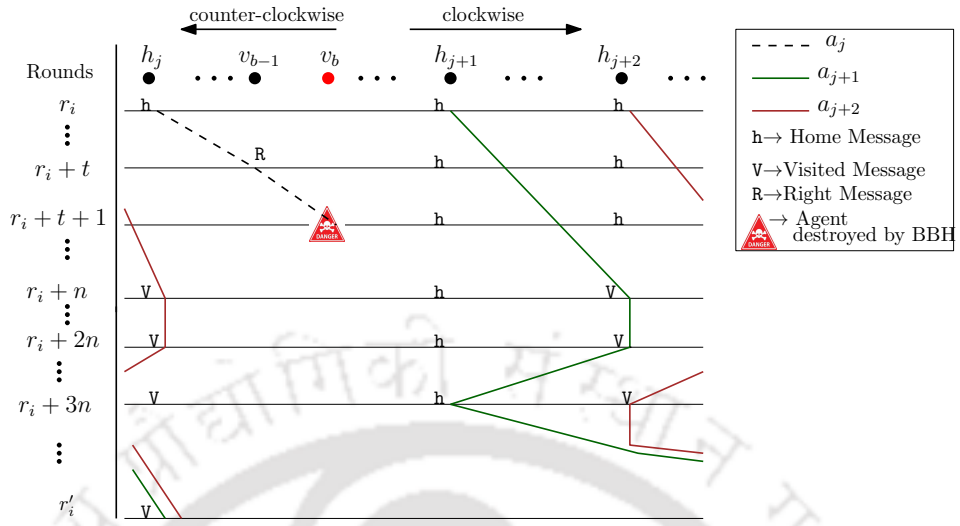


Figure 5.8: Represents the i-th iteration time diagram of PERPEXPLORE-SCAT-WHITEBOARD. The agent a_j gets destroyed at v_b at round $r_i + t + 1$ ($< r_i + n$) while moving clockwise. a_{j+1} detects the anomaly after seeing the *home* type message at h_{j+1} within round $r_i + 3n$ and starts moving clockwise. Within round r'_i ($\leq r_i + 4n - 1$), a_{j+1} meets with a_{j+2} at h_{j+2} and gathers at h_j at round r'_i.

v_b. Otherwise, if a_L gets destroyed at v_b, it does not return to the previous node where a_H is waiting. Even after waiting, when a_H sees that a_L has not returned, it interprets this incident as the next clockwise node is the BBH and starts exploring the ring R, avoiding that node. Thus, if two agents follow this rule, then perpetual exploration of R except v_b is guaranteed.

Case-II: a_j is destroyed while moving counter-clockwise in the i-th iteration. In this case, both the alive agents $a_{(j+1) \pmod 3}$ and $a_{(j+2) \pmod 3}$ find *visited* type message after returning to their corresponding *home* within round $r_i + 3n - 1$. This is because a_j started performing a counter-clockwise movement from round $r_i + 2n$. So, it wrote *visited* type message at $h_{(j+1) \pmod 3}$, and after which during counter-clockwise movement it got destroyed at v_b, i.e., between $r_i + 2n$ and $r_i + 3n - 1$ rounds. So, all alive agents after waiting at their *home* till $r_i + 4n$ round, starts $(i+1)$-th iteration from $r_i + 4n + 1$ ($= r_{i+1}$) round onwards. Again in the $(i+1)$-th iteration, each alive agent starting from their own *home* first erases the existing whiteboard content and writes the corresponding *home* type message before it starts moving clockwise. This clockwise movement will end when the agents reach the end of their segment. Note that since a_j was destroyed in the i-th iteration, no agent erased the

whiteboard at h_j in the first round of iteration $(i+1)$. So the *visited* type message, which was left by $a_{j-1 \pmod 3}$ during iteration i, still remains there at iteration $(i+1)$. So, upon reaching h_j during iteration $(i+1)$, $a_{(j-1) \pmod 3}$ finds the *visited* type message which was written there earlier by it within round $r_{i+1}+n-1$. From this information, $a_{(j-1) \pmod 3}$ interprets that v_b must be in the segment adjacent to its own segment in the clockwise direction. It also interprets that an agent must have been destroyed at v_b while it was moving counter-clockwise in the previous iteration. So, accordingly $a_{(j-1) \pmod 3}$ waits at h_j till round $r_{i+1}+n-1$, and starts moving counter-clockwise from $r_{i+1}+n$ round onwards. This waiting time ensures that while $a_{(j-1) \pmod 3}$ starts moving with the aim of gathering with other alive agents, all of them are at the last node of their corresponding segment, waiting. Now the other agents wait till round $r_{i+1}+2n-1$ before moving counter-clockwise back to their corresponding $home$. So, it is enough for $a_{(j-1) \pmod 3}$ to meet with these other alive agents between rond $r_{i+1}+n$ to round $r_{i+1}+2n-1$. During this counter-clockwise movement of $a_{(j-1) \pmod 3}$, it meets $a_{(j-2) \pmod 3}$ (the only alive agent) at $h_{(j-1) \pmod 3}$. After they meet, $a_{(j-1) \pmod 3}$ communicates the direction of its move to $a_{(j-2) \pmod 3}$ on the whiteboard of $h_{(j-1) \pmod 3}$ and they move together until they reach $h_{(j-2) \pmod 3} (= h_{(j+1) \pmod 3})$ together. They distinguish the node by the *home* type message left there by $a_{(j-2) \pmod 3}$ before starting to move clockwise, from r_{i+1} round onwards. Note that, in the counter-clockwise direction, the nodes in $Seg(a_j)$, starting from the next node of $h_{(j+1) \pmod 3}$ up to the previous node of v_b are the only nodes that were marked *left* by a_j before it was destroyed. So, from $h_{(j+1) \pmod 3}$, the agents $a_{(j-1) \pmod 3}$ and $a_{(j-2) \pmod 3}$ identify themselves as a_L and a_H, respectively, and follow Rule-II. Figure 5.9 demonstrates the step-wise time diagram of the i-th iteration at which a_j gets destroyed, and the $(i+1)$-th iteration, at which the remaining alive agents gather at $h_{j+1 \pmod 3}$.

Rule-II: This rule for a_L and a_H are similar to Rule-I, except for the fact that here, after moving one node in the counter-clockwise direction, the agent a_L searches for the *left* marking. If it finds such a marking it returns to a_H and then both move counter-clockwise and repeat the process. On the other hand, if a_L does not find any *left* marking (it must be v_b) and it stays alive, a_L moves out of that node and starts exploring R, avoiding that node. On the contrary, if a_L gets destroyed, then a_H sees that a_L has not returned while it should

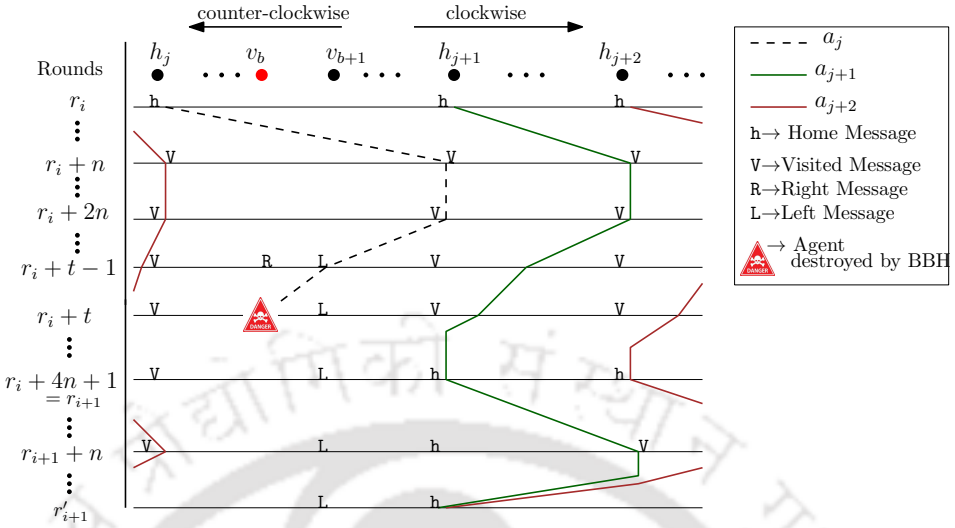


Figure 5.9: Represents the i-th iteration time diagram of PERPEXPLORE-SCAT-WHITEBOARD. The agent a_j gets destroyed at v_b at round $r_i + t + 1$ ($< r_i + 3n - 1$) while moving counter-clockwise. a_{j+2} detects the anomaly at $(i+1)$-th iteration, within round $r_{i+1} + n - 1$, and starts moving counter-clockwise from round $r_{i+1} + n$. At round r'_{i+1} ($\leq r'_{i+1} + 2n$), it gathers with a_{j+1} at h_{j+1}.

have. From this incident, it is interpreted that the next counter-clockwise node is v_b, and it starts exploring R, avoiding that node. Thus again, if two agents follow this rule, then perpetual exploration except v_b of R is guaranteed.

The next remark discusses the modification required to include the cases where the initial starting nodes can have multiplicity.

Remark 5.5.4. Let a_i, a_j, and a_k be three agents that start from two initial nodes, say $home_1$ and $home_2$. By Pigeon hole principle, exactly one of $home_1$ and $home_2$ initially must have two agents. Without loss of generality, let $home_1$ have two agents, say a_i and a_k, initially. In this case, the agents having multiplicity greater than one at the current node do not move. On the other hand, the singleton agent, i.e., a_j, writes a *home* type message at $home_2$, and starts moving clockwise, while marking each node with message *right*. If a_j reaches $home_1$ before being destroyed by the BBH, then $home_1$ now has three agents co-located. Thus, from here the agents execute the whiteboard version of PERPEXPLORE-COLOC-PBL (refer Subsection 5.4.3). On the other hand, let us consider the case when a_j gets destroyed before reaching $home_1$. Note that irrespective of the location of $home_2$, it takes at most $n-1$ rounds for a_j to reach $home_1$ from the beginning. So a_i and a_j wait for n rounds

and find that no one has arrived yet. In this case, both a_i and a_k, together move to $home_2$ from the clockwise direction. They identify $home_2$ by the *home* type message left by a_j. From $home_2$, a_i becomes a_L and a_k becomes a_R, and they execute Rule-I (described in *Case-I*). Rule-I ensures that BBH is detected and perpetual exploration of R, except v_b is guaranteed.

From the above description, we have the following theorem that ensures the correctness of the algorithm PERPEXPLORE-SCAT-WHITEBOARD.

Theorem 5.5.5. *Algorithm* PERPEXPLORE-SCAT-WHITEBOARD *solves* PERPEXPLORATION-BBH *on a ring R with n nodes and with 3 synchronous agents initially scattered under the whiteboard model of communication*

The necessary condition of the above theorem follows from Corollary 5.3.2.

5.6 Conclusion

This chapter addresses perpetual exploration of a ring network in the presence of a *Byzantine black hole*. We term it as PERPEXPLORATION-BBH, and it is explored under three communication models (*face-to-face, pebble,* and *whiteboard*) considering two initial configurations, one is *co-located* and another is *scattered*. The aim has been to minimize the number of agents required to perform PERPEXPLORATION-BBH. To the best of our knowledge, the concept of BBH has not been introduced before this work. We proposed optimal results (in terms of number of agents) for *Pebble* and *Whiteboard* communication models under both initial configurations. Further, an upper bound of 4 agents and a lower bound of 3 agents is provided for the *face-to-face* model, in case of co-located agents.

Future research could focus on proposing an optimal bound for the *face-to-face* model when the agents are initially co-located. Another direction could be proposing constructive lower and upper bounds for the *face-to-face* model when the agents are initially scattered. Additionally, investigating this problem in different scheduler models can be another future direction.

Chapter 6

Perpetual Exploration of Arbitrary Graphs with a Byzantine Black Hole

6.1 Introduction

In Chapter 5, we studied the perpetual exploration problem of a static ring in the presence of a *Byzantine black hole* (BBH). The study focused on different initial configurations of the agents and various communication models. Now, a genuine question arises: what if the underlying topology is arbitrary? This question intrigued us. The reason being, nothing was much known about the difficulty of solving perpetual exploration in a topology, except for a ring, in the presence of BBH.

Observe that, in Chapter 5, our aim is to explore all the safe nodes infinitely often, and since the ring is 2-connected, we faced no issue. But, what if the agents are initially co-located and the BBH is a cut-vertex of the network, then it becomes impossible to visit each safe node in the network, as the BBH may block access by behaving as a black hole. Hence, we tweak our earlier definition of PERPEXPLORATION-BBH (in Chapter 5) into two parts: here, our focus is to perpetually explore *one connected component* of the graph, the connected components would be obtained if the BBH and all its adjacent edges are re-

[6]This chapter has been published as: "Perpetual exploration in anonymous synchronous networks with a Byzantine black hole" in *Proceedings of the 39th International Symposium on Distributed Computing (DISC 2025)*.

moved from the network. In a 2-connected network, this is exactly the same as what we did in Chapter 5. Similarly, here we also refer to it as PERPEXPLORATION-BBH. Next, we studied another variation of the problem, where the agents are required to perpetually explore specifically the connected component that includes their initial position, i.e., their *home*.

Notation	Meaning
$G = (V, E, \lambda)$	Indicates the connected port-labeled graph
v_b	A node belonging to V that indicates the BBH
$home$ or h	The starting node
k	Number of agents that are initially co-located at $home$
$\mathcal{C}_i$	Indicates the i-th component of $G - \{v_b\}$
$\mathcal{E}$	Indicates an execution
T_d	Indicates the destruction time
$I =< G, k, h, v_b >$	Indicates an instance
Δ, n	Maximum degree and number of nodes in the graph G
$Cons(\mathcal{E}, I, t)$	Indicates consistent instances
$S(t)$	Indicates the suspicious nodes at time t
$S_i(t)$	Denotes the suspicious node set of agent with ID i at time t
$\texttt{pos}_a$	Indicates the position of the agent a at a certain time
$P = (V, E, \lambda)$	Indicates a path graph
$L[v_i, v_j]$	Indicates the induced subgraph of P by the nodes $\{v_x : 0 \leq i \leq x \leq j \leq \lvert V \rvert - 1\}$
$L, \texttt{Int}_1, \texttt{Int}_2, F$	Indicates the agents denoted by Leader, Intermediate 1, Intermediate 2 and Follower
F_1, F_2	Indicates the agents denoted by Follower 1 and Follower 2
$\texttt{Anchor}(\alpha)$	Indicates an agent acting as an *anchor* at the current node with respect to port α
SG	Indicates the agents in smaller group
$\texttt{LG}_i$	Indicates the agents in larger group, where $\texttt{LG}_{i+1} \subset \texttt{LG}_i$
E^i_j	Members of $\texttt{LG}_i$ acting as explorers, where $1 \leq j \leq 3$
$N(v)$	Neighbor nodes of v

Table 6.1: List of prominent notations used in this chapter

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.

While investigating these two variations of perpetual exploration, we found many interesting answers, and this chapter covers them. We also observed that there are many challenging directions to explore in the future, some of which are discussed in the conclusion of this chapter.

A list of some of the prominent notations used in this chapter is presented in Table 6.1. Another table of notations specifically used in Section 6.5.1 is presented in Table 6.3.

6.1.1 Our Contribution

We study the PERPEXPLORATION-BBH and PERPEXPLORATION-BBH-HOME problems in synchronous connected networks, where the underlying topology is path, tree and arbitrary or general graph, with at most one BBH under the *face-to-face* communication model. Our objective is to minimize the number of agents required to solve these problems.

Tree: 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 knowledge of the size of the network, i.e., n. However, knowledge of n would not reduce the number of agents, as our lower bounds do not assume that n is unknown.

Path: 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.

Arbitrary Graph: In the case of arbitrary networks, 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 [61]. 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 prove a stronger, and more technical, lower bound of $2\Delta - 1$ agents in presence of a BBH. 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-HOME. Further, Table 6.2 summarises the list of results obtained in this chapter.

Graph Type	**Problem**	**Lower Bound**	**Upper Bound**
Path	PerpExploration-BBH	4 agents	4 agents
	PerpExploration-BBH-Home	6 agents	6 agents
Tree	PerpExploration-BBH	4 agents	4 agents
	PerpExploration-BBH-Home	6 agents	6 agents
General Graph	PerpExploration-BBH	$2\Delta - 1$ agents	$3\Delta + 3$ agents
	PerpExploration-BBH-Home	$2\Delta - 1$ agents	$3\Delta + 3$ agents
	PerpExploration-BH	$\Delta + 1$ agents	$\Delta + 2$ agents

Table 6.2: Summary of results obtained in this chapter.

6.2 Model and Basic Definitions

The agents operate in a simple, undirected, connected port-labeled graph $G = (V, E, \lambda)$, where $\lambda = (\lambda_v)_{v \in V}$ is a collection of port-labeling functions $\lambda_v : E_v \to \{1, \ldots, deg(v)\}$, where E_v is the set of edges incident to node v and $deg(v)$ 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 $v_b \in V$ is a BBH. 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 v_b, as well as all agents that choose to move to v_b in that round. The agents have no information on the position of the BBH, except that it is not located at the *home* node. Furthermore, the agents do not have initial knowledge of the size of the graph.

6.2.1 Problem Definition

We define the PERPETUAL EXPLORATION problem with initially co-located agents in the presence of a BBH, hereafter denoted PERPEXPLORATION-BBH, as the problem of perpetually exploring at least one of the connected components resulting from the removal of the BBH from the graph. If the graph does not contain a BBH, 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 6.2.1. An *instance* of the PERPEXPLORATION-BBH problem is a tuple $\langle G, k, h, v_b \rangle$, 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 $v_b \in (V \setminus \{h\}) \cup \{NULL\}$ is the node that contains the BBH. If $v_b = NULL$, then G does not contain a BBH.

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

Definition 6.2.2. 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 6.2.3. Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_t$ be the connected components of the graph $G - v_b$, resulting from the removal of v_b and all its incident edges from G. If $v_b = NULL$, then $t = 1$ and $\mathcal{C}_1 \equiv G$. Without loss of generality, let $h \in \mathcal{C}_1$.

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 $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_t$ 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 $\mathcal{C}_1$ (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 the problem PERPEXPLORATION-BBH.

Next, we discuss some preliminary notions, used to prove Theorem 6.3.12 and Theorem 6.3.17.

For the following, fix a PERPEXPLORATION-BBH instance $I = \langle G, k, h, v_b \rangle$, where $G = (V, E, \lambda)$, let $\mathcal{A}$ be an algorithm, and fix an execution $\mathcal{E}$ of $\mathcal{A}$ on I.

Definition 6.2.4 (Destruction Time)**.** The *destruction time*, denoted by T_d, is the first round in which the adversary activates the BBH and destroys an agent.

Note that, for a given execution, T_d can be infinite if $v_b = NULL$, or if the adversary never chooses to activate the black hole, or if it never destroys any agent.

Definition 6.2.5 (Benign execution)**.** We say that an execution of $\mathcal{A}$ on I is *benign up to time T* if $T_d > T$. We say that it is *benign* if T_d is infinite.

Definition 6.2.6 (Benign continuation)**.** The *benign continuation from time* t_0 of an execution $\mathcal{E}$ of $\mathcal{A}$ on I is an execution which is identical to $\mathcal{E}$ up to time t_0, and for $t > t_0$ the BBH is never activated.

Note that agents are deterministic and the system is synchronous. Therefore, if $v_b = NULL$, then there exists a unique execution of $\mathcal{A}$ on I, which, in addition, is benign. If the system does contain a BBH, then there exist different executions for different sequences of actions by the adversary. However, the benign execution is still unique in this case.

Definition 6.2.7 (History of an agent)**.** The *history* of an agent a at time $t \geq 1$ is the finite sequence of inputs received by agent a, at the beginning of rounds $1, \ldots, t$.

Definition 6.2.8 (Consistent instances)**.** An instance I' is said to be $(\mathcal{E}, I, t)$-consistent if there exists an execution $\mathcal{E}'$ of $\mathcal{A}$ on I' such that:

- At the beginning of round t, the sets of histories of all alive agents in $\mathcal{E}$ and in $\mathcal{E}'$ are identical.
- If $\tilde{\mathcal{E}}$ and $\tilde{\mathcal{E}}'$ are the infinite benign continuations of $\mathcal{E}$ and $\mathcal{E}'$, respectively, from time t, then the suffixes of $\tilde{\mathcal{E}}$ and $\tilde{\mathcal{E}}'$ starting from time t are identical.

We denote by $Cons(\mathcal{E}, I, t)$ the set of all $(\mathcal{E}, I, t)$-consistent instances.

The usefulness of Definition 6.2.8 lies in the fact that agents with a certain history in a given execution in a given instance can be swapped into any other consistent instance and, in a benign continuation, behave exactly as they would in the original instance.

6.3 Path 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 6.3.1. *4 agents are necessary and sufficient to solve* PERPEXPLORATION-BBH *in path graphs, without initial knowledge of the size of the graph.*

Theorem 6.3.2. *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 6.3.2, 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 6.3.1.

In Section 6.3.1, we propose our lower bound proofs on the number of agents. Then, in Section 6.3.2, we describe the algorithms.

6.3.1 Lower bounds in paths

Lemma 6.3.3. *If $\mathcal{A}$ solves* PERPEXPLORATION-BBH *with k_0 agents, then for every instance I on a path graph with $k \geq k_0$ agents, there is no execution $\mathcal{E}$ of $\mathcal{A}$ on I such that all of the following hold:*

- *At some time $t \geq 1$, exactly one agent a remains alive.*
- $\text{Cons}(\mathcal{E}, I, t)$ *contains instances $I_1 = \langle P, h, k, v_{b1}\rangle$ and $I_2 = \langle P, h, k, v_{b2}\rangle$, where P is a port-labeled path and v_{b1}, v_{b2} are two distinct nodes of P.*

Proof. For a contradiction, let $\mathcal{E}$ be an execution of $\mathcal{A}$ on I that satisfies both conditions simultaneously at time $t \geq 1$, and let $\texttt{Hist}_a$ be the history of the only remaining alive agent a at time t. By consistency, there exist executions $\mathcal{E}_1$ on I_1 and $\mathcal{E}_2$ on I_2 such that, under both, the history of a at time t is exactly $\texttt{Hist}_a$. Moreover, the benign continuations $\tilde{\mathcal{E}}_1, \tilde{\mathcal{E}}_2$ share a common suffix starting from time t.

We distinguish the following cases:

- In $\tilde{\mathcal{E}}_1$ and $\tilde{\mathcal{E}}_2$, at some $t' \geq t$, agent a visits v_{b1} (resp. v_{b2}). Then, the algorithm fails in instance I_1 (resp. I_2) by activating the BBH at time t'.

- In $\tilde{\mathcal{E}}_1$ and $\tilde{\mathcal{E}}_2$, agent a never visits v_{b1} or v_{b2} at any time $t' \geq t$. Let pos be the position of the agent at time t. If v_{b1} and v_{b2} are on the same side of pos, and v_{b1} (resp. v_{b2}) is the farthest from pos, then the algorithm fails on instance I_1 (resp. I_2) because, after time t, it never visits node v_{b2} (resp. v_{b1}), which is in the same component as the agent and is not the BBH. If pos is between v_{b1} and v_{b2}, then the algorithm fails on both instances I_1 and I_2. To see why, consider the instance I_1: in I_1, v_{b2} is not the BBH and it is in the same component as the agent, but is never visited after time t. The reasoning is similar, for instance I_2.

In all cases, we obtain a contradiction with the correctness of algorithm $\mathcal{A}$. □

Lemma 6.3.4. *If $\mathcal{A}$ solves* PERPEXPLORATION-BBH *with k_0 agents, then for every instance I on a path graph with $k \geq k_0$ agents, there is no execution $\mathcal{E}$ of $\mathcal{A}$ on I such that all of the following hold:*

- *At some time $t \geq 1$, exactly two agents a, b remain alive.*
- *$\text{Cons}(\mathcal{E}, I, t)$ contains instances $I_i = \langle P, h, k, v_{bi} \rangle$, $1 \leq i \leq 3$, where P is a port-labeled path and $\{v_{bi} : 1 \leq i \leq 3\}$ are three distinct nodes of P. Let $\mathcal{B}$ be the common suffix, starting from time t, of all benign continuations of the executions witnessing the $(\mathcal{E}, I, t)$-consistency of I_i, for $1 \leq i \leq 3$.*
- *If $\texttt{pos}_a, \texttt{pos}_b$ are the positions of the agents at time t under $\mathcal{B}$, then $\texttt{pos}_a, \texttt{pos}_b$ are on the same side of all of $\{v_{bi} : 1 \leq i \leq 3\}$.*

Proof. For a contradiction, let $\mathcal{E}$ be an execution of $\mathcal{A}$ on I that satisfies all of the conditions simultaneously at time $t \geq 1$, let v_{b1}, v_{b2}, v_{b3} be ordered by increasing distance from $\texttt{pos}_a$ and $\texttt{pos}_b$, and let P_1 denote the subpath of P from v_{b1} to an extremity of P, such that P_1 does not contain any of $\texttt{pos}_a$, $\texttt{pos}_b$.

We distinguish the following cases:

Case 1: In $\tilde{\mathcal{E}}_1$, $\tilde{\mathcal{E}}_2$, and $\tilde{\mathcal{E}}_3$, there exists some time $t' \geq t$ such that agent a (without loss of generality) enters P_1 (hence moves to v_{b1} from outside P_1), and agent b moves to, from, or stays on some node of P_1.

In this case, consider the smallest such t', and let $t'_1 \leq t'$ be the last time that agent b enters P_1 before t'. Note that agent a enters P_1 at time t' without waiting for agent b to exit P_1. Therefore, even if agent b is destroyed by v_{b1} at time t'_1, agent a will still enter P_1 at time t'. We obtain a contradiction with the correctness of algorithm $\mathcal{A}$, as it fails on instance I_1 if the BBH v_{b1} is activated at times t' and t'_1.

Case 2: In $\tilde{\mathcal{E}}_1$, $\tilde{\mathcal{E}}_2$, and $\tilde{\mathcal{E}}_3$, whenever an agent enters P_1 at some time $t' \geq t$, then at that time the other agent is not inside P_1, nor is entering or exiting P_1.

In this case, there must exist at least one time $t' \geq t$ such that agent a (without loss of generality) enters P_1 and moves on to reach v_{b2} before exiting P_1, while agent b remains outside of P_1. Indeed, if this never happens, then the algorithm fails on instance I_3 as it never explores node v_{b2}.

Let $t'_1 > t'$ be the first time after t' such that agent a visits v_{b2}. Now, consider the following executions:

- $\mathcal{E}'_1$ on I_1: same as $\tilde{\mathcal{E}}_1$ up to time t', then v_{b1} is activated at time t' and destroys agent a, then v_{b1} is never activated after t'.
- $\mathcal{E}'_2$ on I_2: same as $\tilde{\mathcal{E}}_2$ up to time t'_1, then v_{b2} is activated at time t'_1 and destroys agent a, then v_{b2} is never activated after t'_1.

Clearly, after time t' and as long as agent a hasn't exited P_1, agent b receives exactly the same inputs in both executions. Therefore, the history of agent b is identical in both executions up to at least time t'_1. Let $\texttt{Hist}'_b$ be the history of agent b at that time. Moreover, the benign continuations of both $\mathcal{E}'_1$ and $\mathcal{E}'_2$ after time t'_1 must be identical. It follows that $\text{Cons}(\mathcal{E}'_1, I_1, t'_1)$ contains I_1 and I_2. This contradicts Lemma 6.3.3. □

Lemma 6.3.5. *If $\mathcal{A}$ solves* PERPEXPLORATION-BBH *with k_0 agents, then for every instance I on a path graph with $k \geq k_0$ agents, there is no execution $\mathcal{E}$ of $\mathcal{A}$ on I such that all of the following hold:*

- *At some time $t \geq 1$, exactly two agents a, b remain alive.*
- $\text{Cons}(\mathcal{E}, I, t)$ *contains instances $I_i = \langle P, h, k, v_{bi}\rangle$, $1 \leq i \leq 3$, where P is a port-labeled path and $\{v_{bi} : 1 \leq i \leq 3\}$ are three distinct nodes of P. Let $\mathcal{B}$ be the common suf-*

fix, starting from time t, of all benign continuations of the executions witnessing the $(\mathcal{E}, I, t)$-consistency of I_i, for $1 \le i \le 3$.

- *If $\texttt{pos}_a, \texttt{pos}_b$ are the positions of the agents at time t under $\mathcal{B}$, then the path connecting $\texttt{pos}_a$ to $\texttt{pos}_b$ contains all of $\{v_{bi} : 1 \le i \le 3\}$.*

Proof. For a contradiction, let $\mathcal{E}$ be an execution of $\mathcal{A}$ on I that satisfies all of the conditions simultaneously at time $t \ge 1$, and let v_{b1}, v_{b2}, v_{b3} be ordered by increasing distance from $\texttt{pos}_a$ (and, therefore, decreasing distance from $\texttt{pos}_b$).

Claim 6.3.6. *In $\mathcal{B}$, no agent ever visits v_{b2} after time t.*

Proof of Claim. Suppose the contrary and, without loss of generality, let a be the first agent to visit v_{b2} at time $t_2 \ge t$. As the two agents are initially on either side of v_{b2}, it follows that the agents do not meet before t_2. Moreover, in order to reach v_{b2}, agent a must also visit v_{b1} at some time $t_1 < t_2$. By activating the BBH at times t_1, t_2 in the respective instances, agent a may be destroyed on either of v_{b1}, v_{b2}. Let $\mathcal{B}'$ be the same as $\mathcal{B}$ up to t_2, when v_{b2} is activated once to destroy agent a and then remains benign. By the lack of communication between agents, $\text{Cons}(\mathcal{B}', I_2, t_2)$, contains I_1, I_2, which contradicts Lemma 6.3.3. □

A corollary of the above claim is that agents never meet after time t. We can further show that agent a never visits v_{b1} and, by symmetry, agent b never visits v_{b3}. Indeed, if agent a ever visits v_{b1}, then the algorithm fails on instance I_1 by destroying agent a. The remaining agent b will never visit v_{b2}, thus failing to perpetually explore its component. We conclude that no agent ever visits any of v_{b1}, v_{b2}, v_{b3} after time t, and therefore the algorithm fails on instance I_2 because no agent perpetually explores its component. □

Lemma 6.3.7. *If $\mathcal{A}$ solves* PERPEXPLORATION-BBH *with k_0 agents, then for every instance I on a path graph with $k \ge k_0$ agents, there is no execution $\mathcal{E}$ of $\mathcal{A}$ on I such that all of the following hold:*

- *At some time $t \ge 1$, exactly three agents a, b, c remain alive.*

- Cons$(\mathcal{E}, I, t)$ *contains instances* $I_i = \langle P, h, k, v_{bi}\rangle$, $1 \le i \le 7$, *where* P *is a port-labeled path and* $\{v_{bi} : 1 \le i \le 7\}$ *are seven distinct nodes of* P. *Let* $\mathcal{B}$ *be the common suffix, starting from time* t, *of all benign continuations of the executions witnessing the* $(\mathcal{E}, I, t)$*-consistency of* I_i, *for* $1 \le i \le 7$.

- *If* $\texttt{pos}_a, \texttt{pos}_b, \texttt{pos}_c$ *are the positions of the agents at time* t *under* $\mathcal{B}$, *then* $\texttt{pos}_a, \texttt{pos}_b, \texttt{pos}_c$ *are on the same side of all of* $\{v_{bi} : 1 \le i \le 7\}$.

- *Assuming without loss of generality that nodes* v_{bi} *are ordered by increasing distance from* $\texttt{pos}_a, \texttt{pos}_b, \texttt{pos}_c$, *and letting* P_1 *denote the subpath of* P *from* v_{b1} *to an extremity of* P *such that* P_1 *does not contain* $\texttt{pos}_a, \texttt{pos}_b, \texttt{pos}_c$, *then in* $\mathcal{B}$, *at most two agents are ever simultaneously in* P_1.

Proof. For a contradiction, let $\mathcal{E}$ be an execution of $\mathcal{A}$ on I that satisfies all of the conditions simultaneously at time $t \ge 1$, and let, for $i \ge 2$, P_i denote the subpath of P_1 spanning the nodes from v_{bi} to an extremity of P. We say that a P_i-event occurs at time $t_0 \ge t$ $(1 \le i \le 7)$, if an agent enters P_i at time t_0 while another agent is already in P_i.

Claim 6.3.8. *In* $\mathcal{B}$, *a* P_4*-event must occur at some time after* t.

Proof of Claim. Indeed, suppose that in $\mathcal{B}$, at most one agent is ever inside P_4 after time t. Then, there must exist some time $t_1' \ge t$ at which agent a (without loss of generality) enters P_4 and does not exit P_4 before reaching v_{b6} at time $t_2' \ge t_1'$, while the other agents stay outside of P_4. Agent a may be destroyed on any of v_{b4}, v_{b5}, v_{b6} by activating the BBH in the respective instance. It follows that I_4, I_5, I_6 are all in Cons$(\mathcal{B}', I_6, t_2')$, where $\mathcal{B}'$ is the same as $\mathcal{B}$ up to time t_2', when v_{b6} is activated. This contradicts Lemma 6.3.4. □

Note that a P_4-event requires a prior P_3-event. Let t_4 be the time of the first P_4-event in $\mathcal{B}$, involving agents a, b without loss of generality. Let $t_3 \le t_4$ be the time of the last P_3-event before t_4. By definition, the P_3-event at time t_3 must involve the same agents a, b and none of these agents may have exited P_3 between t_3 and t_4. Without loss of generality, let agent b be the one that enters P_3 at time t_3. Therefore, this is the last time that agent b enters P_3 before t_4. Let $t_1 \le t_3$ denote the last time agent a entered P_3.

Claim 6.3.9. *In* $\mathcal{B}$, *agent* c *does not enter* P_3 *between* t_1 *and* t_3.

Proof of Claim. Suppose that agent c enters P_3 at time t_1' ($t_1 \le t_1' < t_3$). Given that agent b enters P_3 at time t_3, and by the assumption that at most two agents may ever be in P_1, it follows that agent c must reach v_{b1} at some time t_2' between t_1' and t_3, while agent a remains in P_3 and agent b remains outside of P_1. On the way out, agent c may be destroyed by any of v_{b1}, v_{b2}, v_{b3} by activating the BBH in the respective instance. It follows that I_1, I_2, I_3 are all in $\text{Cons}(\mathcal{B}', I_1, t_2')$, where $\mathcal{B}'$ is the same as $\mathcal{B}$ up to time t_2', when v_{b1} is activated. This contradicts Lemma 6.3.5. □

It follows that only agent b may enter P_3 between t_1 and t_3. Let $t_2 \ge t_1$ be the first time this happens. Then, agent b may not exit P_1 after time t_2, by a similar argument as in the proof of Claim 6.3.9. There is, therefore, no communication after time t_1 between agent c and either of agents a, b. Finally, let $t_4' \le t_4$ be the time when the agent that is already in P_4 at time t_4 enters P_4 for the last time. Now, consider the following executions:

- $\mathcal{B}_1$ on I_3: same as $\mathcal{B}$ up to time t_1, when v_{b3} is activated destroying agent a. Then, benign up to time t_3, when v_{b3} is activated destroying agent b. Then, v_{b3} is never activated after t_3.
- $\mathcal{B}_2$ on I_4: same as $\mathcal{B}$ up to time t_4', when v_{b4} is activated. Then, benign up to time t_4, when v_{b4} is activated. At this point, both agents a, b are destroyed and v_{b4} is never activated after t_4.

Clearly, $\text{Cons}(\mathcal{B}_1, I_3, t_4)$ contains both of I_3, I_4, which contradicts Lemma 6.3.3. □

Lemma 6.3.10. *If $\mathcal{A}$ solves* PERPEXPLORATION-BBH *with k_0 agents, then for every instance I on a path graph with $k \ge k_0$ agents, there is no execution $\mathcal{E}$ of $\mathcal{A}$ on I such that all of the following hold:*

- *At some time $t \ge 1$, exactly three agents a, b, c remain alive.*
- $\text{Cons}(\mathcal{E}, I, t)$ *contains instances $I_i = \langle P, h, k, v_{bi} \rangle$, $1 \le i \le 8$, where P is a port-labeled path and $\{v_{bi} : 1 \le i \le 8\}$ are eight distinct nodes of P.*
- *If $\texttt{pos}_a, \texttt{pos}_b, \texttt{pos}_c$ are the positions of the agents at time t under $\mathcal{E}$, then $\texttt{pos}_a, \texttt{pos}_b, \texttt{pos}_c$ are on the same side of all of $\{v_{bi} : 1 \le i \le 8\}$.*

- *Assuming without loss of generality that nodes v_{bi} are ordered by increasing distance from $\texttt{pos}_a, \texttt{pos}_b, \texttt{pos}_c$, nodes v_{b1}, v_{b2}, v_{b3} are consecutive in P.*

Proof. For a contradiction, let $\mathcal{E}$ be an execution of $\mathcal{A}$ on I that satisfies all of the conditions simultaneously at time $t \geq 1$. Let $\mathcal{B}$ be the common suffix, starting from time t, of all benign continuations of the executions witnessing the $(\mathcal{E}, I, t)$-consistency of I_i, for $1 \leq i \leq 8$. Let, for $i \geq 1$, P_i denote the subpath of P spanning the nodes from v_{bi} to an extremity of P, such that P_i does not contain any of $\texttt{pos}_a, \texttt{pos}_b, \texttt{pos}_c$. We say that a P_i^j-event occurs at time $t_0 \geq t$ ($1 \leq i \leq 8$, $1 \leq j \leq 3$), if some agent enters P_i at time t_0 and at least j agents are in P_i at time t_0 (including the one or the ones that entered at time t_0).

By Lemma 6.3.7, a P_2^3-event must occur in $\mathcal{B}$. Let T be the time of the first P_2^3-event, and assume, without loss of generality, that agent c enters P_2 at time T while agents a, b are in P_2.

Claim 6.3.11. *In $\mathcal{B}$, there is no P_3^2-event involving agents a, b at or before time T.*

Proof of Claim. For a contradiction, let $t_4 \leq T$ be the time of the last P_3^2-event before time T, and assume, without loss of generality, that agent a enters P_3 while agent b is in P_3. Let $t_3 \leq t_4$ be the time of the last P_2^2-event before t_4 involving agents a, b. Without loss of generality, let agent a be the one that enters P_2 at time t_3, and let $t_1 \leq t_3$ be the last time agent b entered P_2.

Consider the following executions:

- $\mathcal{B}_1$ on I_1: same as $\mathcal{B}$ up to time t_3, when v_{b1} is activated destroying agent b. Then, benign up to time T, when v_{b1} is activated destroying agent c. Then, v_{b1} is never activated after T.

- $\mathcal{B}_2$ on I_2: same as $\mathcal{B}$ up to time t_3, when v_{b1} is activated destroying agent b. Then, benign up to time T, when v_{b1} is activated destroying agent c. Then, v_{b2} is never activated after T.

Clearly, $\text{Cons}(\mathcal{B}_1, I_1, T)$ contains both of I_1, I_2, which contradicts Lemma 6.3.3. □

By Claim 6.3.11, either none of a, b has entered P_3 until T, or only one of a, b has entered P_3 until T. We complete the proof by distinguishing the following cases:

Case 1: None of a, b has entered P_3 until time T. Note that, by assumption, a P_2^3-event occurs at time T. It follows that, in that round, all agents are either on, moving to, or moving from node v_{b2}. Therefore, they can all be destroyed in instance I_2 by activating v_{b2} at time T.

Case 2: Agent a (without loss of generality) enters or is in P_3 at time T, and agents b, c enter simultaneously P_2 at time T. Then, the adversary can construct executions $\mathcal{E}_1$ (resp. $\mathcal{E}_2$) in which it follows $\mathcal{B}$ up to time T, when it activates the BBH v_{b1} (resp. v_{b2}) in instance I_1 (resp. I_2) to destroy both agents b, c. $\text{Cons}(\mathcal{E}_1, I_1, T)$ contains both of I_1, I_2, which contradicts Lemma 6.3.3.

Case 3: Each of agents a, b is either on or moving to v_{b2} at time T. In this case, the adversary can destroy all of the agents in instance I_2 by activating v_{b2} at time T.

Case 4: At round T, agent a is at v_{b3}, agent b is at v_{b2}, and agent c moves to v_{b2} from v_{b1}. Let $t_a \leq T$ be the last time before T that a moved from v_{b2} to v_{b3}, and let $t_b \leq T$ be the last time before T that b moved from v_{b1} to v_{b2}. If $t_b \leq t_a$, then the adversary can activate v_{b2} in instance I_2 between times t_a and T to destroy all the agents. If $t_a < t_b$, then the adversary can construct executions $\mathcal{E}_1$ (resp. $\mathcal{E}_2$) in which it follows $\mathcal{B}$ up to t_b, and then it keeps activating the BBH v_{b1} (resp. v_{b2}) in instance I_1 (resp. I_2) to destroy both agents b, c. We then have $I_1, I_2 \in \text{Cons}(\mathcal{E}_1, I_1, T)$, which contradicts Lemma 6.3.3. □

Theorem 6.3.12. *There is no algorithm that solves* PERPEXPLORATION-BBH *with* 3 *agents on all path graphs with at least* 9 *nodes, even assuming that the agents know the number of nodes.*

Proof. Let $\mathcal{A}$ be an algorithm that solves PERPEXPLORATION-BBH with 3 agents on all path graphs with at least 9 nodes. Fix a path P with $n \geq 9$ nodes. Let the home node h be one of the extremities of P, and let v_i, $1 \leq i \leq 8$ be the node at distance i from h. Let 3 agents start at node h and consider the set of instances $\mathcal{I} = \{\langle P, h, 3, v_i \rangle : 1 \leq i \leq 8\}$. Even with knowledge of n, at the very beginning of the benign execution $\mathcal{E}$ of $\mathcal{A}$ (beginning of first round) in, say, I_1, every instance in $\mathcal{I}$ is contained in $\text{Cons}(\mathcal{E}, I_1, 1)$. Moreover, v_1, v_2, v_3 are consecutive in P. This contradicts Lemma 6.3.10. □

We further refine the result of Theorem 6.3.12 to show that PERPEXPLORATION-BBH-

HOME is actually impossible even with 5 agents in sufficiently large paths, even assuming knowledge of the size of the graph (Theorem 6.3.17). To do that, we use the following notion of *suspicious nodes* to simplify the presentation of the proofs. The suspicious nodes are potential positions of the BBH on the path. Formally, given an instance $I = \langle P, k, h, v_b \rangle$, where $P = (V, E, \lambda)$ is a port-labeled path graph, and an execution $\mathcal{E}$ on I, the set $S(t)$ of suspicious nodes at time t is defined as:

$$S(t) = \{s : \langle P', k, h, s \rangle \in \text{Cons}(\mathcal{E}, I, t) \text{ for some path } P'\}$$

Since the graph is assumed anonymous, in this definition a node is identified by its signed distance from the home node h in P, by assuming that the positive (resp. negative) direction on the path is the direction of outgoing port number 1 (resp. 2) from h.

Note that, as long as the agents have not learned the size of the path and if the BBH has not been activated yet, $S(t)$ is an infinite set. Indeed, $\text{Cons}(\mathcal{E}, I, t)$ contains all arbitrarily long paths which are consistent with the finite history of the agents at time t.

In the following part, though, we will assume that the agents know the size of the path. This simplifies the definition of $S(t)$ to:

$$S(t) = \{s : \langle P, k, h, s \rangle \in \text{Cons}(\mathcal{E}, I, t)\}$$

A node is said to be *safe* if it is in $V \setminus S(t)$. At the beginning of an execution only the home node can be assumed safe, therefore $S(1) = V \setminus \{h\}$. In subsequent rounds, each agent keeps its own track of the suspicious node set, which can only be reduced. Indeed, if an agent detects the destruction of one or more agents, it can eliminate from its suspicious node set those nodes that could not house the BBH. Note, however, that it may happen during an execution that one or more agents have an outdated idea of the suspicious node set, specifically if they have not yet been able to receive information from the agent(s) that first detected the destruction of an agent. In this case, the agents with outdated information must still continue to behave consistently with their presumed suspicious node set until they can communicate with the agents having the most current information. We use the

notation $S_i(t)$ to denote the suspicious node set of agent with ID i at time t.

In all the following lemmas and theorems, we assume that $n \geq 9$.

Lemma 6.3.13. *For any exploration algorithm $\mathcal{A}$ with two agents a_1 and a_2 on a path graph $P = (V, E, \lambda) = L[v_0, v_{n-1}]$ containing a BBH, where $L[v_i, v_j]$ indicates the induced subgraph of P induced by the nodes $\{v_x : 0 \leq i \leq x \leq j \leq |V| - 1\}$. There exists a time $T_f \geq T_d$ such that the adversary can create one of the two following situations at T_f.*

1. *Both agents are destroyed.*
2. *Exactly one agent, say a_i stays alive and $|S_i(T_f)| \geq \lfloor \frac{n}{2} \rfloor$.*

Proof. Let the path graph $P = (V, E, \lambda) = L[v_0, v_{n-1}]$ where $|V| = n$. Without loss of generality, we denote the home h to be v_0. It may be noted that $S_i(T) = S_i(0) = L[v_1, v_{n-1}]$ where $i \in \{1, 2\}$ for all $0 \leq T \leq T_d$. Let us consider the ordered sequence of line segments $L_{\lfloor \frac{n}{2} \rfloor} \subset L_{\lfloor \frac{n}{2} \rfloor - 1} \subset \ldots L_2 \subset L_1 \subset P$, where $L_i = L[v_i, v_{n-1}]$. Now we claim that if there is a round T_i such that at T_i both a_1 and a_2 are in P_i then the adversary can create i many instances, denoted by $I_j = \langle P, 2, v_0, v_j \rangle$ where both agents are destroyed, where $0 \leq j \leq i$.

Let the benign execution occur at some round T_i, where both a_1 and a_2 are in P_i. Therefore, there is a round such that at T_j, both a_1 and a_2 are in L_j, where $j \leq i$. Now, without loss of generality, let a_1 first enter L_j before a_2. Let T'_j be the last time when a_1 was on v_j before T_j. So, by activating the BBH at T'_j and T_j, the adversary can destroy both a_1 and a_2, creating scenario 1 for all $j \leq i$. Here $T_d = T'_j$ and $T_f = T_j$.

So, if $i = \lfloor \frac{n}{2} \rfloor$ then, adversary can create $\lfloor \frac{n}{2} \rfloor$ many instances destroying each of the agents in all those instances given there is a time $T_{\lfloor \frac{n}{2} \rfloor}$ when both agents are in $L_{\lfloor \frac{n}{2} \rfloor}$.

Now, let for all T exactly one agent, say a_1, remains outside of $L_{\lfloor \frac{n}{2} \rfloor}$. Due to the benign execution property, a_2 has to explore the whole path graph. So, there exists a time T_* when a_2 is on v_{n-1}. Let T'_* be the last time a_2 was on $v_{\lfloor \frac{n}{2} \rfloor}$ before T_*. Then adversary can create $\lfloor \frac{n}{2} \rfloor$ many instances denoted as $I^*_j = \langle P, 2, v_0, v_j \rangle$ where $\lfloor \frac{n}{2} \rfloor \leq j \leq n - 1$. In each of these instances, a_2 can be destroyed at v_b at time T_d by the adversary after T'_*, and a_1 can not distinguish between any of them. So, for the alive agent a_1, $S_1(T_f) \geq \lfloor \frac{n}{2} \rfloor$ where $T_f = T_d$. □

Lemma 6.3.14. *For any exploration algorithm $\mathcal{A}$ on a path graph $P = (V, E, \lambda) = L[v_0, v_{n-1}]$ with 3 agents a_1, a_2 and a_3, in presence of a BBH, there exists a time $T_f \geq T_d$ such that at T_d, the adversary can create any of the following three scenarios.*

1. *All three agents are destroyed.*
2. *At least one agent stays alive at the component $\mathcal{C}_1$ and no agents are in $\mathcal{C}_2$ of the graph $P \setminus \{v_b\}$ where $\mathcal{C}_1$ contains the home h, $P - v_b = \mathcal{C}_1 \cup \mathcal{C}_2$ and for all alive agents a_j, $|S_j(T_f)| \geq \lfloor \frac{n}{4} \rfloor$.*
3. *Exactly one agent a_i stays alive at component $\mathcal{C}_2$ of the graph $G - \{v_b\}$, where $\mathcal{C}_2$ does not contain the home and $|S_i(T_f)| \geq 2$.*

Proof. Without loss of generality, let $h = v_0$, and we denote let $L_i = L[v_i, v_{n-1}]$ such that $V = \{v_0, \ldots, v_{n-1}\}$. We first show that, if two agents, say a_1 and a_2 does not enter $L_{\lfloor \frac{n}{2} \rfloor}$ but a_3 enters it, then the adversary can create scenario 2 at some time T_f, where two agents a_1 and a_2 stays alive at $\mathcal{C}_1$ and $|S_i(T_f)| \geq \lfloor \frac{n}{2} \rfloor > \lfloor \frac{n}{4} \rfloor$, for $i \in \{1,2\}$ and for some $T_f \geq T_d$. In this case, due to a benign execution property, there must exist a time T_1 when a_3 is at v_{n-1}, and let T_1' be the last time before T_1, when a_3 was on $v_{\lfloor \frac{n}{2} \rfloor}$. Now, the adversary can create $\lfloor \frac{n}{2} \rfloor$ distinct instances denoted as I_α where $I_\alpha = \langle P, 3, v_0, v_{n-1-\alpha} \rangle$ and $0 \leq \alpha < \lfloor \frac{n}{2} \rfloor$ such that it can destroy a_3 in each of these instances at round T_d between T_1 and T_1'. We consider $T_f = T_d$ here. This construction ensures that a_2 and a_3 can not distinguish between these instances. This implies $|S_i(T_f)| \geq \lfloor \frac{n}{2} \rfloor > \lfloor \frac{n}{4} \rfloor$ for all $i \in \{1,2\}$ where a_1 and a_2 are in $\mathcal{C}_1$.

So, between T_1' and T_1, if none among a_1 and a_2 enters $L_{\lfloor \frac{n}{2} \rfloor}$, then the adversary can create a situation worse than scenario 2.

So, let us assume between T_1' and T_1 among a_1 and a_2 only a_2 moved to $L_{\lfloor \frac{n}{2} \rfloor}$ by being on $v_{\lfloor \frac{n}{2} \rfloor}$, for the first time at T_2 and a_1 never moves to $L_{\lfloor \frac{n}{2} \rfloor}$. Now, between T_2 and T_1 if a_2 never enters $L_{\lfloor \frac{3n}{4} \rfloor}$ then, again adversary can create $\lfloor \frac{n}{4} \rfloor$ instances $I_\beta = \langle P, 3, v_0, v_{\lfloor \frac{3n}{4} + \beta \rfloor} \rangle$ where $0 \leq \beta \leq \lfloor \frac{n}{4} \rfloor$, and destroy a_3 at T_d for each of these instances between T_1'' and T_1, T_1'' being the last time a_3 was on $v_{\lfloor \frac{3n}{4} \rfloor}$ before T_1. Note that a_2 and a_3 can not distinguish between these instances due to the construction. Therefore, if we consider $T_f = T_d$, then the adversary can still create scenario 2 in this case.

Now let there exists a time T_{x_0} between T_1' and T_1, when both a_2 and a_3 are in $L_{\lfloor \frac{3n}{4} \rfloor}$. So, there exists T_{x_i} in between T_1' and T_1 such that both a_2 and a_3 are in $L_{\lfloor \frac{3n}{4} \rfloor - i}$ and $T_{x_{i+1}} < T_{x_i}$ for all i where $0 \leq i \leq \lfloor \frac{n}{4} \rfloor$. By similar argument as the second paragraph of Lemma 6.3.13, adversary can destroy both of a_2 and a_3 for each of the $\lfloor \frac{n}{4} \rfloor$ instances $I_i = \langle P, 3, v_0, v_{\lfloor \frac{3n}{4} \rfloor - i} \rangle$ in between time T_1' and T_1. Let T_f be the time when the last alive agent among a_2 and a_3 is killed, and $T_1' \leq T_d \leq T_f \leq T_1$. Due to this construction, $a_1 \in \mathcal{C}_1$, can not distinguish between these instances, and so the adversary can again create scenario 2 at T_f.

So, to be specific, till now we have concluded that if there is at least one agent that never enters $L_{\lfloor \frac{n}{2} \rfloor}$ in between T_1' and T_1 then the adversary can always create scenario 2.

So, now let us consider the case when there is a time T_{*1} between T_1' and T_1 when all three agents are in $L_{\lfloor \frac{n}{2} \rfloor}$. Let T_* be the first time when all agents are in $L_{\lfloor \frac{n}{2} \rfloor}$. Let a_{x_i} enters $L_{\lfloor \frac{n}{2} \rfloor}$ last time before T_* at T_{x_i}. Here, for any $i \in \{1,2,3\}$, $x_i \in \{1,2,3\}$ and $x_i \neq x_j$ if $i \neq j$. Without loss of generality let, $T_{x_1} \leq T_{x_2} \leq T_{x_3} \leq T_*$. Now if $T_{x_1} = T_{x_2}$ then for the instance $I = \langle P, 3, v_0, v_{\lfloor \frac{n}{2} \rfloor} \rangle$, adversary can destroy a_{x_1} and a_{x_2} at $T_{x_1} = T_{x_2}$ and can destroy a_{x_3} at time T_{x_3}, creating scenario 1 at time $T_f = T_{x_3}$. Now let us consider another case, $T_{x_1} < T_{x_2} \leq T_*$. Let $T_{x_2} < T_*$ and let at T_{x_2}, a_{x_3} is in $L_{\lfloor \frac{n}{2} \rfloor}$. This contradicts the assumption that T_* is the first round when all three agents are inside $L_{\lfloor \frac{n}{2} \rfloor}$. Thus either $T_{x_2} = T_*$, or, $T_{x_2} < T_*$ and a_{x_3} is outside of $L_{\lfloor \frac{n}{2} \rfloor}$ at time T_{x_2}. When $T_{x_2} = T_*$ then $T_{x_2} = T_{x_3} = T_*$. In this case for both the instances $I_{*_1} = \langle P, 3, v_0, v_{\lfloor \frac{n}{2} \rfloor - 1} \rangle$ and $I_{*_2} = \langle P, 3, v_0, v_{\lfloor \frac{n}{2} \rfloor} \rangle$, adversary can destroy both a_{x_2} and a_{x_3} at time T_{x_2}. Now, if at T_{x_2}, a_{x_1} is also at $v_{\lfloor \frac{n}{2} \rfloor}$ or moving to it, then adversary destroys all of them, leading to scenario 1 at time $T_f = T_{x_2}$. On the other hand, if a_{x_1} is not at $v_{\lfloor \frac{n}{2} \rfloor}$ or not moving at it at round T_{x_2} then a_{x_1} can not distinguish between both of the above mentioned instances. So, this case leads to scenario 3 at time $T_f = T_{x_2}$. Now let at $T_{x_2} (< T_{x_3} \leq T_*)$, a_{x_3} is not in $L_{\lfloor \frac{n}{2} \rfloor}$. Let us consider the time span starting at T_{x_2} ending at $T_{x_3}^*$ where $T_{x_3}^*$ is the time when a_{x_3} enters $L_{\lfloor \frac{n}{2} \rfloor}$ for the first time after T_{x_2}. Now let us consider the two instances I_{*_1} and I_{*_2}, as discussed earlier. If the adversary activates the BBH for all rounds in the above mentioned time span, i.e., between T_{x_2} and $T_{x_3}^*$, then it can destroy both the agents a_{x_2} and a_{x_3}. Now if in this time span a_{x_1} visits $v_{\lfloor \frac{n}{2} \rfloor}$ then it will be destroyed creating scenario 1 at $T_f = T_{x_3}^*$. Otherwise if it does not visit $v_{\lfloor \frac{n}{2} \rfloor}$ then only a_{x_1} stays alive at T_{x_3} without distinguishing between the instances. This leads to scenario

3 at $T_f = T^*_{x_3}$.

Note that if there is a time when all three agents are in $L_{\lfloor \frac{n}{2} \rfloor}$ implies there is a time T_{*_i} when all three agents are in $L_{\lfloor \frac{n}{2} \rfloor - i}$, where $0 \leq i \leq \lfloor \frac{n}{4} \rfloor$. So with similar reasoning as above for each, we can create scenario 1 or scenario 3 where the BBH can be at any one of the vertices, $v_{\lfloor \frac{n}{2} \rfloor - i}$. □

Lemma 6.3.15. *For any algorithm $\mathcal{A}$ solving* PERPEXPLORATION-BBH-HOME *on a path graph $P = (V, E, \lambda) = L[v_0, v_{n-1}]$, at least one agent needs to be present at home until the destruction time T_d.*

Proof. We prove this lemma by contradiction. Let, without loss of generality, v_0 be the home. Let there be an algorithm $\mathcal{A}$ that solves PERPEXPLORATION-BBH-HOME for the path graph $L[v_0, v_{n-1}]$ such that there exists a time $T (< T_d)$ when there are no agents at the home v_0. Let $i > 0$ be the least integer such that v_i contains any agent at time T. Then for the instance $I = \langle P, k, v_0, v_i \rangle$ if adversary activates the BBH for all rounds starting from round T, then all agents which are alive gets stranded on the component $\mathcal{C}_2$ of graph $P - \{v_b\}$ where $v_b = v_i$ and $\mathcal{C}_2$ does not contain the home v_0. So, it contradicts the assumption that $\mathcal{A}$ solves PERPEXPLORATION-BBH-HOME on P. □

Theorem 6.3.16. *A set of 4 agents cannot solve* PERPEXPLORATION-BBH-HOME *on a path graph of size n', where $n' > 36$.*

Proof. Let us choose the path graph, $P = (V, E, \lambda)$, to be of length $n' = 4n > 36$. Without loss of generality, let v_0 be the *home*. We prove this theorem by proving the following cases one by one.

Case-I: Let until T_d the number of agents staying at *home* is 3. Let a_x be the only agent exploring the path graph P until T_d. In this case, due to benign execution, a_x must visit v_{n-1} at some point, say T_1. Let T'_1 be the time a_x visited v_0 last time before T_1. So adversary can create $n-1$ instances $I_j = \langle P, 4, v_0, v_j \rangle$ where $1 \leq j \leq n-1$ and can destroy a_x in each of these instances in between T'_1 and T_1 at T_d. Thus, the remaining 3 agents at home can not distinguish between these instances. Thus, the problem can be thought of as solving PERPEXPLORATION-BBH-HOME with 3 agents for a path graph of length $\lfloor \frac{n'}{4} \rfloor = n > 9$ (where $n \geq 9$ as assumed), which is impossible due to Theorem 6.3.12.

Case-II: Let until T_d the number of agents staying at *home* is 2.

In this case, by Lemma 6.3.13, adversary can create two possible scenarios, i.e., there exists a time $T_f \geq T_d$ such that, either both the agents are destroyed on or before T_f or, at T_f there is exactly one agent, say a_x among the three agents, remains alive in the component $\mathcal{C}_1$ of graph $P - \{v_b\}$ such that $\mathcal{C}_1$ contains the home and $|S_x(T_f)| \geq \lfloor \frac{n'}{2} \rfloor = 2n > 18$. It can be easily proved that until T_f, no other agent leaves *home* if until T_d, 2 agents stays at *home*. Otherwise, the adversary can force more agents to leave *home* before T_d, which contradicts our assumption. For both of these cases, this problem can be reduced to solving PERPEXPLORATION-BBH-HOME on a path graph of size greater than 9 with at most 3 agents. By Theorem 6.3.12, we conclude that there is no algorithm that solves PERPEXPLORATION-BBH-HOME on a path graph with 4 agents until T_d, if 2 agents stay at *home*.

Case-III: Let until T_d the number of agents staying at *home* is 1.

In this case, for the rest of the 3 agents, the adversary can create one of the three cases as described in Lemma 6.3.14 at some time $T_f \geq T_d$. It can be easily shown that agents who are staying at *home* until T_d never leave *home* until T_f if the number of agents staying at *home* until T_d is 1. It can be shown by providing an adversarial strategy to force the agent at *home* to leave *home* before T_d.

Now, if all three agents are destroyed, then the problem reduces to solving the problem PERPEXPLORATION-BBH-HOME on a path graph with at least $\lfloor \frac{n'}{4} \rfloor = n > 9$ nodes with one agent. This is impossible by Theorem 6.3.12. For the case where by time T_f at least one among the 3 agents exploring is destroyed and the remaining agents a_{x_i} are alive at $\mathcal{C}_1$ ($\mathcal{C}_1$ being the component of $P - \{v_b\}$ that contains the home) with $|S_{x_i}| \geq \lfloor \frac{n'}{4} \rfloor = n > 9$. So the problem can be reduced to solving the PERPEXPLORATION-BBH-HOME with 2 or 3 agents on a path graph of size n. Now it is impossible due to Theorem 6.3.12. Now let us consider the case where there is exactly one alive agent, say a_x, among the three agents exploring initially before T_f and it is at $\mathcal{C}_2$ having $|S_x(T)| \geq 2$ at T_f. So, after T_f, it is the sole duty of the agent at *home*, say a_h, to explore $\mathcal{C}_1$ perpetually. But due to argument in the last paragraph of Lemma 6.3.14, we have $|S_h(T)| \geq \lfloor \frac{n'}{4} \rfloor = n > 9$. So, it has to visit v_b at some time $T_* > T_f$. Now, if the adversary activates, v_b for all rounds after T_f, then no agents from $\mathcal{C}_2$ can cross

v_b to come to $\mathcal{C}_1$ and also a_h gets destroyed at T_*. This implies that the agents cannot solve PERPEXPLORATION-BBH-HOME.

From all the above cases, it is evident that there can not be an algorithm that solves PERPEXPLORATION-BBH-HOME with 4 agents on any path graph. □

Theorem 6.3.17. *A set of 5 agents cannot solve* PERPEXPLORATION-BBH-HOME *on a path graph of size n', where $n' > 144$.*

Proof. Let without loss of generality,the path graph $P = (V, E, \lambda)$ we assume is $L[v_0, v_{16n-1}]$ such that $n > 9$. Let v_0 be the *home*. Let k be the number of agents that stay at v_0 until T_d. Based on the values of $k \in \{1,2,3,4\}$, we have the following cases. We prove this theorem by proving these cases.

Case-I ($k = 4$): Let until T_d the number of agents staying at $home$ is 4. In this case, only one agent, say a_x, explores the path graph until T_d. Now, by a similar argument as in Case-I of Theorem 6.3.16, at T_d, the problem can be reduced to solving PERPEXPLORATION-BBH-HOME with four agents on a path graph of length at least $4n > 36$, which is impossible due to Theorem 6.3.16.

Case-II ($k = 3$): Let until T_d, the number of agents staying at $home$ is 3. Now, let a_{x_1} and a_{x_2} be the only two agents exploring the path graph $L[v_0, v_{16n-1}]$ until T_d. Now, as stated in Lemma 6.3.13, the adversary can create two scenarios which are as follows. There exists a round $T_f \geq T_d$ such that at T_f either both the agents are destroyed or exactly one, say a_{x_i} where $i \in \{1,2\}$, remains alive with $|S_{x_i}(T_f)| \geq 8n$. Also, till T_f, no agents that are staying at *home* till T_d leave *home*. Otherwise, an adversary can make them leave *home* even before T_d, which is a contradiction to our assumption that only two agents explore the path until T_d. So, for both cases, the problem now reduces to solving PERPEXPLORATION-BBH-HOME using at most 4 agents on a path of length at least $4n > 36$, which is impossible due to Theorem 6.3.16.

Case-III ($k = 2$): Let until T_d, the number of agents staying at v_0 is 2. So, there are three agents, say, a_{x_1}, a_{x_2} and a_{x_3} exploring the path graph until T_d. By Lemma 6.3.14, the adversary can create three scenarios at some time $T_f \geq T_d$. Also, for each of these scenarios, the agents staying at *home* till T_d never leave *home* even until T_f. Otherwise, by delaying

T_d, the adversary can enforce agents staying at *home* till T_d to leave *home* before T_d, which is a contradiction to our assumption that until T_d exactly two agents stay at *home*.

In the first scenario, all three agents are destroyed in at least $4n > 36$ many distinct instances denoted by $I_j = \langle L[v_0, v_{16n-1}], 5, v_0, v_j \rangle$, where $1 \leq j \leq 4n$ (by argument from the last paragraph of Lemma 6.3.14). So at T_f, the problem transforms to solving the problem PERPEXPLORATION-BBH-HOME on a path of length at least $4n > 36$ with two agents, which is impossible due to Theorem 6.3.12. Now for the second scenario, where adversary destroys at least one agent among a_{x_i} where $i \in \{1,2,3\}$, without loss of generality, let it be the agent a_{x_3}. Then, a_{x_1} and a_{x_2} stays at $\mathcal{C}_1$, where $\mathcal{C}_1$ is the component of $L[v_0, v_{16n-1}] \setminus \{v_b\}$ containing v_0. Also due to Lemma 6.3.14, $|S_{x_i}(T_f)| \geq \frac{16n}{4} = 4n > 36$, for each $i \in \{1,2\}$. So at T_f, the problem now reduces to solving PERPEXPLORATION-BBH-HOME with 3 or 4 agents on a path graph of size at least $4n > 36$, which is impossible due to Theorem 6.3.16. Now we consider the scenario three at T_f, where the adversary can create a situation where, exactly one of a_{x_1}, a_{x_2} and a_{x_3} remains alive at the component $\mathcal{C}_2$, where $\mathcal{C}_2$ is the component of $L[v_0, v_{16n-1}] \setminus \{v_b\}$ such that $v_0 \notin \mathcal{C}_2$. Let without loss of generality, a_{x_1} be the alive agent at $\mathcal{C}_2$ then as per the condition in Lemma 6.3.14, we have $|S_{x_1}(T_f)| \geq 2$ where $T_f \geq T_d$. In fact, $S_{x_1}(T_f)$ is a contiguous segment of the path graph. Also from the argument in last paragraph of Lemma 6.3.14, adversary can create $4n$ instances denoted by $I_j = \langle L[v_0, v_{16n-1}], 5, v_0, v_j \rangle$, where $1 \leq j \leq 4n$, such that for each of these $4n > 36$ instances, adversary can induce scenario 3 at some time T_f. So, after scenario 3 is induced at T_f, the two agents, say a_{x_4} and a_{x_5} at *home*, has the sole duty to explore $\mathcal{C}_1$. Note that, at T_f, agents a_{x_4} and a_{x_5}, has $|S_{x_i}(T_f)| \geq 4n > 36$ where $i \in \{4,5\}$. Now by Theorem 6.3.12, it is impossible for two agents to solve PERPEXPLORATION-BBH-HOME on a path graph of length $4n > 36$. So to able to perpetually explore, at least one of the agents, i.e., a_{x_4} or a_{x_5} must meet with a_{x_1} after T_f. Let for a_{x_1}, two possible positions of BBH are, v_p and v_{p+1}. Without loss of generality, let a_{x_1} meets a_{x_4}. Note that before meeting, if a_{x_1} reaches v_{p+1}, then for the instance $\langle L[v_0, v_{16n-1}], 5, v_0, v_{p+1} \rangle$ adversary can destroy a_{x_1} before it can meet any of a_{x_4} and a_{x_5}, thus for the remaining alive agents a_{x_4} and a_{x_5}, the problem still remains to solve PERPEXPLORATION-BBH-HOME on a path graph of size at least $4n > 36$ which is impossible due to Theorem 6.3.12. Also we claim that a_{x_4} cannot meet a_{x_1} for the first time at v_x where

$x > p+1$. This is because in this case a_{x_4} have to cross v_p, and adversary can chose from any of the two distinct instances $\langle L[v_0, v_{16n-1}], 5, v_0, v_p\rangle$ and $\langle L[v_0, v_{16n-1}], 5, v_0, v_{p+1}\rangle$ to destroy a_{x_4} in both instances and the other agent in $\mathcal{C}_1$ i.e., a_{x_5} can not distinguish between these two instances, if it is located at v_x ($x < p$) when a_{x_4} moves to v_{p+1} from v_p. Thus failing to solve PERPEXPLORATION-BBH-HOME as well. Note that when a_{x_4} moves from v_p to v_{p+1}, if at this round a_{x_5} is on v_x where $x \geq p$ then for the instance $\langle L[v_0, v_{16n-1}], 5, v_0, v_p\rangle$ adversary activates the BBH for all the upcoming rounds and all alive agents gets stranded in $\mathcal{C}_2$, again failing to solve PERPEXPLORATION-BBH-HOME. So if a_{x_4} and a_{x_1} meets, it must be at the node v_{p+1}. Let they meet first time on v_{p+1} at some round T^*. In this case at round T^*-1, a_{x_4} must be on v_p and a_{x_1} must be on v_{p+2}. Now if at T^*-1, a_{x_5} is with a_{x_4} at v_p then, adversary can chose the instance $\langle L[v_0, v_{16n-1}], 5, v_0, v_p\rangle$ and makes the BBH act as BH for the rest of the execution from round T^*-1, such that it destroys both a_{x_4} and a_{x_5} and make the only living agent a_{x_1}, get stranded at $\mathcal{C}_2$, thus making PERPEXPLORATION-BBH-HOME impossible. On the other hand, if at T^*-1, a_{x_5} is not with a_{x_4} at v_p in $\mathcal{C}_1$ then adversary can chose any of the following two instances, $I_1 = \langle L[v_0, v_{16n-1}], 5, v_0, v_p\rangle$ and $I_2 = \langle L[v_0, v_{16n-1}], 5, v_0, v_{p+1}\rangle$, and makes the BBH act as BH for the rest of the execution. In this case, the agent a_{x_5} can not distinguish between these two instances and can not meet a_{x_1} for any further help. So in this case also, PERPEXPLORATION-BBH-HOME remains impossible to solve. Next, we tackle the case where $k = 1$.

Case-IV ($k = 1$): In this case until T_d, 4 agents namely, $a_{x_1}, a_{x_2}, a_{x_3}$ and a_{x_4}, explores the path graph $L[v_0, v_{16n-1}]$. Now let only one agent, without loss of generality, say a_{x_1}, enters L_{12n}, then by similar argument as in first paragraph of Lemma 6.3.13, adversary can create $4n > 36$ instances denoted by $I_j = \langle L[v_0, v_{16n-1}], 5, v_0, v_j\rangle$ ($12n \leq j \leq 16n-1$) for each of which it can destroy a_{x_1}. Thus, the problem now reduces to solving PERPEXPLORATION-BBH-HOME by 4 agents on a path graph of length at least $4n > 36$, which is impossible due to Theorem 6.3.16.

Now let only two agents, among $a_{x_1}, a_{x_2}, a_{x_3}$ and a_{x_4} enters L_{12n}. Let these agents who enters L_{12n} be, a_{x_1} and a_{x_2}. Without loss of generality, let there exist a round T_1 when a_{x_1} is at v_{16n-1}. Let T_1' be the time when a_{x_1} was at v_{12n} for the last time before T_1. We can say that there exists a time T_1^* between T_1' and T_1 when both a_{x_1} and a_{x_2} are in $L_{12n} \subset L_{8n}$

(due to similar argument as in second and third paragraph of Lemma 6.3.14). If no other agents ever enters L_{8n} when a_{x_1} and a_{x_2} are in L_{8n+j} where $0 \leq j \leq 4n$, then we can also conclude there are rounds T_j^* such that at T_j^* both these agents a_{x_1} and a_{x_2} are in L_{8n+j} where $0 \leq j \leq 4n$. Now due to similar argument used in Lemma 6.3.13, adversary can create $4n$ distinct instances denoted as $I_\alpha = \langle L[v_0, v_{16n-1}], 5, v_0, v_{8+\alpha} \rangle$ $(0 \leq \alpha \leq 4n)$ such that it can destroy both the agents for each of these $4n > 36$ instances. So now the problem reduces to solving PERPEXPLORATION-BBH-HOME with 3 agents on a path graph of length at least $4n > 36$ which is impossible due to Theorem 6.3.12.

Now as argued in Lemma 6.3.14, there is a time when at least three agents are in L_{8n}. So, let there exist a round T_2 such that three agents, without loss of generality, say a_{x_1}, a_{x_2} and a_{x_3} are in $L_{8n} \subset L_{4n}$. Let a_{x_1}, a_{x_2} and a_{x_3} are in L_{4n+j} (for all $0 \leq j \leq 4n$), but a_{x_4} never enters L_{4n}. Now as argued in the last paragraph of Lemma 6.3.14, adversary can create $4n$ many instances denoted as $I_\beta = \langle L[v_0, v_{16n-1}], 5, v_0, v_{8n-\beta} \rangle$ $(0 \leq \beta \leq 4n)$ for which it can create scenario 1 or 3 of Lemma 6.3.14, considering each vertex of segment $[v_0, v_{4n-1}]$ as *segmented home*, which implies the agent which did not leave actual home and a_{x_4} which did not leave the segment $[v_0, v_{4n-1}]$ are at segmented home and rest of three agents left segmented home, to explore the rest of the path. Now, if scenario 1 is achieved, the problem reduces to solving PERPEXPLORATION-BBH-HOME on a path of length at least $4n > 36$ with two agents, which is impossible due to Theorem 6.3.12. If scenario 3 is achieved, then, as argued in case III of this proof, similarly, we can say that solving PERPEXPLORATION-BBH-HOME is impossible for this case too.

So, this concludes that there is a round T_3 (the existence of such a round follows by extending the arguments in Lemma 6.3.14) when each of the 4 agents, $a_{x_1}, a_{x_2}, a_{x_3}$ and a_{x_4} needs to be in $L_{4n} \subset L_1$, as in earlier case we assumed that a_{x_4} did not leave the segment $[v_0, v_{4n-1}]$. Let T_m be the first round, at which all these four agents are at L_{4n}. In this situation, the adversary can create $I_i = \langle L[v_0, v_{16n-1}], 5, v_0, v_i \rangle$, where $1 \leq i \leq 4n-1$ instances, and in each instance it activates the BBH from round T_m onwards. Now, this shows all agents except a_{x_5} (which is at *home* till T_m) gets stuck at $\mathcal{C}_2$, and it is impossible for a_{x_5} to detect the BBH in the segment $[v_1, v_{4n-1}]$.

So we conclude that for each of the cases (i.e., $k \in \{1,2,3,4\}$) it is impossible to solve

PERPEXPLORATION-BBH-HOME. So, it is impossible to solve PERPEXPLORATION-BBH-HOME with 5 agents. □

6.3.2 Description of Algorithm PATH_PERPEXPLORE-BBH-HOME

We provide an algorithm that solves PERPEXPLORATION-BBH-HOME with 6 agents, even when the size of the path is unknown to the agents. We also show how to adapt this algorithm to solve PERPEXPLORATION-BBH with 4 agents. Note that, by Theorems 6.3.12 and 6.3.17, these are the optimal numbers of agents and they cannot be reduced, even assuming knowledge of the size of the path.

We call this algorithm PATH_PERPEXPLORE-BBH-HOME. Let $\langle P, 6, h, v_b \rangle$ be an instance of PERPEXPLORATION-BBH-HOME, where $P = (V, E, \lambda)$ is a port-labeled path. As per Definition 6.2.3, 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, and we explain how to modify the algorithm to handle other cases in Remark 6.3.18. 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 G. Based on their movement, they identify their role as follows: a_0 as F, a_1 as $\texttt{INT}_2$, a_2 as $\texttt{INT}_1$ and a_3 as L. 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 move up to a distance of 2^i from h. Next, it starts exploring backwards in a similar manner, until it reaches h. It may be observed that, in any phase after $j-$th phase (where $2^j = n$), the agents behave in a similar manner, as they behaved in the $j-$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 since all the agents have $\mathcal{O}(\log n)$ bits of memory, each of them knows which phase is cur-

rently going on. Let T_i be the maximum time, required for these 4 agents (i.e., L, $\texttt{INT}_1$, $\texttt{INT}_2$ and F) to return back to h in $i-$th phase. Let us denote the waiting 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, $\texttt{INT}_1$, $\texttt{INT}_2$ 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* (refer to Section 4.2 in Chapter 4).

Now, if the first set of agents (i.e., L, $\texttt{INT}_1$, $\texttt{INT}_2$ and F) fails to return to h within T_i rounds, then that means the adversary has activated at v_b. In this case we claim that, at least one agent among L, $\texttt{INT}_1$, $\texttt{INT}_2$ and F stays alive at a node in $\mathcal{C}_2$ knowing the location of BBH, where $\mathcal{C}_2$ is the component of $P-\{v_b\}$, such that it does not contain h. Let α be such an alive agent, where $\alpha \in \{L, \texttt{INT}_1, \texttt{INT}_2, F\}$. Then, α places itself on the adjacent node of v_b in $\mathcal{C}_2$. It may be noted that α knows which phase is currently going on, and so it knows the exact round at which F_1 and F_2 start cautious move. Also, it knows the exact round at which F_1 first visits v_b, say at round r. α waits till round $r-1$, and at round r it moves to v_b. Now at round r, if the adversary activates BBH, it destroys both F_1 and α, then F_1 fails to return to F_2 in the next round. This way, F_2 knows the exact location of BBH, while it remains in $\mathcal{C}_1$. So it can explore $\mathcal{C}_1$ by itself perpetually. The right figure in Fig. 6.8, represents the case where L detects BBH 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 v_b, and at the same time L as well visits v_b from v_{j+3}. The adversary activates BBH, and both are destroyed. So, at round r_1+5, F_2 finds failure of F_1's return and understands the next node to be BBH, while it is present in $\mathcal{C}_1$. Accordingly, it perpetually explores $\mathcal{C}_1$.

On the other hand, if at round r, the adversary doesn't activate BBH, then F_1 meets with α and knows that they are located on the inactivated BBH. In this case, they move back to $\mathcal{C}_1$ and start exploring the component $\mathcal{C}_1$, avoiding BBH. The left figure in Fig. 6.8 explores this case, where L detects the position of BBH 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 v_b, L also decides to visit v_b. But, in this situation, the adversary does not activate BBH at round r_1+4, so both F_1 and L meets, gets the knowledge from L that they are on BBH. In the next round, they move to v_{j+1}, which is a node in $\mathcal{C}_1$, where they meet F_2 and share this information. After which, they

perpetually explore $\mathcal{C}_1$.

We now describe how the set of four agents, i.e., L, $\texttt{INT}_1$, $\texttt{INT}_2$, and F, create and translate the pattern to the next node.

Creating pattern: $L, \texttt{INT}_1, \texttt{INT}_2$ and F take part in this step from the very first round of any phase, starting from h. In the first round, L, $\texttt{INT}_1$ and $\texttt{INT}_2$ move to the next node. Then in the next round, only $\texttt{INT}_2$ returns to *home* to meet with F. Note that, in this configuration the agents L, $\texttt{INT}_1$, $\texttt{INT}_2$ and F are at two adjacent nodes while F and $\texttt{INT}_2$ are together and L and $\texttt{INT}_1$ are together on the same node. We call this particular configuration the *pattern*, and it is pictorially explained in Fig. 6.1.

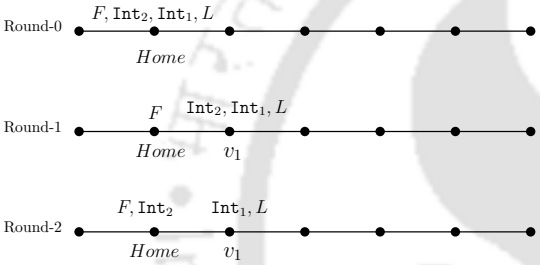


Figure 6.1: Depicts step-by-step movement, while creating the pattern from h

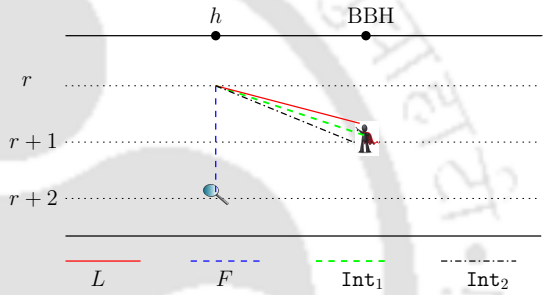


Figure 6.2: The time diagram, which depicts the case when F detects the position of the BBH while it is at h, while creating a pattern.

Note that, in the second round of creating pattern if $\texttt{INT}_2$ does not return to F, F knows that it must have been destroyed in the next node by the BBH. So F knows the exact location of the BBH and explores the component $\mathcal{C}_1$ (where $P - \{v_b\} = \mathcal{C}_1 \cup \mathcal{C}_2$, where $\mathcal{C}_1, \mathcal{C}_2 \subset P$ and $h \in \mathcal{C}_1$) perpetually, refer to Fig. 6.2.

Translating pattern: After the pattern is formed in the first two rounds of a phase, the agents translate the pattern to the next node until the agent L reaches either 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 $\texttt{INT}_1$ is on v_1 and F and $\texttt{INT}_2$ is on v_0. This translation of the pattern makes sure that after 5 consecutive rounds L and $\texttt{INT}_1$ is on v_2 and F and $\texttt{INT}_2$ 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 v_b (such that $v_b \in \{v_0, v_1, v_2\}$) is as follows.

Round 1: L moves to v_2 from v_1.

Round 2: $\texttt{INT}_2$ moves to v_1 from v_0 and L moves to v_1 back from v_2.

Round 3: $\texttt{INT}_2$ 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 $\texttt{INT}_2$ moves to v_1 from v_0 together.

Round 5: $\texttt{INT}_1$ moves to v_2 from v_1 to meet with L.

After the completion of round 5, the pattern is translated from nodes v_0 and v_1 to nodes v_1 and v_2. The pictorial description of the create and translate pattern is explained in Fig. 6.4.

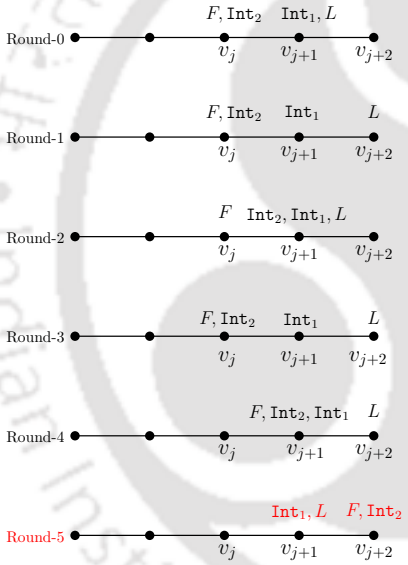


Figure 6.3: Depicts the step-wise interchange of roles, when the agents reach the end of the path graph, while performing the translate pattern

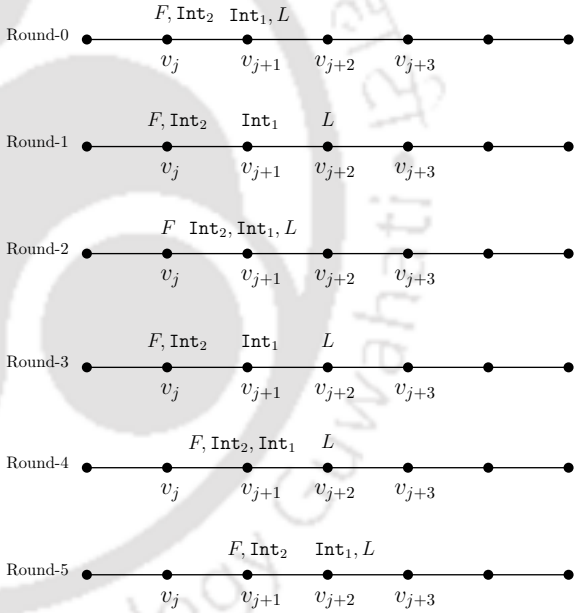


Figure 6.4: Depicts translating pattern steps

Now, suppose v_2 is the node up to which the pattern was supposed to translate at the current phase (or, it can also be the end of the path graph). 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 $\texttt{INT}_2$ and $\texttt{INT}_1$. In round 3 of the same sub-phase, F gets that information from $\texttt{INT}_2$. So, at the end of the current sub-phase, all agents have the information that they have ex-

plored either one end of the path graph or the node up to 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, $\texttt{INT}_1$ changes role to $\texttt{INT}_2$ and $\texttt{INT}_2$ changes role to $\texttt{INT}_1$ (refer to Fig. 6.3). Then, from the next sub-phase onwards, they 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 a similar manner as described above. So, at round 5 of this current sub-phase, F (previously L) and $\texttt{INT}_2$ (previously $\texttt{INT}_1$) also reaches h, and meets with L (previously F) and $\texttt{INT}_1$ (previously $\texttt{INT}_2$). We name the exact procedure as TRANSLATE_PATTERN.

Intervention by the BBH: Next, we describe the situations that can occur if the BBH destroys at least one among these 4 agents within sub-phase i in phase j, say. At the starting of sub-phase i, suppose the agents L, $\texttt{INT}_1$ is on v_1 and F, $\texttt{INT}_2$ is on v_0, at the end sub-phase i, the goal of L, $\texttt{INT}_1$ is to reach v_2 and F, $\texttt{INT}_2$ is to reach v_1. Without loss of generality, we assume here that the pattern is translating away from h in the current sub-phase.

Case-I: In this case, we look into the case when v_2 is not the endpoint for the current sub-phase.

If v_2 is BBH: We describe the cases that can arise, depending on which round within this sub-phase, BBH gets activated and destroys at least one among these 4 agents.

Let at round 1, BBH is activated for the first time in the sub-phase i. Then, in round 2 of the sub-phase i, L does not return to v_1 and meets with $\texttt{INT}_1$ and $\texttt{INT}_2$. Thus $\texttt{INT}_1$ and $\texttt{INT}_2$ knows that v_2 is BBH and they can explore $\mathcal{C}_1$ by themselves as in round 2 all of them are at $\mathcal{C}_1$ (refer to (i) of Fig. 6.5, where v_{j+2} is BBH and it symbolises v_2 in the description, also here in the figure, the sub-phase starts from round r).

If it is activated at round 2 for the first time in the sub-phase i, then no agents are destroyed, as at round 2 none of these agents are present at v_2, and they continue the next rounds of the sub-phase as usual.

If at round 3, BBH is activated for the first time in the sub-phase i, then at round 5 of sub-phase i when $\texttt{INT}_1$ reaches v_2 and it stays alive then it knows that it is on BBH as L is not there (refer to (ii) of Fig. 6.5, where L gets destroyed at round $r+3 \approx 3$ of sub-phase i,

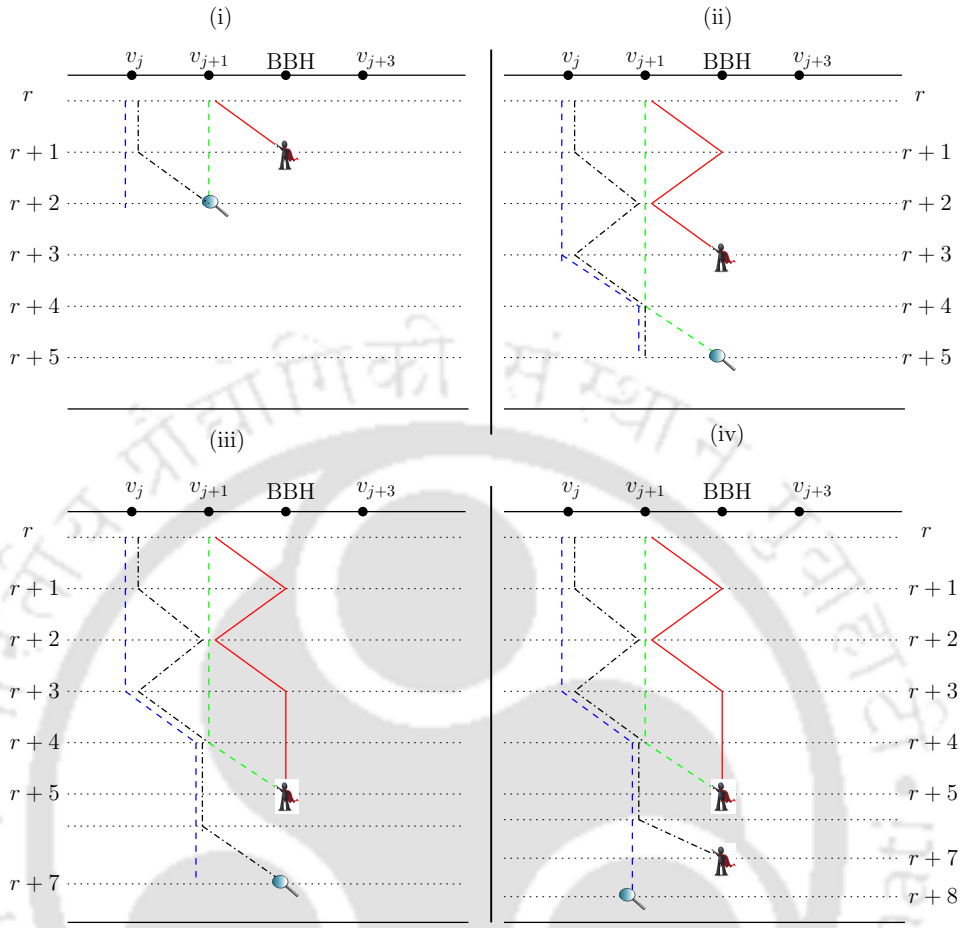


Figure 6.5: Time Diagram depicting each case, that can arise if $v_b = v_{j+2}$ intervenes, while 4 agents are translating the pattern.

and $\texttt{INT}_1$ detects it at round $r+5 \approx 5$ of sub-phase i, when it visits $BBH = v_{j+2}$). So, in this case, it moves back to $v_1 \in \mathcal{C}_1$ and starts exploring $\mathcal{C}_1$ by itself. On the other hand if at round 5 of sub-phase i, $\texttt{INT}_1$ is also destroyed at v_2 then at round 2 of sub-phase $(i+1)$, when $\texttt{INT}_2$ reaches v_2, if it stays alive, it knows that it is on BBH as both $\texttt{INT}_1$ and L are missing (refer to (iii) of Fig. 6.5, where at round $r+5 \approx 5$ of sub-phase i, $\texttt{INT}_1$ also gets destroyed, and $\texttt{INT}_2$ detects it when it visits BBH at round $r+7 \approx 2$ of sub-phase $(i+1)$). Similarly, it moves back to v_1 and starts exploring $\mathcal{C}_1$ by itself. Now, if $\texttt{INT}_1$ is also destroyed at round 2 of sub-phase $(i+1)$ then at round 3 of sub-phase $(i+1)$, F at $v_1 (\in \mathcal{C}_1)$ finds that $\texttt{INT}_2$ is missing. From this, F interprets that the BBH is at v_2, and thus it starts exploring $\mathcal{C}_1$ by itself.

Let the BBH be activated at round 4, for the first time in the sub-phase i, then it destroys

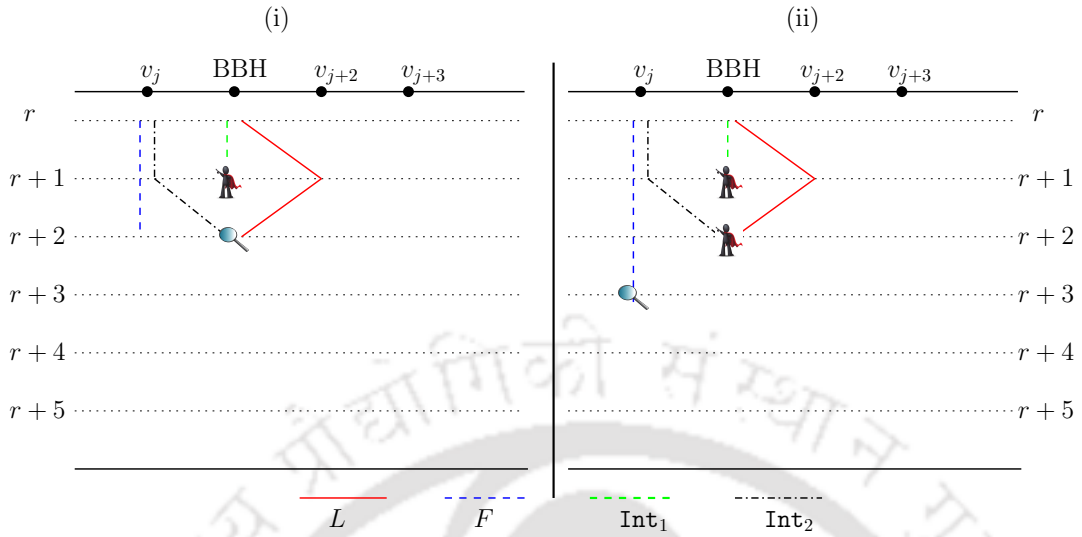


Figure 6.6: Depicts the time diagram, where $v_{j+1} = v_b$, in (i) it is activated at $r+1$ and in (ii) it is activated at rounds $r+1$ and $r+2$

L, so at round 5 of the same sub-phase, when $\mathtt{INT}_1$ moves to v_2 and it stays alive, it finds that L is missing. Hence, it knows that it is on the BBH (refer to (ii) of Fig. 6.5, except that instead of round $r+3 \approx 3$ of sub-phase i, BBH is activated at round $r+4 \approx 4$ of sub-phase i) and moves back to $v_1 \in \mathcal{C}_1$ and starts exploring $\mathcal{C}_1$ by itself.

If BBH is activated at v_2 in the sub-phase i for the first time in round 5, then it destroys both L and $\mathtt{INT}_1$ at v_2. Next, when $\mathtt{INT}_2$ moves to v_2 at round 2 of sub-phase $(i+1)$, suppose BBH is again activated (if not already activated), then $\mathtt{INT}_2$ is also destroyed at round 2 of sub-phase $(i+1)$, so it fails to return back to v_1 at the same round. So, at round 3 of sub-phase $(i+1)$, F knows that v_2 is the exact location of BBH by finding out that $\mathtt{INT}_2$ is missing at v_1 (refer to (iv) of Fig. 6.5, where after L, $\mathtt{INT}_1$ gets destroyed at round $r+5 \approx 5$ of sub-phase i, $\mathtt{INT}_2$ also gets destroyed at round $r+7 \approx 2$ of sub-phase $(i+1)$, then F understands this at round $r+8 \approx 3$ of sub-phase $(i+1)$). So, F then starts exploring $\mathcal{C}_1$ by itself.

If v_1 is BBH: Again, we describe the cases, which can arise depending on at which round within i-th sub-phase, BBH is activated, and destroys at least one among these 4 agents.

Let at round 1, the BBH is activated for the first time in the sub-phase i. Then, in round 2 of the sub-phase i, when $\mathtt{INT}_2$ and L visit v_1, and if they stay alive, they find that $\mathtt{INT}_1$ is missing. This information helps them understand that they are on the BBH (refer to (i) of Fig. 6.6 where at round $r+2 \approx 2$ of sub-phase i, L and $\mathtt{INT}_1$ detect the position of the BBH),

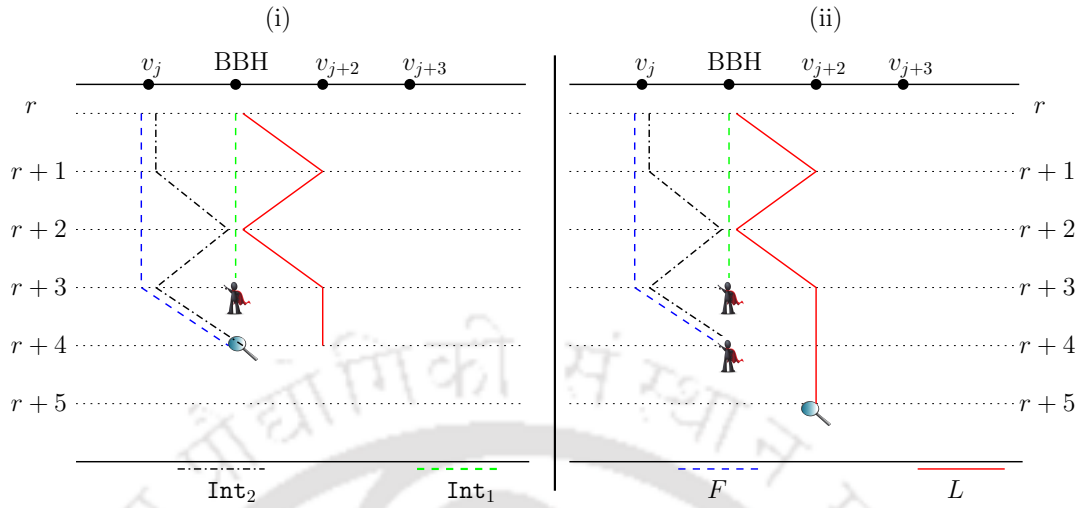


Figure 6.7: Depicts the time diagram, where $v_{j+1} = v_b$, in (i) it is activated at $r+3$ and in (ii) it is activated at rounds $r+3$ and $r+4$

so they move back to $v_0 \in \mathcal{C}_1$ and start exploring $\mathcal{C}_1$, perpetually. On the other hand, if BBH is activated at round 2, then both L and $\mathtt{INT}_2$ also get destroyed at v_1 at round 2. In this situation, at round 3, whenever F (present at $v_0 \in \mathcal{C}_1$) finds that $\mathtt{INT}_2$ has not arrived from v_1, it understands that v_1 is BBH (refer to (ii) of Fig. 6.6, where L detects the position of the BBH at round $r+3 \approx 3$ in sub-phase i, after $\mathtt{INT}_2$, L gets destroyed at round $r+2$ and $\mathtt{INT}_1$ gets destroyed at $r+1$), and explores $\mathcal{C}_1$ perpetually.

If at round 2, the BBH is activated for the first time in the sub-phase i. Then, in round 2 itself, $\mathtt{INT}_1$, $\mathtt{INT}_2$ and L gets destroyed at v_1. In this case, F present at $v_0 \in \mathcal{C}_1$, finds that $\mathtt{INT}_2$ fails to return from v_1 at round 3, hence it understands, v_1 is the BBH, and performs perpetual exploration of $\mathcal{C}_1$. Refer to (ii) of Fig. 6.6, a similar case is discussed, where all three agents L, $\mathtt{INT}_1$ and $\mathtt{INT}_2$ gets destroyed due to v_{j+1} being the BBH and that is detected by F at round $r+3 \approx 3$ in sub-phase i.

If at round 3, the BBH is activated for the first time in the sub-phase i. Then, at round 4, whenever F and $\mathtt{INT}_2$ visit v_1, and if they stay alive, then they find that $\mathtt{INT}_1$ is missing. Hence, they understand that they are on the BBH, and so they return to $v_0 \in \mathcal{C}_1$, and explore $\mathcal{C}_1$ perpetually. In (i) of Fig. 6.7, a similar case is discussed.

On the other hand, if the BBH is active at round 4 as well, then F and $\mathtt{INT}_2$ also get destroyed at v_1. In this situation, at round 5, L present at $v_2 \in \mathcal{C}_2$, finds that $\mathtt{INT}_1$ fails to

return from v_1. Hence, L understands that v_1 is the BBH (refer to (ii) of Fig. 6.7). In this situation, since $L \in \mathcal{C}_2$ hence it cannot reach h in the current phase. However, it knows the distance between h and v_1, and it also knows the exact round since the start of the current phase. Moreover, it also understands when the other two agents, i.e., F_1 and F_2, currently waiting at h, start their cautious movement, and exactly one of them reaches v_1 after it identifies the BBH uniquely. Let t_1 be that round. In this scenario, L waits until round $t_1 - 1$ and moves to v_1 at round t_1 along with exactly one agent, say F_1, from the group that was moving cautiously. Next, if the BBH is activated at round t_1, then at round $t_1 + 1$, F_2, waiting for F_1 at $v_0 \in \mathcal{C}_1$, finds that F_1 fails to return. In this situation, F_2 understands that v_1 is the BBH, and explores $\mathcal{C}_1$ perpetually (refer to (ii) of Fig. 6.8). On the other hand, if at t_1, the BBH is not activated, then at round t_1 it meets with L, gathers the knowledge that v_1 is the BBH, and returns to $v_0 \in \mathcal{C}_1$, and explores $\mathcal{C}_1$ perpetually (refer to (i) of Fig. 6.8).

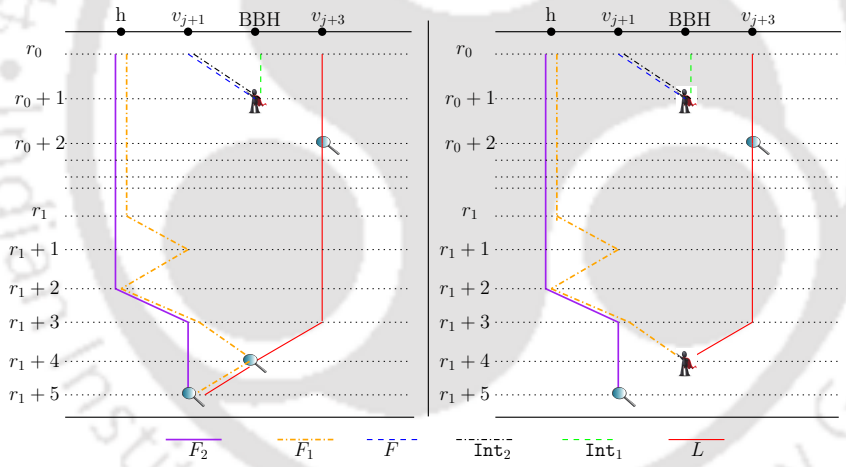


Figure 6.8: Represents the time diagram, in which at least one among F_1 and F_2 detects the BBH, and perpetually explores the path graph

In round 4, if the BBH is activated for the first time at v_1. Then, F, $\mathtt{INT}_2$ and $\mathtt{INT}_1$ each are destroyed at v_1. In this case, L present at $v_2 \in \mathcal{C}_2$ understands v_1 to be the BBH, when at round 5, it finds that $\mathtt{INT}_1$ fails to return from v_1 (refer to Fig. 6.10). By a similar argument, as described earlier, at least F_2 explores perpetually.

Now, if the BBH is activated at v_1 in the sub-phase i for the first time in round 5, then it destroys F and $\mathtt{INT}_2$. In this situation, $\mathtt{INT}_1$ and L understands this, when they are at $v_2 \in \mathcal{C}_2$ in round 2 of sub-phase $(i+1)$, after failure of $\mathtt{INT}_2$'s return from v_1 at the same round.

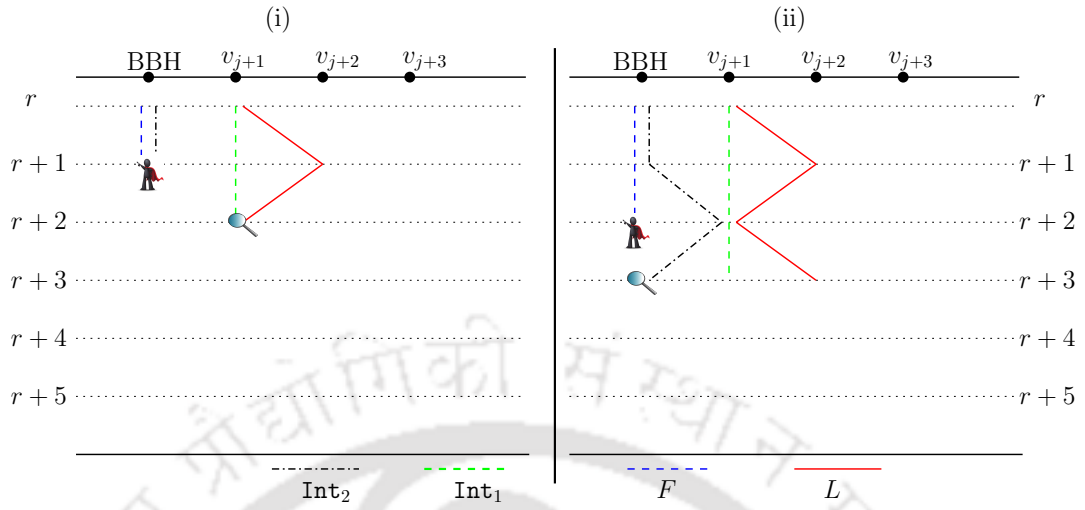


Figure 6.9: Depicts the time diagram, where $v_j = v_b$, in (i) it is activated at $r+1$ and in (ii) it is activated at round $r+2$

If v_0 is BBH: We accordingly describe the cases, which can arise depending on at which round within this sub-phase, the BBH gets activated and destroys at least one among these 4 agents.

Let at round 1, the BBH is activated for the first time in the sub-phase i. Then it destroys both F and $\texttt{INT}_2$, and this can be understood by L and $\texttt{INT}_1$ at round 2, when they find that $\texttt{INT}_2$ fails to meet them from v_0 at round 2 (refer to (i) of Fig. 6.9). By a similar argument, described for the case when v_1 is BBH, here as well, at least F_2 perpetually explores $\mathcal{C}_1$.

If the BBH is activated for the first time at round 2, then F is destroyed at v_0. Next, in round 3 of this sub-phase, whenever $\texttt{INT}_2$ visits v_0 and it is not destroyed, it finds that F is missing. Hence, it understands that v_0 is the BBH (refer to (ii) of Fig. 6.9), and moves to $v_1 \in \mathcal{C}_2$. On the contrary, if $\texttt{INT}_2$ also gets destroyed at round 3, then at round 4 of this sub-phase, whenever $\texttt{INT}_1$ finds that F and $\texttt{INT}_2$ fail to return from v_0. $\texttt{INT}_1$ understands that v_0 is the BBH (refer to Fig. 6.11), and it stays at $v_1 \in \mathcal{C}_2$, until F_1 visits v_0. Similar to the earlier argument, here as well, at least F_2 perpetually explores $\mathcal{C}_1$.

In rounds 4 and 5 of this sub-phase, if the BBH is activated for the first time, then since no agents are present on v_0, none of the agents understands that v_0 is the BBH.

Case-II: In this case, we look into the case where v_2 is the end node (i.e., either the end of the path graph, or the last node to be explored in the current phase) for the current

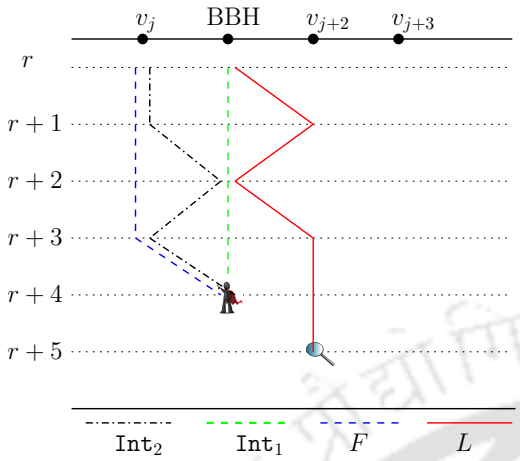


Figure 6.10: Depicts the time diagram, where $v_{j+1} = v_b$ and it is activated at round $r+4$

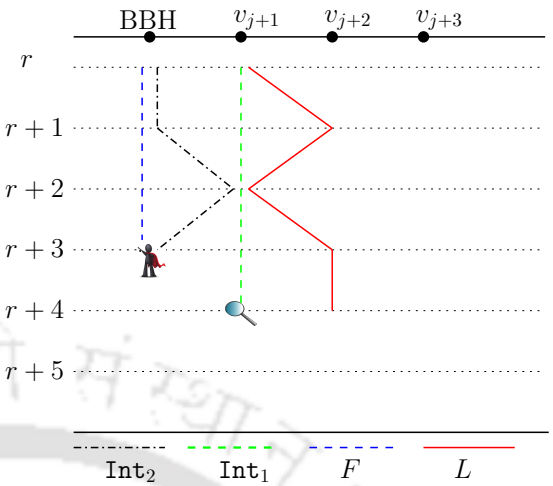


Figure 6.11: Depicts the time diagram, where $v_j = v_b$ and it is activated at round $r+3$

sub-phase. We will only discuss a few particular cases; the other cases are similar to those described in Case I.

Let us consider i to be a sub-phase, at which L first understands that v_2 is the end node, so within round 5 of this sub-phase, each agent understands this fact, and at round 5, they change their roles (refer to Fig. 6.3). Suppose v_2 is the BBH, and let the adversary activate it at round 5 of sub-phase i, while new $\texttt{INT}_2$ and F are present at v_2. This destroys both of them, and in round 2 of sub-phase $(i+1)$, the new L and $\texttt{INT}_1$ finds that new $\texttt{INT}_2$ fails to visit v_1 from v_2, hence they identify v_1 to be the BBH, and as they are present in $\mathcal{C}_1$, so they perpetually explore $\mathcal{C}_1$. Refer to (i) of Fig. 6.12, where v_{j+3} is the last node, and it is the BBH. The adversary activates the BBH at round $r+1$, which symbolises the last round of the earlier sub-phase. In this round, all alive agents change their roles, i.e., earlier F, $\texttt{INT}_2$ change to L and $\texttt{INT}_1$ at v_{j+1}. From the next round onwards, they begin executing the next sub-phase with their new roles. So, accordingly, at round $r+3$, both L and $\texttt{INT}_1$ finds that new $\texttt{INT}_2$ fails to arrive from v_{j+2}, and accordingly detect v_{j+2} to be the BBH, and continue exploring $\mathcal{C}_1$ perpetually.

On the other hand, if v_1 is selected to be the BBH, and it is activated at round 5 of sub-phase i, when they are about to change their roles, after understanding v_2 to be the end node. Hence, the new L and $\texttt{INT}_1$ gets destroyed, whereas in round 2 of sub-phase $(i+1)$,

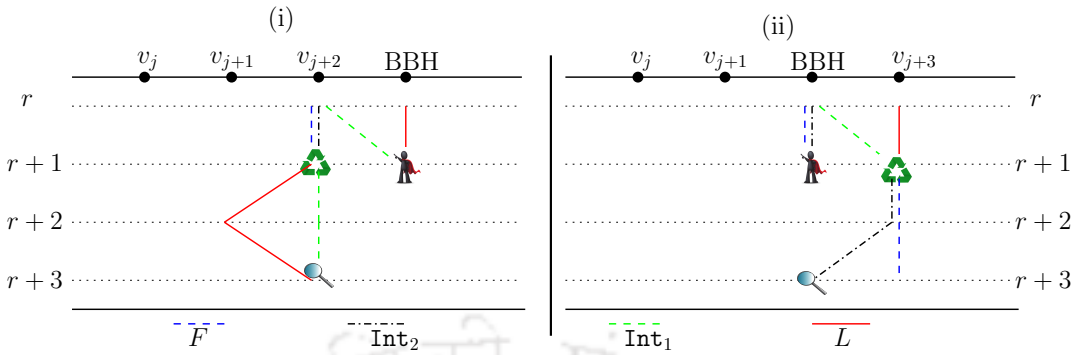


Figure 6.12: Depicts the time diagram, where in (i) $v_{j+3} = BBH$ and it is activated at round $r+1$, in (ii) $v_{j+1} = BBH$ and it is activated at round $r+1$ as well

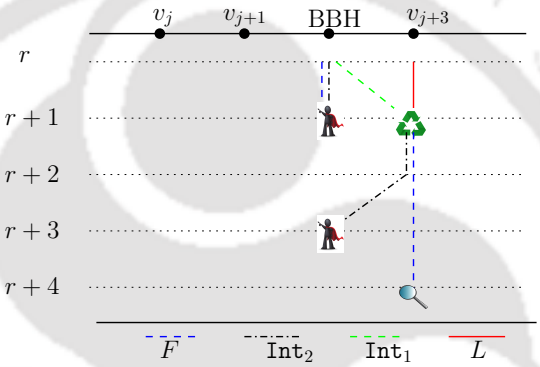


Figure 6.13: Depicts the time diagram, where $v_{j+2} = v_b$ and it is activated at rounds $r+1$ and $r+3$.

new $\mathtt{INT}_2$ visits v_1. If the BBH is not activated at this round, $\mathtt{INT}_2$ finds L and $\mathtt{INT}_1$ to be missing, hence concludes that the current node is the BBH (refer to (ii) of Fig. 6.12), and returns to v_2. Next, by a similar argument as described in Case-I, when v_1 is the BBH, eventually F_1 and F_2 start moving from h, cautiously, and at the end, at least F_2 perpetually explores $\mathcal{C}_1$. On the contrary if the BBH is activated at round 2 of sub-phase $(i+1)$, then $\mathtt{INT}_2$ gets destroyed at v_2, and this is detected by new F at v_2 at round 3 of sub-phase $(i+1)$, when it finds $\mathtt{INT}_2$ fails to arrive from v_1 (refer to Fig. 6.13). It stays at v_2 until F_1 visits v_1, and by a similar argument as described in the earlier case, at least F_2 explores $\mathcal{C}_1$ perpetually. The other situations in this case follow exactly the same pattern as those in Case I. Only difference is that, in all situations in Case-I, where the agent detects the BBH, the moment it is situated on the BBH, then it is asked to visit the previous node (as the previous node is in $\mathcal{C}_1$) and explore $\mathcal{C}_1$ perpetually, but here in Case-II, in all those situations, the agent must

visit the next node (as now the previous node belongs to $\mathcal{C}_2$ and the next node belongs to $\mathcal{C}_1$, since they changed their direction).

Remark 6.3.18. If h is not at an extreme end of the path graph P, then the first group of 4 agents choose the lowest port direction first, starting from h, at the start of each phase. They make pattern and then translate the pattern along that direction until at most 2^i distance (if the current phase is $i-$th phase), then they start returning back to h by translating the same pattern, and thereafter from h, make pattern and translate the pattern to the other direction, again until at most 2^i distance. Thereafter, using similar movement, returns to h. Only after all agents return to h, the current phase ends.

Theorem 6.3.19. *Algorithm* PATH_PERPEXPLORE-BBH-HOME *solves* PERPEXPLORATION-BBH-HOME *with 6 agents in path graphs, without knowledge of the size of the graph.*

It may be noted from the high-level idea that, in any benign execution, the first set of 4 agents perpetually explores P. Otherwise, if the BBH intervenes in the movement of the first 4 agents, either when they are making pattern or translating pattern, then two situations can occur: (1) at least one agent is alive, it knows the precise position of the BBH and it is situated in $\mathcal{C}_1$, (2) at least one agent is alive, it knows the precise position of the BBH and it is situated in $\mathcal{C}_2$.

In situation (1), since the agent is in $\mathcal{C}_1$, and it knows the position of the BBH, it can perpetually explore $\mathcal{C}_1$, so PERPEXPLORATION-BBH-HOME is achieved, since $h \in \mathcal{C}_1$. In situation (2), it has been shown in Case I and Case II earlier that at least F_2 eventually also gets to know the position of the BBH, while it is still positioned in $\mathcal{C}_1$, and thereafter it perpetually explores $\mathcal{C}_1$. In this situation as well, PERPEXPLORATION-BBH-HOME is achieved.

Corollary 6.3.20. *The algorithm consisting of* MAKE_PATTERN *and* TRANSLATE_PATTERN *solves* PERPEXPLORATION-BBH *with 4 agents in path graphs, without knowledge of the size of the graph.*

The above corollary follows from the fact that, in any benign execution, the 4 agents executing these algorithms explores P perpetually, but if the BBH intervenes, then these algorithms ensure that at least one agent remains alive either in $\mathcal{C}_1$ or $\mathcal{C}_2$, knowing the exact

position of the BBH. Hence, thereafter it can perpetually explore its current component, in turn solving PERPEXPLORATION-BBH.

6.4 Tree Network

In this section, we modify the path algorithm in Section 6.3.2, and we show how to obtain algorithm TREE_PERPEXPLORE-BBH-HOME, which solves PERPEXPLORATION-BBH-HOME with 6 agents in trees, without the agents having knowledge of the size of the tree.

Let $\langle G, 6, h, v_b \rangle$ be a PERPEXPLORATION-BBH-HOME instance, where $G = (V, E, \lambda)$ is a port-labeled tree. As per Definition 6.2.3, all agents are initially co-located at h (the *home* node). We consider, without loss of generality, the node h to be the root of G. Initially, the four least ID agents start exploring G, while the other two agents wait at h. The four agents, namely, a_0, a_1, a_2 and a_3, are termed as L, $\texttt{INT}_1$, $\texttt{INT}_2$ and F, based on their movements. The exploration they perform is exactly the same as the one explained in Section 6.3.2 (i.e., MAKE_PATTERN and then TRANSLATE_PATTERN). But unlike a path graph, where, except for the parent port (i.e., the port along which the agent has reached the current node from the previous node), only one port remains to be explored from each node, here there can be at most $\Delta - 1$ ports to choose, where Δ is the maximum degree in G. To tackle this, the agents perform a strategy similar to $k-$`Increasing-DFS` [70]. Similar to earlier algorithm on a path graph, the exploration of first four agents is divided in to phases, in the i-th phase, the agents explore at most 2^i nodes and then returns back to h, where in each phase the pattern is translated. As stated in [70], to explore a graph with diameter at most 2^i and maximum degree Δ, performing $k-$`Increasing-DFS` where $k \geq \alpha 2^i \log\Delta$ such that α is a constant, for each phase the agents require $\mathcal{O}(2^i \log\Delta)$ memory, i.e., in total $\mathcal{O}(n \log\Delta)$ bits of memory. So, within $T_i \leq 5 \cdot 2^{i+1} + 5$ rounds, if the first four agents fail to reach h, then the remaining two agents, a_4 and a_5, termed as F_1 and F_2 start moving cautiously, while executing $k-$`Increasing-DFS`, where $k \geq \alpha 2^i \log\Delta$. As per the earlier argument, if the first four agents fail to return, that implies at least one agent is alive, which knows the exact position of the BBH, and it is located in $\mathcal{C}_j$ $(1 < j \leq i)$, where $G - v_b = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_i$, such that $h \in \mathcal{C}_1$. In addition to that, since the two agents F_1 and F_2 follow exactly the same path,

as followed by the first four agents, the only alive agent, say L, not only knows the exact round at which F_1, F_2 started their movement from h, but it also knows the path they must follow. So, inevitably, it knows the exact round at which F_1 (minimum ID among F_1, F_2) is scheduled to visit v_b. At the same time, L also jumps to v_b, and as per the earlier argument in the case of a path graph, at least F_2 gets to know the position of the BBH, while it is still in $\mathcal{C}_1$, hence it can perpetually explore $\mathcal{C}_1$.

The next theorem concludes the successful solution of PERPEXPLORATION-BBH-HOME with 6 agents in trees, and the proof follows from the preceding discussion and the arguments of Section 6.3.2 for path graphs.

Theorem 6.4.1. *Algorithm* TREE_PERPEXPLORE-BBH-HOME *solves* PERPEXPLORATION-BBH-HOME *with 6 agents in tree graphs, without knowledge of the size n of the graph. Each agent needs at most* $\mathcal{O}(n \log \Delta)$ *bits of memory, where* Δ *is the maximum degree of the tree.*

The necessary condition of Theorem 6.4.1 follows from Theorem 6.3.17.

Again, at least one agent among the first four agents remains alive and eventually knows the position of the BBH if the BBH intervenes in their movement. Moreover, the alive agent (or agents) with knowledge of the BBH is either in $\mathcal{C}_1$ or $\mathcal{C}_j$, for some $j \neq 1$. Hence, it can perpetually explore $\mathcal{C}_1$ or $\mathcal{C}_j$. This concludes our next corollary.

Corollary 6.4.2. *A modification of* TREE_PERPEXPLORE-BBH-HOME *solves* PERPEXPLORATION-BBH *with 4 agents in tree graphs, without knowledge of the size n of the graph. Each agent needs at most* $\mathcal{O}(n \log \Delta)$ *bits of memory, where* Δ *is the maximum degree of the tree.*

The necessary condition of Corollary 6.4.2 follows from Theorem 6.3.12.

6.5 Arbitrary 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 6.5.1 below), which carries over directly to the problem

PERPEXPLORATION-BBH-HOME, and an algorithm for PERPEXPLORATION-BBH-HOME using $3\Delta+3$ agents (Theorem 6.5.28 below), which also solves PERPEXPLORATION-BBH.

6.5.1 Lower bound in arbitrary graphs

In this section, we construct a class of graphs $\mathscr{G}$, and we show that, given any algorithm $\mathcal{A}$ that claims to solve PERPEXPLORATION-BBH with $2(\Delta-1)$ agents, where Δ is the maximum degree of the graph, an adversarial strategy exists such that it chooses a $G \in \mathscr{G}$ on which $\mathcal{A}$ fails. Accordingly, we prove the following theorem.

Theorem 6.5.1. *For every $\Delta \geq 4$, there exists a class of graphs $\mathscr{G}$ 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 $\mathscr{G}$.*

The construction of $\mathscr{G}$ is based on incremental addition of blocks as follows.

Block-1 (Constructing Path): We define a path $\mathscr{P}$, which consists of two types of vertices. The first type of vertices are denoted by v_i, for all $i \in \{1,2,\ldots,\Delta\}$. The second type of vertices lie between v_i and v_{i+1}, for all $i \in \{1,2,\ldots,\Delta-1\}$, and they are denoted by u^i_j, for all $j \in \{1,2,\ldots,l_i\}$, where $l_i+1 = dist_{\mathscr{P}}(v_i, v_{i+1})$, such that $dist_G(a,b)$ indicates the shortest distance between two vertices a and b in G. Fig. 6.14 illustrates the path graph $\mathscr{P}$, with vertices of type v_i and u^i_j.

Figure 6.14: This figure depicts the path graph $\mathscr{P}$, constructed in Block-1 consisting of two types of vertices, where l_{i-1} depicts the $dist(v_{i-1}, v_i)-1$.

Block-2 (Attaching BBH): We partition the set $\mathscr{V} = \{v_1, v_2, \ldots, v_{\Delta-1}\}$ in to two disjoint sets $\mathscr{V}_1$ and $\mathscr{V}_2$. For each $v_i \in \mathscr{V}_1$, the BBH node is attached to v_i; this extension structure is called $\texttt{Ext}_1$. Finally, from the BBH, a node z is attached, which is further connected to $\Delta-1$ other degree 1 vertices. For each $v_i \in \mathscr{V}_2$, a node w_i is connected, which is further connected to $\Delta-2$ vertices of degree 1. Finally, each w_i is connected to the BBH; this type

of extension structure is denoted by $\texttt{Ext}_2$. If $|\mathscr{V}_1| = 0$ we have a special case[1]. An example in Fig. 6.15 is illustrated the aforementioned extension, where at least $v_1, v_{\Delta-1} \in \mathscr{V}_1$ and at least $v_2 \in \mathscr{V}_2$.

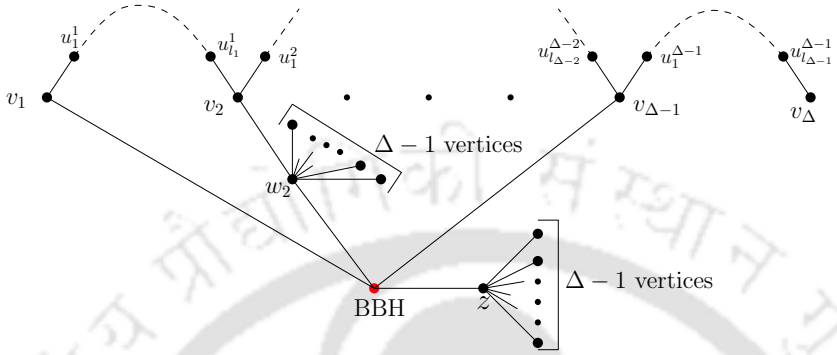


Figure 6.15: Indicates the addition of $\texttt{Ext}_1$ and $\texttt{Ext}_2$ along $\mathscr{P}$, where at least $v_1, v_{\Delta-1} \in \mathscr{V}_1$ and at least $v_2 \in \mathscr{V}_2$.

Block-3 (Attaching Trees): With respect to a vertex v, we define a tree $\mathscr{T}_v$ as follows: $\mathscr{T}_v$ is rooted at v with height 2, v has only one child, say v' and where v' has $\Delta-1$ leaves attached to it. In particular, if from v, the edge (v,v') originating from v has port i, then we call the tree $\mathscr{T}_v^i$. Next, these trees are attached from each vertex along $\mathscr{P}$ in the following manner:

1. From $v_1 \in \mathscr{P}$, $\Delta-2$ such trees are attached, only except the port leading to u_1^1 and the port leading to $\texttt{Ext}_1$ or $\texttt{Ext}_2$, depending on $v_1 \in \mathscr{V}_1$ or $\mathscr{V}_2$.

2. From each $v_i \in \mathscr{P}$, where $i \in \{2,\ldots,\Delta-1\}$, $\Delta-3$ such trees are attached, except along the ports leading to $u_{l_{i-1}}^{i-1}$, u_1^{i+1} and the one leading to $\texttt{Ext}_1$ or $\texttt{Ext}_2$, depending on $v_i \in \mathscr{V}_1$ or $\mathscr{V}_2$.

3. From each $u_j^i \in \mathscr{P}$ where $i \in \{1,2,\ldots,\Delta-1\}$ and $j \in \{1,2,\ldots,l_i\}$, $\Delta-2$ such trees are attached, except the ports leading to the previous and next node along $\mathscr{P}$.

4. If $v_\Delta \in \mathscr{V}_2$, then $\Delta-2$ such trees are attached from v_Δ, except along the ports leading to $\texttt{Ext}_2$ and $u_{l_{\Delta-1}}^{\Delta-1}$. If $v_\Delta \notin \mathscr{V}_2$, then $\Delta-1$ such trees are attached from v_Δ, except along the port leading to $u_{l_{\Delta-1}}^{\Delta-1}$.

Final Graph Class: The final graph class $\mathscr{G}$ is amalgamation of Block-1, Block-2 and Block-3. Now, based on the partition of $\mathscr{V}$, three separate graph subclass can be defined from

[1]As a special instance, abusing the definition of $\mathscr{V}$ we can include v_Δ to the set $\mathscr{V}_2$ if $|\mathscr{V}_1| = 0$. Otherwise, if $|\mathscr{V}_1| > 0$, then $v_\Delta \notin \mathscr{V}_1$ or $v_\Delta \notin \mathscr{V}_2$.

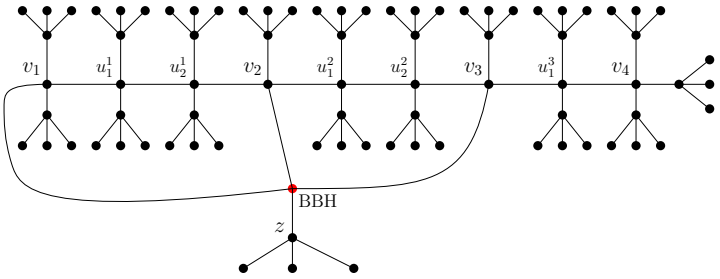


Figure 6.16: Illustrates an example graph of a member of graph class $\mathscr{G}_1$, where $\Delta = 4$, where the distance between v_1, v_2 is 2, v_2, v_3 is 2 and v_3, v_4 is 1.

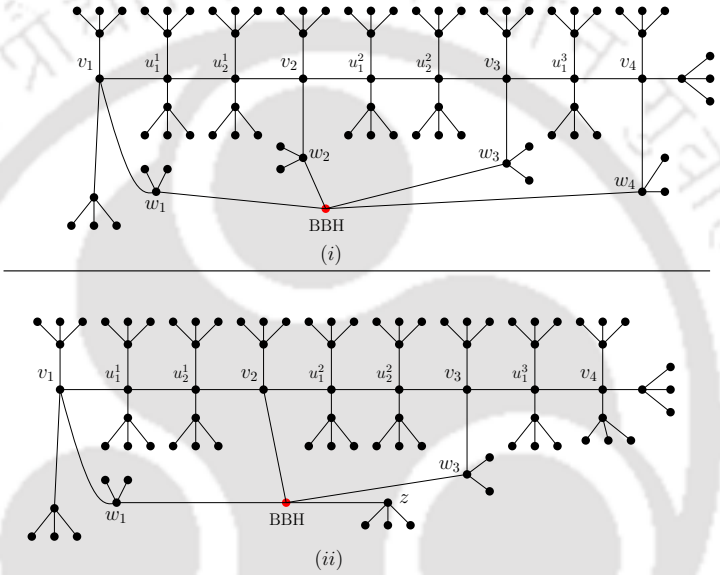


Figure 6.17: (i) Illustrates an example graph from graph class $\mathscr{G}_2$, (ii) an example graph from graph class $\mathscr{G}_3$, where $v_1, v_3 \in \mathscr{V}_2$ and $v_2 \in \mathscr{V}_1$. In both graphs the $\Delta = 4$, such that distance between v_1, v_2 is 2, v_2, v_3 is 2 and v_3, v_4 is 1.

$\mathscr{G}$, namely $\mathscr{G}_1$, $\mathscr{G}_2$ and $\mathscr{G}_3$. The graph class $\mathscr{G}_1$, preserves the characteristics that, each v_i belong to $\mathscr{V}_1$, where $i \in \{1,2,\ldots,\Delta-1\}$, refer Fig. 6.16. The graph class $\mathscr{G}_2$, preserves the characteristics that, each v_i belong to $\mathscr{V}_2$, where $i \in \{1,2,\ldots,\Delta-1\}$, refer to Fig. 6.17$(i)$. Lastly, the remaining graphs in $\mathscr{G}$ belong to $\mathscr{G}_3$, refer to Fig. 6.17(ii).

Given an algorithm $\mathcal{A}$ which claims to solve PERPEXPLORATION-BBH with $2(\Delta-1)$ agents, the adversarial counter strategy takes the graph class $\mathscr{G}$ and algorithm $\mathcal{A}$ as input, and returns a specific port-labelled graph $G \in \mathscr{G}$. We will prove that the algorithm $\mathcal{A}$ fails on G.

Before formally explaining the counter strategies used by the adversary for all choices (i.e., the instructions given to the agents during the execution), the algorithm $\mathcal{A}$ can have,

we define some of the functions and terminologies to be used by the adversary to choose a graph from $G \in \mathscr{G}$. Let V_{exp} denote the nodes in G, already explored by the collection of agents executing $\mathcal{A}$, and V^c_{exp} denote the remaining nodes to be explored, where $V = V_{exp} \cup V^c_{exp}$. The function $\lambda^1_{u,\mathcal{A}} : E_u \rightarrow \{1,\ldots,deg(u)\}$ denotes the port label ordering to be returned from each node of $u \in V \setminus \{v_1, v_2, \ldots, v_\Delta\}$, based on the algorithm $\mathcal{A}$. The function $\lambda^2_{v,\mathcal{A}} : E_v \rightarrow \{1,\ldots,deg(v)\}$ returns the port label ordering from each $v \in \{v_1, v_2, \ldots, v_\Delta\}$, based on the algorithm $\mathcal{A}$. So, $\lambda^1 = (\lambda^1_{u,\mathcal{A}})_{u \in V \setminus \{v_1,\ldots,v_\Delta\}}$ and $\lambda^2 = (\lambda^2_{v,\mathcal{A}})_{v \in \{v_1,\ldots,v_\Delta\}}$ are denoted to be the set of port label, and finally $\lambda = \lambda^1 \cup \lambda^2$ is defined to be the collection of port labels on G. Next, we define the distance function $D_{\mathcal{A}} : \{v_1, \ldots, v_\Delta\} \times \{v_1, \ldots, v_\Delta\} \rightarrow \mathbb{N}$, which assigns the graph to be chosen from $\mathscr{G}$, to have a distance of $dist_{\mathscr{P}}(v_i, v_j)$ from v_i to v_j along $\mathscr{P}$ (defined in Block-1) on the chosen graph, for $i \neq j$ and $v_i, v_j \in \{v_1, v_2, \ldots, v_\Delta\}$.

The adversarial counter strategy works as follows: v_1 is set as *home* for any graph chosen from $\mathscr{G}$, i.e., all the agents are initially co-located at *home*. Next, we discuss all possible choices $\mathcal{A}$ can use from v_1, then from v_2 and finally from v_i (where $i \in \{2,\ldots,\Delta-1\}$) one after the other. Accordingly, we state the counter strategies of the adversary to choose a graph from $\mathscr{G}$, after each such choice. We start with *home* (i.e., v_1).

Choices of algorithm $\mathcal{A}$ and counter strategies of adversary at *home* : The agents are initially co-located at v_1, since it is designated as *home*. Based on the choices that an algorithm can have from v_1, we explain the countermeasures taken by the adversary to choose a certain graph from $\mathscr{G}$ as per $\mathcal{A}$, accordingly.

Choice-1: If at the first step, $\mathcal{A}$ assigns at least 2 agents to perform the first movement, and suppose it is along a port j from v_1, where $j \in \{1,\ldots,deg(v_1)\}$.

Counter: The adversary chooses a graph in which $v_1 \in \mathscr{V}_1$. Moreover, the port-labeled function at v_1, $\lambda^2_{v_1,\mathcal{A}}$, returns an ordering where the port from v_1 along the edge (v_1, BBH) is j.

Choice-2: As per the execution of $\mathcal{A}$, let r_1 (for some $r_1 > 0$) be the first round, at which any one agent from v_1 travels a node, which is at a 2-hop distance.

Counter: If there exists a round $r'_1 < r_1$ at which more than one agent visits a neighbor node of v_1 with respect to some port j, then in that scenario, adversary selects $v_1 \in \mathscr{V}_1$

and returns $\lambda^2_{v_1,\mathcal{A}}$ such that v_1 is connected to the BBH via port j. On the other hand, if there does not exist such a round r'_1, then choose $v_1 \in \mathscr{V}_1$ or $v_1 \in \mathscr{V}_2$, if $v_1 \in \mathscr{V}_1$ then return $\lambda^2_{v_1,\mathcal{A}}$ and $\lambda^1_{BBH,\mathcal{A}}$ such that at $r_1 - 1$ and r_1 rounds, the single agent must be at the BBH and z, respectively. If $v_1 \in \mathscr{V}_2$ then return $\lambda^2_{v_1,\mathcal{A}}$ and $\lambda^1_{w_1,\mathcal{A}}$ (where w_1 is part of $\texttt{Ext}_2$, refer explanation in Block-2) such that at rounds $r_1 - 1$ and r_1, the single agent must be at w_1 and the BBH, respectively.

Lemma 6.5.2. *As per Counter of Choice-2, if there exists no $r'_1 < r_1$, then the adversary can activate the BBH, such that even after the destruction of one agent within round r_1, the remaining alive agents cannot know the exact node of the BBH from v_1.*

Proof. Consider two instances, first $v_1 \in \mathscr{V}_1$ and second $v_1 \in \mathscr{V}_2$. In both instances, the adversary activates the BBH at rounds $r_1 - 1$ and r_1, respectively. Now, since no new information is gained by the agent till round $r_1 - 1$ considering the fact that $deg(BBH) = deg(z)$ if $v_1 \in \mathscr{V}_1$ and $deg(BBH) = deg(w_1) = \Delta$, where w_1 is part of $\texttt{Ext}_2$ if $v_1 \in \mathscr{V}_2$. So this means the agent will invariably move to the BBH from w_1, if $v_1 \in \mathscr{V}_2$ and to z from the BBH, if $v_1 \in \mathscr{V}_2$ at round r_1. This shows that, even after an agent is destroyed, the remaining agents cannot know which among the two nodes along this 2-length path from v_1 is indeed the BBH. □

According to the above lemma, it is demonstrated that, even after the destruction of one agent from v_1 by the BBH, the exact location of the BBH relative to v_1 cannot be determined. But, since $\mathcal{A}$ aims to solve PERPEXPLORATION-BBH, $\mathcal{A}$ needs to destroy at least one more agent from v_1 in order to detect the exact position of the BBH from v_1. Let that round at which the second agent gets destroyed by the BBH from v_1 be r''_1. So, $[r_1, r''_1]$ signifies the total number of rounds between the destruction of the first agent and the second agent along v_1. Finally, after all the possible choices that can arise from v_1 as per execution of $\mathcal{A}$, and its respective countermeasures of the adversary, the possible choices of graph class reduce to $\mathscr{G}^1$, where $\mathscr{G}^1 \subset \mathscr{G}$.

Theorem 6.5.3. *At least 2 agents are destroyed by the BBH from v_1.*

The proof of the above theorem follows from the Counter of Choice-1 and the conclusion of Lemma 6.5.2, i.e., in other words, it is shown that for any choice $\mathcal{A}$ taken from v_1,

at least 2 agents are destroyed by the BBH, within 2 hops of v_1. Next, from v_1 the agents eventually reach v_2, while executing $\mathcal{A}$. Based on the choices that $\mathcal{A}$ can have from v_2, we discuss the counter strategies of the adversary in the next section.

Choices of algorithm $\mathcal{A}$ and counter strategies of adversary at v_2: We explain the choices that an algorithm $\mathcal{A}$ can have while exploring new nodes from v_2; accordingly, we present the adversarial counters. The following lemma discusses the fact that, after reaching v_2 for the first time, any agent trying to explore V^c_{exp} from v_2 needs to traverse at least 2 hops from v_2.

Lemma 6.5.4. *In order to explore V^c_{exp} from v_2, $\mathcal{A}$ must instruct at least one agent to travel at least 2 hops from v_2.*

Proof. Any graph in which a node of the form v_2 exists, as per the construction of $\mathscr{G}$, there exist vertices at least 2 hop distance apart, which are only reachable through v_2. This implies that, if no agent from v_2 visits a node which is at 2 hop distance, then there will exist some vertices which will never be explored by any agent, in turn contradicting our claim that $\mathcal{A}$ solves PERPEXPLORATION-BBH. □

First, we discuss all possible knowledge that the set of agents can acquire, before reaching v_2 from v_1. Let at least one agent reach v_2 at round r_2° for the first time from v_1, and r_2 ($> r_2^\circ$) is the first round when at least one agent is destroyed from v_2 within at most 2 hops of v_2. The agents can gain the map of the set V_{exp}, explored so far. Define $r_1'' - r_1 = t_1$, $r_2 - r_2^\circ = Wait_2$ and $t_1 - Wait_2 = Time_1$. Next, we define the concept of *Conflict-Free* for any node $v_i \in \mathscr{V}$.

Definition 6.5.5 (Conflict-Free)**.** A node $v_i \in G$ (for some $G \in \mathscr{G}$) is said to be Conflict-Free, if as per the execution of $\mathcal{A}$, any agent first visits v_i at round r_i°, and round r_i ($r_i > r_i^\circ$) be the first round after r_i°, at which at least one agent gets destroyed by the BBH within at most 2 hop distance of v_i, while moving from v_i, then the adversary must ensure the following condition:

- Within the interval $[r_i^\circ, r_i]$, no agent from v_j, for all $j \in \{1, 2, \ldots, i-1\}$, tries to visit along $\texttt{Ext}_1$ if $v_j \in \mathscr{V}_1$ or $\texttt{Ext}_2$ if $v_j \in \mathscr{V}_2$.

To ensure, v_2 to be Conflict-Free, the adversary sets the distance between v_1 and v_2 along $\mathscr{P}$ as follows: if $Time_1 > 0$, and there exists an $l_1 \in \mathbb{N}$, such that $c_1 \cdot \Delta^{l_1+1} < Time_1$, where c_1 (is a constant) is the maximum round for which each alive agent remains stationary while moving from v_1 towards v_2, then return $D_{\mathcal{A}}(v_1, v_2) = l_1 + 1$, and modify the port-labeling of v_1 using $\lambda^2_{v_1,\mathcal{A}}$, and insert the port label of u^1_i using $\lambda^1_{u^1_i,\mathcal{A}}$ (for all $i \in \{1,2,\ldots,l_1\}$) such that the first agent's l_1 distance movement from v_1 after round r_1 is to v_2. Otherwise, return $D_{\mathcal{A}}(v_1, v_2) = r''_1 + 1$. Call $D_{\mathcal{A}}(v_1, v_2) = l_1 + 1$. Again, modify the port-labeling of v_1 using $\lambda^2_{v_1,\mathcal{A}}$, and insert the port label of u^1_i using $\lambda^1_{u^1_i,\mathcal{A}}$ (for all $i \in \{1,2,\ldots,l_1\}$) such that the first agent's l_1 distance movement from v_1 after round r''_1 is to v_2. So, this means the graph to be chosen from $\mathscr{G}$ must have $u^1_1,\ldots,u^1_{l_1}$ many nodes between v_1 and v_2. In the following lemma, we prove that v_2 is Conflict-Free.

Lemma 6.5.6. *v_2 is Conflict-Free.*

Proof. Without loss of generality, a single agent performed the first 2-hop distance movement from v_1, and got destroyed at round r_1. As per the conclusion of Lemma 6.5.2, at least one other also gets destroyed through v_1 at round r''_1, in order to determine the exact position of the BBH. If two agents first performed a movement from v_1, in that case $r_1 = r''_1$. Now, $[r_1 + 1, r''_1 - 1]$ is the interval within which no agent from v_1 tried to visit two possible positions of the BBH from v_1, and moreover, the adversary acts in such a way that, at round r''_1+1 onwards, each agent trying to visit from v_1 knows the exact position of the BBH. As per the construction of $\mathscr{G}$, to reach v_2 from v_1, except using the nodes connected to v_1 via $\texttt{Ext}_1$ or $\texttt{Ext}_2$, any agent takes at most $c_1 \cdot \Delta^{l_1+1}$ rounds, where c_1 signifies the maximum number of rounds for which each alive agent remains stationary during their movement from v_1 towards v_2. Next, $Time_1$ calculates the number of rounds remaining after subtracting $t_1 = r''_1 - r_1$ (i.e., the number of rounds between the first and second agent destruction from v_1) with $Wait_2 = r_2 - r^\circ_2$ (i.e., the number of rounds between an agent first arrives at v_2 and an agent gets destroyed by the BBH from v_2). There are two conditions for choosing the graph:

Condition-1: If there exists an $l_1 \in \mathbb{N}$ such that it satisfies the condition $c_1 \cdot \Delta^{l_1+1} < Time_1$, then the adversary returns $D_{\mathcal{A}}(v_1, v_2) = l_1 + 1$, and the adversary chooses a graph

from $\mathscr{G}$, such that $dist_{\mathscr{P}}(v_1, v_2) = l_1 + 1$. Also, it performs the port labeling at the nodes $v_1, u_1^1, \ldots, u_{l_1}^1$, in a way that, whenever an agent, executing $\mathcal{A}$, visits a neighbour at a distance l_1 for the first time from v_1, then it reaches the node v_2. Set this round as r_2°. In the worst case, $r_2^\circ = r_1 + c_1 \cdot \Delta^{l_1+1}$. It may be noted that, since $Time_1 = t_1 - Wait_2 > c_1 \cdot \Delta^{l+1}$, this implies $t_1 > Wait_2 + c_1 \cdot \Delta^{l+1}$, which also implies $r_1'' > r_1 + r_2 - r_2^\circ + c_1 \cdot \Delta^{l+1}$. This signifies that during the interval $[r_2^\circ, r_2]$ no agent from v_1 tries to locate the BBH, hence v_2 is Conflict-Free.

Condition-2: Otherwise, if there does not exist such l_1 to satisfy the above condition, then the adversary returns $D_{\mathcal{A}}(v_1, v_2) = r_1'' + 1$, we call it $l_1 + 1$. In this situation, the adversary activates the BBH in such a way that, any agent from v_1 knows the exact position of the BBH from round $r_1'' + 1$ onwards, and we assume that, further no agent from v_1 tries to visit the BBH along $\texttt{Ext}_1$ or $\texttt{Ext}_2$ (if it does, then a simple modification of $D_{\mathcal{A}}(v_1, v_2)$ will ensure further that, again v_2 is Conflict-Free). This shows that, since $r_2^\circ \geq r_1'' + 1$, so it implies that v_2 is Conflict-Free. □

Next, since, $BBH \in V_{exp}$ (as per Choice-2 from the node v_1, if $v_1 \in \mathscr{V}_1$), so as part of the map of V_{exp}, the agents can know the port labeling of the edge (v_1, BBH). Based on these, $\mathcal{A}$ can have two classes of choices: the agents use the knowledge of the port label of (v_1, BBH), and second, they do not. We call these first class of choices as *Choice-A* class and second as *Choice-B* class. We discuss all possible choices in *Choice-A* class first.

Choice-A1: Algorithm $\mathcal{A}$ may instruct at least 2 agents to explore a neighbor of V_{exp}^c at round r_2 along port j, where $r_2 > r_2^\circ$ and $j \in \{1, 2, \ldots, deg(v_2)\}$.

Counter: The adversary sets $v_2 \in \mathscr{V}_1$, and accordingly it chooses a graph in $\mathscr{G}$ satisfying this criteria. Moreover, the port-labeled function at v_2, i.e., $\lambda^2_{v_2,\mathcal{A}}$, returns an ordering where the port from v_2 towards the edge (v_2, BBH) is j.

Choice-A2: As per the execution of $\mathcal{A}$, let r_2 ($> r_2^\circ$) be the first round, at which a single agent travels a node which is at 2 hop distance of v_2 in V_{exp}^c.

Counter: If there exists a round r_2', such that $r_2^\circ < r_2' < r_2$, at which more than one agent tries to visit a neighbor of v_2 in V_{exp}^c with respect to some port j, then in that scenario, adversary selects $v_2 \in \mathscr{V}_1$ and returns $\lambda^2_{v_2,\mathcal{A}}$ such that v_2 is connected to the BBH via port j.

On the other hand, if there does not exist such r_2', then choose $v_2 \in \mathscr{V}_1$ or $v_2 \in \mathscr{V}_2$. If $v_2 \in \mathscr{V}_1$, then return $\lambda^2_{v_2,\mathcal{A}}$ and modify $\lambda^1_{BBH,\mathcal{A}}$ such that at rounds r_2-1 and r_2, the agent is at the BBH and at any neighbor of the BBH except v_2. If $v_2 \in \mathscr{V}_2$, then return $\lambda^2_{v_2,\mathcal{A}}$ such that at round r_2-1 the agent is at w_2, and return $\lambda^1_{w_2,\mathcal{A}}$ such that the port label of w_2 to the BBH is exactly same as the port label of the BBH to v_1.

Lemma 6.5.7. *If $\mathcal{A}$ instructs at least two agents to move simultaneously to a neighbor of v_2 in V^c_{exp}, then the adversary can destroy all of these agents, even if $\mathcal{A}$ uses the knowledge of the map of V_{exp} during its execution from v_2.*

The proof of the above lemma is simple, as after the agent's reach v_2 from v_1 at round r_2°, if $\mathcal{A}$ decides to send at least two agents from v_2 along port j, where the j-th port does not belong to the current map, in that case, the adversary can choose $v_2 \in \mathscr{V}_1$ and set the port connecting v_2 to the BBH as j. This shows that even if the algorithm $\mathcal{A}$ uses the map of V_{exp}, the adversary is still able to destroy all the agents, which are instructed to move from v_2 after round r_2° for the first time.

Lemma 6.5.8. *As per Counter of Choice-A2, if there exists no r_2', such that $r_2^\circ < r_2' < r_2$, then the adversary can activate the BBH, such that after destruction of one agent from v_2 within round r_2, remaining agents cannot know the exact node from v_2 to the BBH.*

Proof. Since the Counter is part of the Choice-A class, $\mathcal{A}$ uses the knowledge of the port from the BBH to v_1, while exploring 2 hops from v_2 for the first time after round r_2°. Let that port connecting the BBH to v_1 be ρ (where $\rho \in \{1,\ldots,\Delta\}$). First, it must be noted that the distance from v_1 to v_2 through the BBH along an induced subgraph of G, defined by $H = (V \setminus \{u^1_1,\ldots,u^1_{l_1}\}, E)$ where $G \in \mathscr{G}$ is at least 2. So, in order to reach v_1 from v_2, any agent, say a_i, needs to visit a node which is at least 2 hop distance from v_2 on H. So, to make use of the knowledge of the port from the BBH to v_1, the algorithm $\mathcal{A}$ can use only the following strategy. $\mathcal{A}$ uses the gained knowledge, places some agent at v_1, and instructs a_i to use the port ρ at r_2-1 rounds, since at round r_2 any agent from v_2 visits a node which is at 2 hop distance, for the first time after r_2°.

In this strategy, as per our Counter of Choice-A2, the adversary can create two instances, first $v_2 \in \mathscr{V}_1$ (refer to Instance-1 in Fig. 6.18), and activate the BBH at round r_2-1.

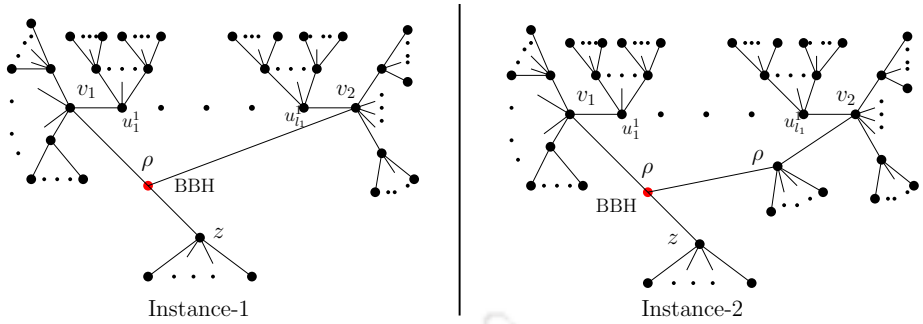


Figure 6.18: Illustrates the two instances, which the adversary can create in *Counter* of *Choice-A2*.

In the second instance, $v_2 \in \mathscr{V}_2$, where the port number from w_2 to the BBH is ρ (refer to Instance-2 in Fig. 6.18), and activates the BBH at round r_2. Since both the BBH and w_2 are of degree Δ, the agent a_i till round $r_2 - 1$ does not gain any different knowledge. Hence, in the first instance, it gets destroyed at $r_2 - 1$ and in the second instance gets destroyed at r_2. The remaining alive agents (including the one waiting at v_1) do not gain any different knowledge, as in both instances a_i fails to reach v_1 from round r_2 onwards. So, the remaining agents cannot determine the location of the BBH from v_2. □

Next, we discuss the choices in the class *Choice-B*, i.e., in which the agent does not use the knowledge of the port label from the BBH to v_1.

Choice-B1: $\mathcal{A}$ may instruct at least 2 agents to explore a neighbor of V^c_{exp} at round r_2 along port j, where $r_2 > r_2^\circ$ and $j \in \{1, 2, \ldots, deg(v_2) - 1\}$.

Counter: Similar counter as the one described in Choice-A1 for the node v_2.

Choice-B2: As per the execution of $\mathcal{A}$, let r_2 be the first round, at which a single agent travels to a node which is at 2 hop distance of v_2 in V^c_{exp}.

Counter: If there exists a round r_2', such that $r_2^\circ < r_2' < r_2$, at which more than one agent visits a neighbor of v_2 with respect to some port j, then in that scenario, adversary selects $v_2 \in \mathscr{V}_1$ and returns $\lambda^2_{v_2,\mathcal{A}}$ such that v_2 is connected to the BBH via port j. If there does not exist such r_2', then choose $v_2 \in \mathscr{V}_1$ or $v_2 \in \mathscr{V}_2$. If $v_2 \in \mathscr{V}_1$, then return $\lambda^2_{v_2,\mathcal{A}}$ and modify $\lambda^1_{BBH,\mathcal{A}}$ such that at rounds $r_2 - 1$ and r_2, the agent is at the BBH and at any neighbor of the BBH, except v_2. If $v_2 \in \mathscr{V}_2$, then return $\lambda^2_{v_2,\mathcal{A}}$ and $\lambda^1_{w_2,\mathcal{A}}$ such that at rounds $r_2 - 1$ and r_2, the agent is at w_2 and at the BBH.

Lemma 6.5.9. *As per Counter of Choice-B2, if there exists no r_2', then the adversary can activate the BBH, such that after the destruction of one agent from v_2 within round r_2, remaining agents cannot know the exact node from v_2 to the BBH.*

The idea of the proof is similar to Lemma 6.5.2. Moreover, we can similarly conclude that, at least one more agent must be destroyed by the BBH from v_2 irrespective of the knowledge gained by the agents while traversing from v_1 (refer to Lemmas 6.5.7, 6.5.8 and 6.5.9), and that round is denoted as r_2''.

Corollary 6.5.10. *Given any graph from $\mathscr{G}$, at round $r_2^\circ - 1$, the set V_{exp}, can only contain the nodes v_1, u_j^1 (for all $j \in \{1,2,\ldots,l_1\}$) along $\mathscr{P}$ of $G \in \mathscr{G}$.*

The above corollary is a direct consequence of Lemma 6.5.2 and Lemma 6.5.6, as any path to a node in V_{exp}^c from v_1 must either pass through the BBH or through v_2. Since v_2 is conflict-free, that implies till round r_2°, either one agent is destroyed at the BBH from v_1, or the adversary can activate the BBH in such a way that at least 2 agents have been destroyed and no more agent can cross the BBH without being destroyed. Moreover, r_2° indicates the first round any agent moves from v_1 to v_2. This proves the corollary, claiming V_{exp} can contain only the nodes v_1, u_j^1 (for all $j \in \{1,2,\ldots,l_1\}$) along $\mathscr{P}$ of $G \in \mathscr{G}$.

Theorem 6.5.11. *At least 2 agents are destroyed by the BBH from v_2.*

Proof. The adversary chooses $G \in \mathscr{G}$, such that v_2 is Conflict-Free. This implies, the adversary does not require to activate the BBH between the interval $[r_2^\circ, r_2]$ due to any agent's movement from v_1, towards $\texttt{Ext}_1$ or $\texttt{Ext}_2$ (depending whether $v_1 \in \mathscr{V}_1$ or $v_1 \in \mathscr{V}_2$). Next, as per Lemmas 6.5.7, 6.5.8 and 6.5.9, shows that irrespective of the knowledge gained by the agents, there exists a round r_2'' ($\geq r_2$) at which the second agent gets destroyed from v_2, while traversing along $\texttt{Ext}_1$ or $\texttt{Ext}_2$ from v_2 (depending on $v_2 \in \mathscr{V}_1$ or $v_2 \in \mathscr{V}_2$). Hence, this proves the theorem. □

Finally, after all possible choices that can arise from v_2 as per execution of $\mathcal{A}$, and its respective counter measures of the adversary, the possible choices of the graphs reduces to $\mathscr{G}^2$, where $\mathscr{G}^2 \subset \mathscr{G}^1 \subset \mathscr{G}$. Next, in general, we discuss the choices $\mathcal{A}$ can have after reaching

v_i (where $i \in \{3,\ldots,\Delta-1\}$), and accordingly discuss the adversarial counters in the following section.

Choices of algorithm $\mathcal{A}$ and counter strategies of adversary at v_i, for $2 < i \leq \Delta - 1$: In this case as well, we explain the choices that an algorithm $\mathcal{A}$ can have while exploring new nodes from v_i; accordingly, we present the adversarial counters.

Lemma 6.5.12. *In order to explore V^c_{exp} from v_i, $\mathcal{A}$ must instruct at least one agent to travel at least 2 hops from v_i.*

The idea of the proof of the above lemma is similar to Lemma 6.5.4. Next, we discuss the possible knowledge the agents might have gained when any agent visits v_i for the first time.

Let the round at which at least one agent reaches v_i for the first time be r°_i, and r_i ($> r^\circ_i$) indicates the round at which at least one agent gets destroyed from v_i within 2 hops of v_i. The agent can gain the map of the set V_{exp}, explored yet. Define $T^{max}_{i-1} = \max_{j=1}^{i-1}(r''_j - r_j)$, $r_i - r^\circ_i = Wait_i$ and $T^{max}_{i-1} - Wait_i = Time_{i-1}$.

To ensure v_i to be Conflict-Free, the adversary sets the distance between v_{i-1} and v_i along $\mathscr{P}$ as follows: if $Time_{i-1} > 0$, and there exists some $l_{i-1} \in \mathbb{N}$ such that $C'_{i-1} \cdot \Delta^{l_{i-1}+1} < Time_{i-1}$, where $C'_{i-1} = \max\{c_1, c_2, \ldots, c_{i-2}\}$, then return $D_{\mathcal{A}}(v_{i-1}, v_i) = l_{i-1} + 1$ and modify the port-labeling of v_{i-1} using $\lambda^2_{v_{i-1},\mathcal{A}}$, and port label u^{i-1}_j using $\lambda^1_{u^{i-1}_j,\mathcal{A}}$ (for all $j \in \{1,2,\ldots,l_{i-1}\}$) such that first agent's l_j distance movement from v_j on V^c_{exp}, after round r_{j-1} is to v_j. Otherwise, return $D_{\mathcal{A}}(v_{i-1}, v_i) = r''_{i-1} + 1$, call it $l_{i-1} + 1$. Also modify the port-labeling of v_{i-1} using $\lambda^2_{v_{i-1},\mathcal{A}}$, and port label of $\lambda^1_{u^{i-1}_j,\mathcal{A}}$ (for all $j \in \{1,2,\ldots,l_{i-1}\}$) such that the first agent's l_{i-1} distance movement after round r''_{i-1} is to v_i. In the following lemma, we have shown that v_i is indeed Conflict-Free.

Lemma 6.5.13. *v_i is Conflict-Free.*

Again, the proof of this is similar to the one discussed in Lemma 6.5.6.

Next, for all $v_j \in \mathscr{V}_1$ where $j \in \{1,2,\ldots,i-1\}$, the agent can know the exact port labelings of the edge (v_j, BBH). Based on these, we have again two classes of choices: first, the

agents use the knowledge of the port labels (v_i, BBH) for all such j, and second, they do not. We call the first class of choices *Choice-A* and the second class of *Choice-B*. We discuss all the possible choices in *Choice-A* class first.

Choice-A1: $\mathcal{A}$ may instruct at least 2 agents to explore a neighbor of v_i from V^c_{exp} at round r_i along port j, where $r_i > r_i^\circ$ and $j \in \{1,2,\ldots,deg(v_i)\}$.

Counter: The counter idea is the same as the one explained in Choice-A1 for v_2.

Choice-A2: Let r_i be the first round, at which a single agent travels a node which is at 2 hop distance of v_i belonging to V^c_{exp}.

Counter: The counter idea is similar to the one explained in Choice-A2 for v_2.

Lemma 6.5.14. *If $\mathcal{A}$ instructs at least two agents to move simultaneously to a neighbor of v_2 in V^c_{exp}, then the adversary can destroy all of these agents, even if $\mathcal{A}$ uses the knowledge of the map of V_{exp} during its execution from v_2.*

The proof of the above lemma is similar to Lemma 6.5.7.

Lemma 6.5.15. *As per Counter of Choice-A2, if there exists no r'_i, where $r_i^\circ < r'_i < r_i$, then the adversary can activate the BBH, such that after the destruction of one agent from v_i within round r_i, remaining agents cannot know the exact node from v_i to the BBH.*

The proof of the above lemma is similar to Lemma 6.5.15.

Next, we discuss the choices in the class *Choice-B*, i.e., when the agents do not use the knowledge of the port label from the BBH to v_j, for any $j \in \{1,2,\ldots,i-1\}$.

Choice-B1: $\mathcal{A}$ may instruct at least 2 agents to explore a neighbor of v_i from V^c_{exp} at round r_i along port j, where $r_i > r_i^\circ$ and $j \in \{1,2,\ldots,deg(v_i)\}$.

Counter: The counter idea is similar to the one described for Choice-B1 of v_2.

Choice-B2: Let r_i be the first round at which a single agent travels a node which is at 2 hop distance of v_i in V^c_{exp}.

Counter: The counter idea is similar to the one described for Choice-B2 of v_2.

Lemma 6.5.16. *As per Counter of Choice-B2, if there exists no $r'_i < r_i$, then the adversary can activate the BBH, such that after the destruction of one agent from v_i within round r_i, the remaining agents cannot know the exact node from v_2 which is the BBH.*

The proof of the lemma is similar to Lemma 6.5.9.

Corollary 6.5.17. *Given any graph $\mathcal{G}$ at round $r_i^\circ - 1$, the set V_{exp} can only contain the nodes v_j, u_k^j (where $j \in \{3,\ldots,\Delta-1\}$ and $k \in \{1,2,\ldots,l_j\}$) along $\mathcal{P}$ of $G \in \mathcal{G}$.*

Theorem 6.5.18. *At least 2 agents are destroyed by the BBH from v_i.*

Proof. The graph chosen by the adversary from $\mathcal{G}$, satisfies the condition that, v_i is Conflict-Free, i.e., within the interval $[r_i^\circ, r_i]$, the adversary need not activate the BBH, due to any movement from v_j along $\texttt{Ext}_1$ or $\texttt{Ext}_2$ (for all $v_i \in \mathcal{V}_1$ or $v_i \in \mathcal{V}_2$, where $j \in \{1,2,\ldots,i-1\}$). Further, Lemmas 6.5.14, 6.5.15 and 6.5.16, ensure that, irrespective of the knowledge gained by the agents before round r_i°, there exists a round r_i'' ($\geq r_i$) at which the second agent gets destroyed from v_i, while traversing along $\texttt{Ext}_1$ or $\texttt{Ext}_2$ from v_i (depending on $v_i \in \mathcal{V}_1$ or $v_i \in \mathcal{V}_2$). Hence, this proves the theorem. □

So, the possible graph class choices eventually reduces to $\mathcal{G}^{\Delta-1}$, where $\mathcal{G}^{\Delta-1} \subset \cdots \subset \mathcal{G}^1 \subset \mathcal{G}$.

Remark 6.5.19. If $v_\Delta \in \mathcal{V}_2$, then choose $D_{\mathcal{A}}(v_{\Delta-1}, v_\Delta)$, in a similar idea as chosen for v_i (for all $i \in \{3,4,\ldots,\Delta-1\}$). Also, the choices posed by $\mathcal{A}$ from v_Δ are exactly similar to the ones from $v_{\Delta-1}$, and their adversarial countermeasures are also similar. So, using them, we can conclude that not only v_Δ can be Conflict-Free, but also, similar to Theorem 6.5.18, we can say that at least 2 agents are destroyed by the BBH from v_Δ.

Otherwise, set $D_{\mathcal{A}}(v_{\Delta-1}, v_\Delta) = 2$. So, finally we can conclude that depending on whether $v_\Delta \in \mathcal{V}_2$ or $v_\Delta \notin \mathcal{V}_2$, the possible graph choices further reduces to $\mathcal{G}^\Delta$, where $\mathcal{G}^\Delta \subset \mathcal{G}^{\Delta-1}$.

Finally, to conclude the proof of Theorem 6.5.1, the adversary chooses the graph $G = (V, E, \lambda)$ from $\mathcal{G}^\Delta$, where $\mathcal{G}^{\Delta-1} \subset \mathcal{G}^{\Delta-2} \subset \cdots \subset \mathcal{G}$, and sets $v_1 = home$. Next starting from v_1, Theorems 6.5.3, 6.5.11 and 6.5.18 ensure that 2 agents are destroyed from each $v_i \in V$, where $i \in \{1,2,\ldots,\Delta-1\}$. This contradicts the fact that $\mathcal{A}$ solves PERPEXPLORATION-BBH with $2(\Delta-1)$ agents.

Notation	Description
$\mathscr{G}$	Indicates the specific graph from which the adversary chooses the graph, as per $\mathcal{A}$.
$\mathscr{G}^i$	Subclass of $\mathscr{G}$, where $\mathscr{G}^i \subset \mathscr{G}^{i-1} \subset \cdots \subset \mathscr{G}$.
$\mathscr{P}$	Indicates the path graph on a graph in $\mathscr{G}$ consisting of vertices of type v_i and u^i_j.
$\mathscr{V}$	Indicates the set of vertices $v_1, \ldots, v_{\Delta-1}$.
$dist_{\mathscr{P}}(v_{i-1}, v_i)$	Indicates the distance between v_{i-1} and v_i along $\mathscr{P}$, also denoted by $l_{i-1}+1$.
$\texttt{Ext}_1$,$\texttt{Ext}_2$	Indicates two variation of subgraphs.
$\mathscr{G}_1, \mathscr{G}_2, \mathscr{G}_3$	The different sub-class under class $\mathscr{G}$, where $\mathscr{G}^i \in \mathscr{G}_1$ or $\mathscr{G}_2$ or $\mathscr{G}_3$, for all i.
w_i	It is a vertex connected to $v_i \in \mathscr{V}_2$, where it is of degree Δ
T^i_v	A tree special tree (explained in Block-3) originating from v along port i.
$\lambda^1_{v,\mathcal{A}}$	Function for adversary to port label each vertex, except $\{v_1, v_2, \ldots, v_\Delta\}$.
$\lambda^2_{v_i,\mathcal{A}}$	Function for adversary to port label each vertex v_i, where $i \in \{1,2,\ldots,\Delta\}$.
V_{exp}, V^c_{exp}	Indicates the set of explored and unexplored vertices, such that $V = V_{exp} \cup V^c_{exp}$.
r°_i	Indicates the round at which any agent first visits v_i.
r_i, r''_i	Indicates the rounds at which the first and second agent destroyed from v_i.
t_i	Indicates the interval of rounds between the destruction of the first agent. and second agent from v_i.
T^{max}_j	Maximum of t_i, for all $i \in \{1,2,\ldots,j-1\}$.
$Wait_i$	Indicates the number rounds between any agent first visits v_i and the first agent (or agents) getting destroyed.
$Time_{i-1}$	Indicates $T^{max}_{i-1} - Wait_i$

Table 6.3: Table for the notations used in Section 6.5.1.

6.5.2 Description of Algorithm GRAPH_PERPEXPLORE-BBH-HOME

Here we discuss the algorithm, termed as GRAPH_PERPEXPLORE-BBH-HOME that solves PERPEXPLORATION-BBH-HOME on an arbitrary graph, $G = (V, E, \lambda)$. We will show that our algorithm requires at most $3\Delta + 3$ agents. Let $\langle G, 3\Delta+3, h, v_b\rangle$ (where v_b be the BBH node) 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 6.3.2), EXPLORE (explained in this section) and BFS-TREE-CONSTRUCTION [34]. Before delving into the details of our algorithm that solves PERPEXPLORATION-BBH-HOME, we recall the concept 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,

$\mathscr{P} = \{P_v$: edge labeled shortest path from h to $v, \forall v \in V$ 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 $\mathscr{P}$ from w one after the other, to find if one among them takes it to the marked node h. If so, then it adds a cross-edge (u, w) to its map. Otherwise, it adds to the already constructed BFS tree, the node w, accordingly $\mathscr{P} = \mathscr{P} \cup P_w$ is updated. The underlying data structure of ROOT_PATHS [34] is used to perform these processes. This strategy guarantees as per Proposition 9 of [34], 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 (as shown in Theorem 6.5.28, $k = 3\Delta + 3$ agents are sufficient), where they are initially co-located at a node $h \in V$, which is referred to 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 $\texttt{LG}_0$ form $\texttt{LG}_1$. 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 $\mathcal{C}_1$ that leads to the BBH

(where $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_t$ are the connected components of $G - v_b$, such that $h \in \mathcal{C}_1$), we term these blocking agents as *anchors*, whereas the remaining alive agents must perpetually explore at least $\mathcal{C}_1$.

Initially from h, the members of SG start their movement, and the members of $\mathtt{LG}_0$ stay 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 6.3.2). 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 into two sub-phases: i_1-th phase and i_2-th phase. In the i_1-th phase, the members of SG make 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_2+1) 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 6.3.2, each translation using TRANSLATE_PATTERN requires 5 rounds and creating the pattern using MAKE_PATTERN 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 6.3.2). The agent (or agents) that 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 $\mathtt{Anchor}(\alpha)$.

Movement of $\texttt{LG}_0$: 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 $\texttt{LG}_0$ start their movement.

Starting from h, the underlying movement of the members of $\texttt{LG}_0$ is similar to BFS-TREE-CONSTRUCTION, but while moving from one node to another, they do not execute either MAKE_PATTERN or TRANSLATE_PATTERN, unlike the members of SG. In this case, if all members of $\texttt{LG}_0$ are currently at a node $u \in V$, then the three lowest ID members of $\texttt{LG}_0$ 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 $\texttt{LG}_0$ is v, where $v \in N(u)$, then the following procedure is performed by the explorers of $\texttt{LG}_0$, before $\texttt{LG}_0$ finally decides to visit v.

Suppose at round r (for some $r > 0$), $\texttt{LG}_0$ 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 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 $r + 4 \cdot (deg(v) - 1) + 1$ both E_2^0 and E_3^0 return to u. And in the subsequent round, each member of $\texttt{LG}_0$ visits 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 $\texttt{LG}_0$ choose to visit from a neighbor node u. After the completion of EXPLORE(v), each member of $\texttt{LG}_0$ (including the explorers) visit v from u.

It may be noted that, if the members of $\texttt{LG}_0$ 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 $\texttt{LG}_0$ check if there exists an *anchor* agent blocking that port, which leads to v. If that is the case, then $\texttt{LG}_0$ 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 $\texttt{LG}_0$ members, then they backtrack to the parent node of u, and start iterating the same process.

From the above discussion, we can make the following remark.

Remark 6.5.20. If at some round t, the explorer agents of $\texttt{LG}_i$ (i.e., E_1^i, E_2^i and E_3^i), are ex-

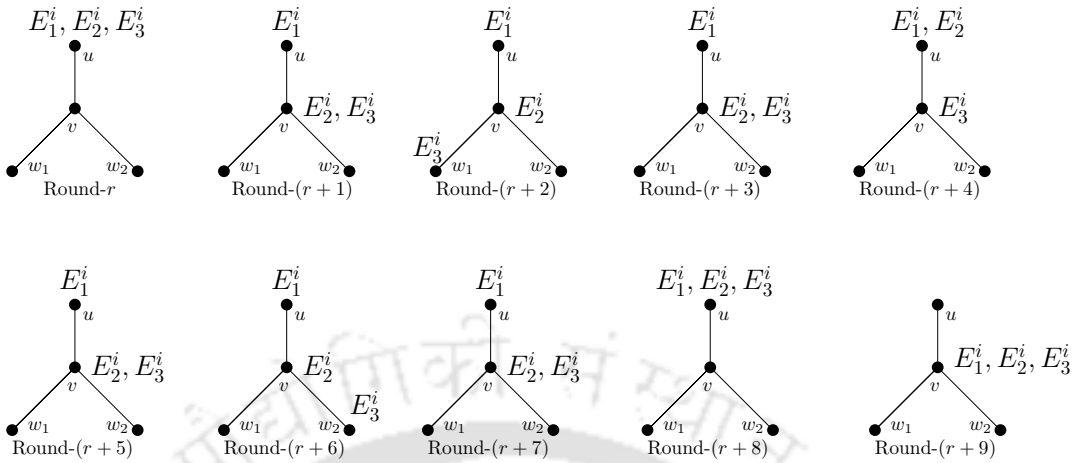


Figure 6.19: Depicts the round-wise execution of EXPLORE(v) from u by the explorer agents of $\mathtt{LG}_i$, for some $i \geq 0$ on the neighbors w_1 and w_2 of v.

ploring a two length path, say $P = u \rightarrow v \rightarrow w$, 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 $u \rightarrow v$ first. Now, from v, E^i_3 chooses the next port in a particular pre-decided order (excluding the port through which it entered v). So, whenever it returns to v to meet E^i_2 after visiting a node w, E^i_2 knows which port it last visited and which port it will choose next and relay that information back to other agents of $\mathtt{LG}_i$ on u. So, after E^i_2 returns to v again from u when E^i_3 starts visiting the next port, all other agents know about it.

During the execution of EXPLORE(v) from u, the agent E^0_3 can face one of the following situations:

- It can find an *anchor* agent at v, acting as $\mathtt{Anchor}(\beta)$, for some $\beta \in \{1, \ldots, deg(v)\}$. In that case, during its current execution of EXPLORE(v), E^0_3 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 $\mathtt{Anchor}(\beta')$, where $\beta' \in \{1, \ldots, deg(w)\}$. If the port connecting w to v is also β', then E^0_3 understands v (or its previous node) is the BBH, and accordingly tries to return to u, along the path $w \rightarrow v \rightarrow u$, 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 the port connecting w to v is not β', then E^0_3 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 $\mathtt{LG}_1$ is formed, where $\mathtt{LG}_1 = \mathtt{LG}_0 \setminus \{E_3^0\}$. Next, the members of $\mathtt{LG}_1$ 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 $\mathtt{LG}_0$ in the next round, and then returns back to v, and becomes $\mathtt{Anchor}(\beta)$, where $\beta \in \{1,\ldots,deg(v)\}$ and β is the port connecting v to w. On the other hand, $\mathtt{LG}_0$ after receiving this information from E_2^0, transforms to $\mathtt{LG}_1$ (where $\mathtt{LG}_1 = \mathtt{LG}_0 \setminus \{E_2^0, E_3^0\}$) 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 $\mathtt{Anchor}(\beta)$ at u, where β is the port connecting u to v. Moreover, the remaining members of $\mathtt{LG}_0$, i.e., $\mathtt{LG}_0 \setminus \{E_1^0, E_2^0, E_3^0\}$ forms $\mathtt{LG}_1$ 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).

From the above discussions, we conclude the following lemma.

Lemma 6.5.21. *Let at round t (for some $t > 0$), the explorer agents of $\mathtt{LG}_i$ (i.e., E_1^i, E_2^i and E_3^i) starts exploring the 2 length path $P = u \to v \to w$ from u while executing* EXPLORE*(v). Then, during the exploration of the path P, we have the following results.*

1. *If any of the explorer agents gets destroyed at v then, there will be at least one explorer agent of $\mathtt{LG}_i$ that identifies v as the BBH and the port from u leading to v is the port leading to the BBH. Also, it can relay this information to the remaining agents of $\mathtt{LG}_i$.*

2. *If any of the explorer agents gets destroyed at w then, there will be at least one explorer agent of $\mathtt{LG}_i$ that identifies w as the BBH and the port from v leading to w is the port leading to the BBH. Also, it can relay this information to the remaining agents of $\mathtt{LG}_i$.*

3. *If an explorer agent meets an anchor agent at w pointing the port from w to v as the port leading to the BBH then, there will be at least an explorer agent of $\mathtt{LG}_i$ that identifies v as the BBH and the port from u leading to v is the port leading to the BBH. Also, it can relay this information to the remaining agents of $\mathtt{LG}_i$.*

Illustration of points 1 and 3 of Lemma 6.5.21 is described in (i) and (ii) of Fig. 6.20.

It may be noted that, if for some $j > 0$, $|\mathtt{LG}_j|$<3, then those members perpetually explore G by executing simply BFS-TREE-CONSTRUCTION and not performing EXPLORE(), unlike the movement for the members of $\mathtt{LG}_t$, for $0 \leq t < j$. Whenever during this execution, the member of $\mathtt{LG}_j$, encounter $\mathtt{Anchor}(\beta)$ at a node, say u, then the member of $\mathtt{LG}_j$ avoids choosing the port β for its next move.

In addition to that, after the members in $\mathtt{LG}_i$ (for some $i \geq 1$), obtain the map of G, they perpetually explore each node in G except the BBH, by performing simple BFS traversal, and while performing this traversal, the members of $\mathtt{LG}_i$ always avoids the ports, blocked by some *anchor* agent.

Next, we prove the correctness of our algorithm GRAPH_PERPEXPLORE-BBH-HOME.

Lemma 6.5.22. *The members of $\mathtt{SG}$ perpetually explore G until their movement is intervened by the BBH.*

Proof. The members of $\mathtt{SG}$ operate in phases. As stated earlier, each i-th phase (for some $i > 0$) is sub-divided into two parts: i_1-th phase and i_2-th phase. In the i_1-th phase, the agent translates up to 2^i nodes, and each node that it translates to is decided by the underlying algorithm BFS-TREE-CONSTRUCTION. In the i_2-th phase, no matter what the position of the members of $\mathtt{SG}$, they start translating back to the home. By Proposition 9 of [34], it

is shown that if an agent follows BFS-TREE-CONSTRUCTION algorithm, then it eventually constructs the map within $\mathcal{O}(n^3\Delta)$ steps with $\mathcal{O}(n\Delta \log n)$ memory, where $|V| = n$ and Δ is the maximum degree of $G = (V, E)$. This implies that, for a sufficiently large j, such that $2^j \geq cn^3\Delta$, for some constant c, the members of SG must have obtained the map of G in the j_1-th phase, if the BBH does not intervene in the movement of SG, in any phase less than equal j-th phase. So, for any subsequent phase after j-th phase until the BBH intervenes, the members of SG explores G. □

Lemma 6.5.23. *If BBH intervenes in the movement of SG, at least one member of SG acts as an anchor.*

Proof. The movement of the members of SG is based on translation, and in order to translate, they execute the algorithms MAKE_PATTERN and TRANSLATE_PATTERN. If the BBH intervenes during their movement, it means the BBH intervenes during translation, or in other words, the BBH intervenes during MAKE_PATTERN or TRANSLATE_PATTERN. We already know that if the BBH intervenes during the execution of either MAKE_PATTERN or TRANSLATE_PATTERN, then at least one agent executing these algorithms must identify the location of the BBH (as per Section 6.3.2). As per the argument in Section 6.3.2, if at round r (for some $r > 0$), the member determines the position of the BBH, then either it is on BBH at round r in which case it moves to an adjacent node in $\mathcal{C}_1$, say v, or at round r the member is present at an adjacent node v' ($v' \in \mathcal{C}_1$ or $\mathcal{C}_j$, $j \neq 1$). In either situation, the member remains at v or v', *anchoring* the edge connecting BBH to v from round r onwards or BBH to v' from round r' onwards. □

Before going to the next section of correctness, we define the notion of $\texttt{LG}_i$ at a node u (for any $u \in V$), if every member that is part of $\texttt{LG}_i$ is at that node u.

Lemma 6.5.24. *For any $i \geq 0$, $\texttt{LG}_i$ can never be located at the BBH.*

Proof. If possible, let there exist a round t (for some $t > 0$) and $i \geq 0$ such that $\texttt{LG}_i$ is located on the BBH node v_b at round t. Without loss of generality, let this be the first round when $\texttt{LG}_i$ is on the BBH for any i. Now, since initially $\texttt{LG}_0$ was located at $h \in \mathcal{C}_1$ ($\mathcal{C}_1$ is the component of $G - v_b$ containing h), $\texttt{LG}_i$ must have moved onto v_b from some vertex $u \in N(v_b) \cap \mathcal{C}_1$.

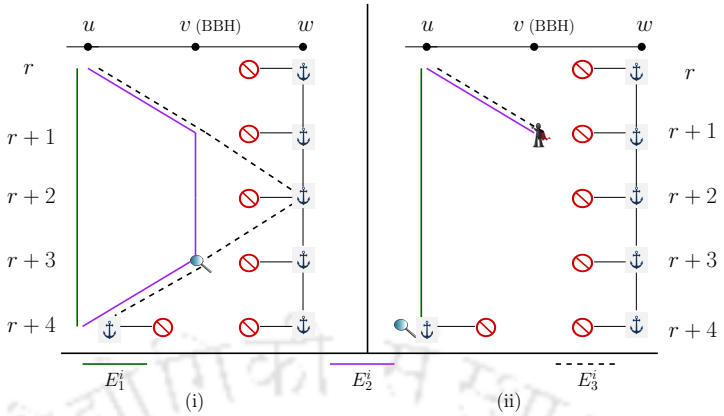


Figure 6.20: Depicts the time diagram of EXPLORE(v) of $\mathtt{LG}_i$ along a specific path $P = u \rightarrow v \rightarrow w$, where $v = v_b$, in which w contains an *anchor* agent for the edge (v, w). In (i), it depicts even if v_b is not activated, an explorer agent settles as an *anchor* at u for the edge (u, v). In (ii), it depicts that activation of v_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).

This implies there must exists a round $t' < t$ such that at t', $\mathtt{LG}_i$ is at u and E_1^i, E_2^i and E_3^i starts procedure EXPLORE(v_b) from u. Note that u does not have any agent acting as anchor, otherwise $\mathtt{LG}_i$ can never move to v_b from u. Now, since there is at least one neighbor, say u', of v_b where there is an agent settled as an anchor (as per Lemma 6.5.23), the distance between u and u' must be 2. So when executing EXPLORE(v_b), either E_3^i meets with the anchor agent at u' while exploring the path $u \rightarrow v_b \rightarrow u'$ or, at least one of E_2^i and E_3^i gets destroyed at v_b while executing EXPLORE(v_b). Now, from the results obtained in points 1 and 3 of Lemma 6.5.21, there exists an explorer agent that will know that v_b is the BBH and the port, say β, from u leading to v_b is the port leading to the BBH. In this case, this agent relays this information to the other members of $\mathtt{LG}_i$ located at u and settles at u as $\mathtt{Anchor}(\beta)$. The remaining members of $\mathtt{LG}_i$ forms $\mathtt{LG}_{i+1}$ and continue the algorithm, avoiding the port β from u. This contradicts our assumption that $\mathtt{LG}_i$ moves to v_b from u. Hence, we can conclude that $\mathtt{LG}_i$ can never be located on the BBH for any $i \geq 0$. □

From the above result, we can have the following corollary.

Corollary 6.5.25. *For any $i \geq 0$, $\mathtt{LG}_i$ can never be located at a node outside $\mathcal{C}_1$, where $G - v_b = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_t$ and $h \in \mathcal{C}_1$, v_b is denoted to be the BBH.*

Lemma 6.5.26. *Let $\mathcal{U} \subseteq N(v_b) \cap \mathcal{C}_1$, such that if $u \in \mathcal{U}$, then u does not contain any anchor. If $|\mathcal{U}| > 0$, then our algorithm* GRAPH_PERPEXPLORE-BBH-HOME *ensures that $|\mathcal{U}|$ decreases, eventually.*

Proof. If at time t (for some $t > 0$), the members of $\mathtt{LG}_i$ (for some $i \geq 0$) reach u, then we show that eventually u will not belong to $\mathcal{U}$. As per our algorithm, whenever $\mathtt{LG}_i$ reaches a node u, then three explorer agents of $\mathtt{LG}_i$ perform EXPLORE() on the next neighbor node to be chosen by $\mathtt{LG}_i$ to visit from u. Suppose, at time $t' > t$, $\mathtt{LG}_i$ chooses to visit v_b from u (since $u \in N(v_b)$). Hence, before visiting v_b, three explorer agents start EXPLORE(v_b), and during this execution, as per points 1 and 3 of Lemma 6.5.21, one explorer agent of $\mathtt{LG}_i$ settles down at u acting as an *anchor* for the edge (u, v_b). So, now u cannot belong to $\mathcal{U}$ (as it violates the definition of the set $\mathcal{U}$), so $\mathcal{U}$ gets modified to $\mathcal{U} \setminus \{u\}$.

Let at time t, we assume $|\mathcal{U}| = c$ for some constant c, and we suppose $\mathcal{U}$ does not decrease from t onwards. Now, as per the definition of $\mathcal{U}$, it contains all such nodes which are neighbors to v_b but does not contain any *anchor*. Let $u_1 \in V$ be the last node in $\mathcal{U}$, at which an *anchor* got settled at time t' (where $t' < t$) and due to which $\mathtt{LG}_i$ got formed from $\mathtt{LG}_{i-1}$. Now, as per our assumption, as $\mathcal{U}$ does not decrease from t onwards, so $\mathtt{LG}_i$ remains $\mathtt{LG}_i$. As per our algorithm, it follows that $\mathtt{LG}_i$ will visit every node of G (except through the edges, blocked by *anchor*). This means that $\mathtt{LG}_i$ will visit v_b through a neighbor node that does not contain any *anchor*, which cannot happen as per Lemma 6.5.24. □

The corollary follows from the above lemma.

Corollary 6.5.27. *Our algorithm ensures that eventually, $\mathcal{U}$ becomes $\emptyset$.*

We can now prove Theorem 6.5.28, which we recall below:

Theorem 6.5.28. *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. As per Corollary 6.5.27, we know that eventually $|\mathcal{U}| = \phi$. This means that every neighbor of v_b, reachable from h, contains an *anchor* agent, blocking that edge which

leads to v_b. Again, as per the correctness of BFS-TREE-CONSTRUCTION or just BFS traversal (provided the remaining agents know the map of G), the remaining agents (i.e., the agents which are alive and not *anchor* or `Marker`) will visit every reachable node of G, except the ones blocked by *anchor*, perpetually. Since, only v_b is blocked to visit by all the *anchors*, present at all of its neighbors, so this guarantees that $\mathcal{C}_1$ will be perpetually visited, hence it solves PERPEXPLORE-BBH-HOME.

Next, we consider just one 2 length path $P = u \rightarrow v_b \rightarrow w$ (refer to Fig. 6.20, where $u \in \mathcal{U}$ and w contains an *anchor*, blocking the port (v, BBH). So, whenever $\texttt{LG}_i$ (for some $i \geq 0$) reaches u (it will reach u, as per Lemma 6.5.26), according to our algorithm EXPLORE(v_b) will start eventually (i.e., before $\texttt{LG}_i$ decides to move to v_b), and it is known as per Lemma 6.5.21, an explorer will settle as a *anchor* at u, surely after they finish visiting the path P (it may happen earlier during the execution of EXPLORE(v_b) as well). Now, while visiting P, in the worst scenario, the adversary can activate v_b, the moment E_2^i and E_3^i are at v_b. This means, the remaining explorer E_1^i will become an *anchor* at u, for the edge (u, v_b) and u will not be in $\mathcal{U}$. Now, this situation can occur from each neighbor of v_b, except one, i.e., where already at least one member of `SG` is settled as an *anchor*. This proves that, $3(\Delta - 1)$ agents are required to *anchor*, $\Delta - 1$ adjacent ports of v_b (if $deg(v_b) = \Delta$), in the worst case. So, except the `Marker` agent (which remains stationary at h), in the worst case $3\Delta + 1$ agents are required to `anchor` each Δ many adjacent ports of v_b, including members of `SG`. Now, this does not ensure perpetual exploration, as all alive agents are stationary, either as an *anchor* or `Marker`. Hence, we require at least $3\Delta + 3$ agents to solve PERPEXPLORE-BBH-HOME. In addition to this, to execute our algorithm, we use the underlying concept of BFS-TREE-CONSTRUCTION, and this requires each agent to have a memory of $\mathcal{O}(n\Delta \log n)$. □

The next remark calculates the total time required for `SG` to translate to each node in G.

Remark 6.5.29. The movement of `SG` is performed in phases. So, in j-th phase it performs at most 2^j translations using the underlying algorithm of BFS-TREE-CONSTRUCTION, and then returns back to h using again at most 2^j translations. Each translation further requires 5 rounds. So, j-th phase is executed within $\mathcal{O}(2^{3j}\Delta)$ rounds. Hence, to explore G it takes $\sum_{j=1}^{\log n} \mathcal{O}(2^{3j}\Delta) = \mathcal{O}(n^3\Delta)$ rounds.

The following remark discusses the time required to explore every node of G by $\texttt{LG}_i$ after SG fails to return at the i-th phase, for some $i > 0$. and SG, respectively.

Remark 6.5.30. Let v be a node, and w be a neighbor of v with degree Δ. It must be observed that, to perform EXPLORE(w) by $\texttt{LG}_i$ from u (for some $i \geq 0$), and eventually reach w, it takes $4(\Delta-1)+2$ rounds, as to explore a neighbor of w it takes 4 rounds and except the parent, all neighbors must be explored, and finally from v to reach w it takes another 2 rounds. As discussed earlier, in general, to perform BFS-TREE-CONSTRUCTION, it takes $\mathcal{O}(n^3\Delta)$ rounds to visit each node of G, and construct a map of G.

So, for $\texttt{LG}_i$ to visit each node in G, and accordingly construct a map also during the course of which *anchor* each port in $\mathcal{C}_1$, $\texttt{LG}_i$ takes $(4(\Delta-1)+2)\cdot\mathcal{O}(n^3\Delta) = \mathcal{O}(n^3\Delta^2)$ rounds.

6.5.3 Perpetual Exploration in the Presence of a Black Hole

In the special case in which the BBH 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 [61].

For a fixed $\Delta \geq 4$, we construct an (unlabeled) graph $G_\Delta = (V_\Delta, E_\Delta)$, of maximum degree Δ, which contains (referring to Figure 6.21) a single vertex v of degree Δ (i.e., the maximum degree in G), Δ many vertices of degree 4 (refer to the vertices u_i, for $i \in \{0,1,\ldots,\Delta-1\}$), and Δ many vertices of degree 1 (refer to the vertices w_i, for $i \in \{0,1,\ldots,\Delta-1\}$).

Theorem 6.5.31. *For every $\Delta \geq 4$, there exists a port-labeling of graph G_Δ such that any algorithm using at most Δ agents fails to solve* PERPEXPLORATION-BH *in G_Δ.*

Proof. Consider the graph $G = (V, E, \lambda)$ explained in Fig. 6.21, and let u_0 be *home*, where initially the agents $A = \{a_1, \ldots, a_\Delta\}$ are co-located, consider v to be the BH. We shall prove this theorem by contradiction. Suppose, $\mathcal{A}$ be an algorithm that solves PERPEXPLORATION-

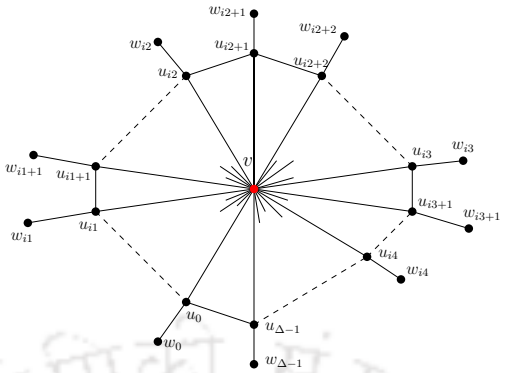


Figure 6.21: Depicts a graph $G = (V, E, \lambda)$, with the vertex v being the BBH.

BH with Δ agents, starting from u_0. It may be noted that the agents do not have the knowledge of the port-labeling of G_Δ.

Since, in the worst case, to gather the map of G, the algorithm $\mathcal{A}$ must instruct the agents to visit every node in V. Now, for each $u_i \in V$, whenever u_i is visited by an agent for the first time, say at round r_i, the agent cannot identify the ports from u_i leading to w_i or v or u_{i+1}. So, the adversary, based on which, has returned the port-labeling function $\lambda_{u_i,\mathcal{A}}$ for each $i \in \{0, 1, \ldots, \Delta-1\}$, such that, let r'_i be the first round after r_i at which any agent (say, a_j) from u_i decides to visit an unexplored port, then a_j reaches v at round $r'_i + 1$, based on $\lambda_{u_i,\mathcal{A}}$. Since v is the BH, a_j gets destroyed along the edge (u_i, v). This phenomenon can occur for each $u_i \in V$, for all $i \in \{0, 1, \ldots, \Delta - 1\}$. This shows that Δ agents are destroyed, leading to a contradiction for the algorithm $\mathcal{A}$. □

Next remark discusses an algorithm that solves PERPEXPLORATION-BH with $\Delta+2$ agents.

Remark 6.5.32. A slight modification of BFS-TREE-CONSTRUCTION algorithm can generate a $\Delta + 2$ algorithm for PERPEXPLORATION-BH. A brief idea of which is as follows: an agent stays at *home*, acting as a `Marker`. Among the remaining agents, one agent is selected to be an `Explorer`. Each agent maintains a set of edge-labeled paths, $\mathscr{P}$, as explained in BFS-TREE-CONSTRUCTION. During their traversal, the agents perform a *cautious* walk, i.e., the `Explorer` agent is the one to visit a new node, and after `Explorer` agent finds it safe, then only remaining agents visits the new node. After the agents visit a new node, say v, they check whether v has already been visited or not. To check this, `Explorer` remains at v, whereas other agents return to `Marker`. Next, they start visiting all the paths stored

in $\mathscr{P}$, one after the other. If any of these paths lead to v (where the `Explorer` is present), they update the edge, say (u, v), taken by the `Explorer` to reach v, as a cross edge in the map. Otherwise, if no path leads to the `Explorer`, then v is added to the map, and $\mathscr{P} = \mathscr{P} \cup P_v$. Accordingly, the remaining agents, except `Marker`, reach back to the `Explorer`. This process repeats.

Now, suppose v is already visited, and there exists a port ρ, which leads to the black hole. So, whenever `Explorer` visits the node with respect to ρ, it gets destroyed. Remaining agents at v, update in the map that, with respect to v, the port ρ leads to the black hole. Accordingly, a new `Explorer` is elected, and the process continues. In future, during this process, if ever the agents again reach v (which they understand via the stored edge-labeled paths in $\mathscr{P}$), through any cross edge not already present in the map constructed yet, then the agents do not take the port ρ from v again. This shows that, at most Δ agents are destroyed, one agent remains alive, except the `Marker`, which has the knowledge of all the ports leading to the black hole, and can perpetually explore the current component of $G - \text{BH}$. So, a $\Delta + 2$ agent PERPEXPLORATION-BH algorithm can be formulated, using this strategy.

6.6 Conclusion

In this chapter, we gave the first non-trivial upper and lower bounds on the optimal number of agents for perpetual exploration, in the presence of at most one BBH in arbitrary graphs.

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 arbitrary graphs. Second, note that our arbitrary graph lower bound of $2\Delta - 1$ holds only for graphs of maximum degree at least 4 (Theorem 6.5.1). 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 BBH is activated in each round), close the gap between $\Delta + 1$ and $\Delta + 2$ agents. Lastly, what about this study in other scheduler models (such as in semi-synchronous or asynchronous scheduler models)?

Chapter 7

Conclusion

In this thesis, we have touched on many interesting and fundamental topics related to distributed algorithms with mobile entities. Our goal has been to design problems that address many relevant models of interest. Starting from Chapter 3, we studied the treasure hunt problem. We considered the underlying topology to be the Euclidean plane, and introduced how pebbles can be used to design efficient treasure hunt algorithms. This chapter showcases the trade-off between the number of pebbles (k) that can be placed vs. the cost of finding the treasure. We obtained many interesting results based on the value of k, but for the case when $k \geq 11$, we are unable to provide any lower bound on the cost of finding the treasure. Hence, this remains an open question.

In Chapter 4, we looked into the black hole search problem. Normally, this problem is often studied in static graphs. Nothing much was known about this problem on dynamic graphs, except the results by Di Luna et al. [54, 56], which only focused on the dynamic ring. So, we extended this problem to a dynamic cactus. We solved this problem for two models of dynamicity: first, when one edge is dynamic, and second, when at most k edges are dynamic. Here, the results obtained by us are not optimal, and there is a significant gap for k dynamic edge case, on the number of agents. So, these are some of the interesting future directions.

Next, in Chapters 5 and 6, we looked into the problem of perpetual exploration in the presence of a more powerful variant of black hole, termed as *Byzantine black hole* (BBH).

In Chapter 5, we specifically studied this problem when the underlying topology is a static ring. Here, we investigated how the bounds on the number of agents required to solve this problem change with different models of communication (such as pebble, whiteboard and face-to-face) under various initial agent configurations (such as co-located and scattered). Almost all our results in this chapter are either optimal or close to optimal; however, certain open directions still remain. For example, we are not able to establish any upper or lower bound results on the number of agents to solve this problem when the agents are initially scattered and they communicate via a face-to-face model of communication.

In Chapter 6, we extended the problem of perpetual exploration in the presence of a BBH in arbitrary unknown graphs. This chapter analyses the fact that, by just making the black hole more powerful, the same problem becomes more difficult to solve. The results obtained in acyclic graphs are optimal; however, there is a gap between the upper bound and lower bound results for the arbitrary graph case. So, an immediate extension is to establish optimal bounds for the arbitrary graph case as well.

In Tables 7.1 and 7.2, we give a summary of all the chapters studied in the thesis. It contains the aim of the chapters, an overview of the results obtained and finally, in the remarks, we discuss the positive and negative sides of the chapter.

Component	**Key Findings / Results / Remarks**
CHAPTER 3: Treasure Hunt in Euclidean Plane	
Question	For given $k \geq 0$, what is the minimum cost of treasure hunt if at most k pebbles are placed in the plane?
Result	1. Proved it is impossible to find treasure with 1 pebble. 2. Designed a treasure hunt algorithm when $k = 2$ and $k \geq 11$.
Remarks	1. First work to explore the treasure hunt problem with pebbles in the Euclidean plane. 2. No lower bound exists.

Table 7.1: Summary of Thesis Results and Remarks

Component	Key Findings / Results / Remarks
CHAPTER 4: Black Hole Search in a Dynamic Cactus	
Aim	Design an efficient black hole search algorithm under various models of dynamicity.
Results	1. When one edge is dynamic, obtained optimal black hole search strategy in terms of number of agents. 2. When k edges are dynamic, obtained upper bound and lower bound results to perform a black hole search.
Remarks	1. Not many results were known before this work for black hole search in dynamic graphs. 2. Solves a weak version of the black hole search. 3. The results are not optimal.
CHAPTER 5: Perpetual Exploration of a Ring with a Byzantine Black Hole	
Aim	Design an efficient perpetual exploration algorithm that explores each safe node in the presence of a BBH
Results	1. Obtained optimal or close to optimal results when the agents are initially co-located, for each model of communication (pebble, whiteboard, face-to-face). 2. Obtained optimal results for the pebble and whiteboard model of communication, when the agents are initially scattered.
Remarks	1. Investigates each model of communication in each agent's initial configuration. 2. Obtaining algorithm and lower bound when agents are initially scattered, for face-to-face model of communication is open.
CHAPTER 6: Perpetual Exploration of Arbitrary Graphs with a Byzantine Black Hole	
Aim	1. Design an efficient perpetual exploration algorithm that explores each safe node in the presence of a BBH of at least one component of $G - \{BBH\}$ 2. Design an efficient perpetual exploration algorithm that explores each safe node in the presence of a BBH of the component containing the starting node.
Results	1. Obtained optimal results for acyclic graphs. 2. Arbitrary graph results highlight how the presence of BBH makes the problem more difficult to solve in comparison to the presence of a black hole.
Remarks	1. First results obtained for a more powerful variant of a black hole in arbitrary graphs. 2. A gap exists in the arbitrary graph result.

Table 7.2: Extension of Table 7.1

7.1 Future Work

As a byproduct of this thesis there can be several directions for future work, here in this section we list some of the results.

1. Consider the same treasure hunt problem as defined in Chapter 3, but now suppose the adversary has the ability to misplace some of the pebbles from the positions placed by the Oracle. First, let us suppose a single pebble is misplaced by the adversary, how difficult will the same treasure hunt problem become? This is an interesting direction, as similar studies have only been done in graph networks [113].

2. In the black hole search problem we studied in Chapter 4, we only considered the underlying topology to be 1-interval connected. Now, some connectivity models are defined in Chapter 1, such as T-interval connectivity (where $T > 1$), Connectivity time and T-path connectivity. What about solving the black hole search problem in these connectivity models of dynamicity? This is an interesting and open direction.

3. In Chapter 6, we studied the perpetual exploration in the presence of a byzantine black hole. What about perpetual exploration in the presence of other more powerful variants of the byzantine black hole, such as gray$^+$ hole or red hole?

4. Another question: what if there are multiple gray holes or byzantine black holes? In that case, what is the minimum number of agents required to perpetually explore the underlying topology?

5. In Chapter 5 and 6, we considered the scheduler to be synchronous. What about the case where the scheduler is semi-synchronous or asynchronous?

Bibliography

[1] Ankush Agarwalla, John Augustine, William K Moses Jr, Sankar K Madhav, and Arvind Krishna Sridhar. Deterministic dispersion of mobile robots in dynamic rings. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–4, 2018.

[2] Chrysovalandis Agathangelou, Chryssis Georgiou, and Marios Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 250–259, 2013.

[3] Steve Alpern. Rendezvous search: A personal perspective. *Operations Research*, 50(5):772–795, 2002.

[4] Steve Alpern, Robbert Fokkink, L Gasieniec, Roy Lindelauf, and VS Subrahmanian. Search theory. *Springer*, 10:978–1, 2013.

[5] Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*. Springer, 2003.

[6] Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55. Springer Science & Business Media, 2006.

[7] Spyros Angelopoulos. Online search with a hint. *Information and Computation*, 295:105091, 2023.

[8] Spyros Angelopoulos, Alejandro López-Ortiz, and Konstantinos Panagiotou. Multi-target ray searching problems. *Theoretical Computer Science*, 540:2–12, 2014.

[9] John Augustine and William K Moses Jr. Dispersion of mobile robots: a study of memory-time trade-offs. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–10, 2018.

[10] Ricardo A Baeza-Yates, Joseph C Culberson, and Gregory JE Rawlins. Searching with uncertainty extended abstract. In *Scandinavian Workshop on Algorithm Theory*, pages 176–189. Springer, 1988.

[11] Iman Bagheri, Lata Narayanan, and Jaroslav Opatrny. Evacuation of equilateral triangles by mobile agents of limited communication range. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 3–22. Springer, 2019.

[12] Balasingham Balamohan, Paola Flocchini, Ali Miri, and Nicola Santoro. Time optimal algorithms for black hole search in rings. *Discrete Mathematics, Algorithms and Applications*, 3(04):457–471, 2011.

[13] 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.

[14] Evangelos Bampas, Nikos Leonardos, Euripides Markou, Aris Pagourtzis, and Matoula Petrolia. Improved periodic data retrieval in asynchronous rings with a faulty host. *Theoretical Computer Science*, 608:231–254, 2015.

[15] Vic Baston and Shmuel Gal. Rendezvous search when marks are left at the starting points. *Naval Research Logistics (NRL)*, 48(8):722–731, 2001.

[16] Hanane Becha and Paola Flocchini. Optimal construction of sense of direction in a torus by a mobile agent. *International Journal of Foundations of Computer Science*, 18(03):529–546, 2007.

[17] Anatole Beck. On the linear search problem. *Israel Journal of Mathematics*, 2(4):221–228, 1964.

[18] Anatole Beck. An optimal search (richard bellman). *SIAM Review*, 27(3):447–448, 1985.

[19] Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

[20] Michael A Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 269–278, 1998.

[21] Adri Bhattacharya, Barun Gorain, and Partha Sarathi Mandal. Treasure hunt in graph using pebbles. In *Stabilization, Safety, and Security of Distributed Systems: 24th International Symposium, SSS 2022, Clermont-Ferrand, France, November 15–17, 2022, Proceedings*, pages 99–113. Springer, 2022.

[22] Adri Bhattacharya, Giuseppe F. Italiano, and Partha Sarathi Mandal. Black hole search in dynamic tori. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2024, June 5-7, 2024, Patras, Greece*, volume 292 of *LIPIcs*, 2024.

[23] Adri Bhattacharya, Giuseppe F Italiano, and Partha Sarathi Mandal. Searching for a black hole in a dynamic cactus. *Journal of Graph Algorithms and Applications*, 29(2):127–166, 2025.

[24] Lélia Blin, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. Exclusive perpetual ring exploration without chirality. In *International Symposium on Distributed Computing*, pages 312–327. Springer, 2010.

[25] Anthony Bonato, Konstantinos Georgiou, Calum MacRury, and Paweł Prałat. Algorithms for p-faulty search on a half-line. *Algorithmica*, 85(8):2485–2514, 2023.

[26] François Bonnet, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. Asynchronous exclusive perpetual grid exploration without sense of direction. In *International Conference On Principles Of Distributed Systems*, pages 251–265. Springer, 2011.

[27] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

[28] Sébastien Bouchard, Yoann Dieudonné, Arnaud Labourel, and Andrzej Pelc. Almost-optimal deterministic treasure hunt in unweighted graphs. *ACM Trans. Algorithms*, 19(3), 2023.

[29] Sébastien Bouchard, Yoann Dieudonné, Andrzej Pelc, and Franck Petit. Deterministic treasure hunt in the plane with angular hints. *Algorithmica*, 82:3250–3281, 2020.

[30] Sébastien Bouchard, Arnaud Labourel, and Andrzej Pelc. Impact of knowledge on the cost of treasure hunt in trees. *Networks*, 80(1):51–62, 2022.

[31] Quentin Bramas, Pascal Lafourcade, and Stéphane Devismes. Infinite grid exploration with synchronous myopic robots without chirality. *Discrete Applied Mathematics*, 375:193–214, 2025.

[32] Quentin Bramas, Toshimitsu Masuzawa, and Sébastien Tixeuil. Crash-tolerant exploration of trees by energy-sharing mobile agents. In *28th International Conference on Principles of Distributed Systems (OPODIS 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[33] Jie Cai, Paola Flocchini, and Nicola Santoro. Network decontamination from a black virus. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pages 696–705. IEEE, 2013.

[34] 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.

[35] Jérémie Chalopin, Shantanu Das, Arnaud Labourel, and Euripides Markou. Black hole search with finite automata scattered in a synchronous torus. In *Distributed Computing: 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings 25*, pages 432–446. Springer, 2011.

[36] Jérémie Chalopin, Shantanu Das, Arnaud Labourel, and Euripides Markou. Tight bounds for black hole search with scattered agents in synchronous rings. *Theoretical Computer Science*, 509:70–85, 2013.

[37] Prabhat Kumar Chand, Manish Kumar, Sumathi Sivasubramaniam, and Anisur Rahaman Molla. Fault-tolerant dispersion of mobile robots. *Discrete Applied Mathematics*, 377:299–313, 2025.

[38] Timothy H Chung, Geoffrey A Hollinger, and Volkan Isler. Search and pursuit-evasion in mobile robotics: A survey. *Autonomous robots*, 31(4):299–316, 2011.

[39] Andrew Collins, Jurek Czyzowicz, Leszek Gąsieniec, and Arnaud Labourel. Tell me where i am so i can meet you sooner: (asynchronous rendezvous with location information). In *International Colloquium on Automata, Languages, and Programming*, pages 502–514. Springer, 2010.

[40] Jurek Czyzowicz, Leszek Gąsieniec, Thomas Gorry, Evangelos Kranakis, Russell Martin, and Dominik Pajak. Evacuating robots via unknown exit in a disk. In *International Symposium on Distributed Computing*, pages 122–136. Springer, 2014.

[41] Jurek Czyzowicz, Kostantinos Georgiou, and Evangelos Kranakis. Group search and evacuation. In *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 335–370. Springer, 2019.

[42] Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71(2-3):229–242, 2006.

[43] Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Searching for a black hole in synchronous tree networks. *Comb. Probab. Comput.*, 16(4):595–619, 2007.

[44] Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, and Sunil Shende. Wireless autonomous robot evacuation from equilateral triangles and squares. In *International Conference on Ad-Hoc Networks and Wireless*, pages 181–194. Springer, 2015.

[45] Jurek Czyzowicz, Andrzej Pelc, and Arnaud Labourel. How to meet asynchronously (almost) everywhere. *ACM Transactions on Algorithms (TALG)*, 8(4):1–14, 2012.

[46] Archak Das, Kaustav Bose, and Buddhadeb Sau. Memory optimal dispersion by anonymous mobile robots. *Discrete Applied Mathematics*, 340:171–182, 2023.

[47] Shantanu Das, Dariusz Dereniowski, Adrian Kosowski, and Przemysław Uznański. Rendezvous of distance-aware mobile agents in unknown graphs. In *International Colloquium on Structural Information and Communication Complexity*, pages 295–310. Springer, 2014.

[48] Shantanu Das, Giuseppe Antonio Di Luna, Daniele Mazzei, and Giuseppe Prencipe. Compacting oblivious agents on dynamic rings. *PeerJ Computer Science*, 7, 2021.

[49] Xiaotie Deng and Christos H Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

[50] Anders Dessmark, Pierre Fraigniaud, Dariusz R Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.

[51] Stéphane Devismes, Yoann Dieudonné, and Arnaud Labourel. Graph Exploration: The Impact of a Distance Constraint. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025)*, volume 334 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:18, 2025.

[52] Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. Terminating exploration of a grid by an optimal number of asynchronous oblivious robots. *The Computer Journal*, 64(1):132–154, 2021.

[53] G Di Luna, Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Distributed exploration of dynamic rings. *Distributed Computing*, 33:41–67, 2020.

[54] Giuseppe A Di Luna, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Black hole search in dynamic rings: The scattered case. In *27th International Conference on*

Principles of Distributed Systems (OPODIS 2023). Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2024.

[55] Giuseppe Antonio Di Luna, Paola Flocchini, Linda Pagli, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Gathering in dynamic rings. *theoretical computer science*, 811:79–98, 2020.

[56] 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.

[57] Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.

[58] Yann Disser, Jan Hackfeld, and Max Klimm. Undirected graph exploration with $\theta(\log\log n)$ pebbles. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 25–39. SIAM, 2016.

[59] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile search for a black hole in an anonymous ring. In *International Symposium on Distributed Computing*, pages 166–179. Springer, 2001.

[60] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. *Distributed Computing*, 19, 2006.

[61] 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.

[62] Stefan Dobrev, Nicola Santoro, and Wei Shi. Scattered black hole search in an oriented ring using tokens. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8. IEEE, 2007.

[63] Stefan Dobrev, Nicola Santoro, and Wei Shi. Using scattered mobile agents to locate a black hole in an un-oriented ring with tokens. *International Journal of Foundations of Computer Science*, 19(06):1355–1372, 2008.

[64] Yuval Emek, Tobias Langner, Jara Uitto, and Roger Wattenhofer. Solving the ants problem with asynchronous finite state machines. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II 41*, pages 471–482. Springer, 2014.

[65] Ofer Feinerman, Amos Korman, Zvi Lotker, and Jean-Sébastien Sereni. Collaborative search on the plane without communication. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, pages 77–86, 2012.

[66] Sándor Fekete, Chris Gray, and Alexander Kröller. Evacuation of rectilinear polygons. In *International Conference on Combinatorial Optimization and Applications*, pages 21–30. Springer, 2010.

[67] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Decontamination of hypercubes by mobile agents. *Networks: An International Journal*, 52(3):167–178, 2008.

[68] Paola Flocchini, Matthew Kellett, Peter C Mason, and Nicola Santoro. Searching for black holes in subways. *Theory of Computing Systems*, 50:158–184, 2012.

[69] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, et al. Distributed computing by mobile entities. *Current Research in Moving and Computing*, 11340(1), 2019.

[70] 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.

[71] Pierre Fraigniaud and Andrzej Pelc. Deterministic rendezvous in trees with little memory. In *International Symposium on Distributed Computing*, pages 242–256. Springer, 2008.

[72] Pierre Fraigniaud and Andrzej Pelc. Delays induce an exponential memory gap for rendezvous in trees. *ACM Transactions on Algorithms (TALG)*, 9(2):1–24, 2013.

[73] Konstantinos Georgiou, Sean Leizerovich, Jesse Lucier, and Somnath Kundu. Evacuating from l_p unit disks in the wireless model. *Theoretical Computer Science*, 944:113675, 2023.

[74] Barun Gorain and Partha Sarathi Mandal. Approximation algorithms for sweep coverage in wireless sensor networks. *J. Parallel Distributed Comput.*, 74(8):2699–2707, 2014.

[75] Barun Gorain, Kaushik Mondal, Himadri Nayak, and Supantha Pandit. Pebble guided optimal treasure hunt in anonymous graphs. *Theoretical Computer Science*, 922:61–80, 2022.

[76] Pritam Goswami, Adri Bhattacharya, Raja Das, and Partha Sarathi Mandal. Perpetual exploration of a ring in presence of byzantine black hole. In *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, 2024.

[77] Tsuyoshi Gotoh, Paola Flocchini, Toshimitsu Masuzawa, and Nicola Santoro. Exploration of dynamic networks: tight bounds on the number of agents. *Journal of Computer and System Sciences*, 122:1–18, 2021.

[78] Tsuyoshi Gotoh, Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Exploration of dynamic tori by multiple agents. *Theoretical Computer Science*, 850:202–220, 2021.

[79] Tsuyoshi Gotoh, Yuichi Sudo, Fukuhito Ooshita, and Toshimitsu Masuzawa. Dynamic ring exploration with (h, s) view. *Algorithms*, 13(6):141, 2020.

[80] David Ilcinkas and Ahmed M Wade. Exploration of dynamic cactuses with sublogarithmic overhead. *Theory of Computing Systems*, 65:257–273, 2021.

[81] James S Jennings, Greg Whelan, and William F Evans. Cooperative search and rescue with a team of mobile robots. In *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97*, pages 193–200. IEEE, 1997.

[82] Artur Jeż and Jakub Łopuszański. On the two-dimensional cow search problem. *Information Processing Letters*, 109(11):543–547, 2009.

[83] Ming-Yang Kao, John H Reif, and Stephen R Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and computation*, 131(1):63–79, 1996.

[84] Tanvir Kaur and Kaushik Mondal. Distance-2-dispersion: dispersion with further constraints. In *International Conference on Networked Systems*, pages 157–173. Springer, 2023.

[85] Tanvir Kaur, Ashish Saxena, Partha Sarathi Mandal, and Kaushik Mondal. Black hole search in dynamic graphs. In *Proceedings of the 26th International Conference on Distributed Computing and Networking*, pages 221–230, 2025.

[86] Bernard O Koopman. A theoretical basis for method of search and screening. Technical report, 1946.

[87] Dariusz R Kowalski and Adam Malinowski. How to meet in anonymous network. *Theoretical Computer Science*, 399(1-2):141–156, 2008.

[88] 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.

[89] Evangelos Kranakis, Danny Krizanc, and Sergio Rajsbaum. Mobile agent rendezvous: A survey. In *International Colloquium on Structural Information and Communication Complexity*, pages 1–9. Springer, 2006.

[90] Ajay D Kshemkalyani and Faizan Ali. Efficient dispersion of mobile robots on graphs. In *Proceedings of the 20th International Conference on Distributed Computing and Networking*, pages 218–227, 2019.

[91] Ajay D Kshemkalyani, Manish Kumar, Anisur Rahaman Molla, Debasish Pattanayak, and Gokarna Sharma. Dispersion is (almost) optimal under (a) synchrony. In *Proceedings of the 37th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 367–381, 2025.

[92] Ajay D Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Fast dispersion of mobile robots on arbitrary graphs. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 23–40. Springer, 2019.

[93] Ajay D Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Dispersion of mobile robots on grids. In *International Workshop on Algorithms and Computation*, pages 183–197. Springer, 2020.

[94] Ajay D Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Efficient dispersion of mobile robots on dynamic graphs. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 732–742. IEEE, 2020.

[95] Ajay D Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Dispersion of mobile robots using global communication. *Journal of Parallel and Distributed Computing*, 161:100–117, 2022.

[96] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 513–522, 2010.

[97] Anissa Lamani, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal deterministic ring exploration with oblivious asynchronous robots. In *International Colloquium on Structural Information and Communication Complexity*, pages 183–196. Springer, 2010.

[98] Elmar Langetepe. Searching for an axis-parallel shoreline. *Theoretical Computer Science*, 447:85–99, 2012.

[99] Euripides Markou and Michel Paquette. Black hole search and exploration in unoriented tori with synchronous scattered finite automata. In *Principles of Distributed Systems: 16th International Conference, OPODIS 2012*, pages 239–253. Springer, 2012.

[100] Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. *Journal of Parallel and Distributed Computing*, 74(1):2016–2026, 2014.

[101] Stanislav Miklík. *Exploration in faulty networks*. PhD thesis, Comenius University, Bratislava, 2010.

[102] Avery Miller and Andrzej Pelc. Time versus cost tradeoffs for deterministic rendezvous in networks. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 282–290, 2014.

[103] Avery Miller and Andrzej Pelc. Tradeoffs between cost and information for rendezvous and treasure hunt. *Journal of Parallel and Distributed Computing*, 83:159–167, 2015.

[104] Anisur Rahaman Molla, Kaushik Mondal, and William K Moses. Byzantine dispersion on graphs. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 942–951. IEEE, 2021.

[105] Anisur Rahaman Molla and William K Moses Jr. Dispersion of mobile robots: the power of randomness. In *International Conference on Theory and Applications of Models of Computation*, pages 481–500. Springer, 2019.

[106] Fukuhito Ooshita, Ajoy K Datta, and Toshimitsu Masuzawa. Self-stabilizing rendezvous of synchronous mobile agents in graphs. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 18–32. Springer, 2017.

[107] Petrişor Panaite and Andrzej Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33(2):281–295, 1999.

[108] Debasish Pattanayak, Gokarna Sharma, and Partha Sarathi Mandal. Dispersion of mobile robots in spite of faults. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 414–429. Springer, 2023.

[109] Andrzej Pelc and Ram Narayan Yadav. Cost vs. information tradeoffs for treasure hunt in the plane. *arXiv preprint arXiv:1902.06090*, 2019.

[110] Subhajit Pramanick, Saswata Jana, Adri Bhattacharya, and Partha Sarathi Mandal. Mutual visibility of luminous robots despite angular inaccuracy. *Theoretical Computer Science*, 1011:114723, 2024.

[111] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.

[112] Hans-Anton Rollik. Automaten in planaren graphen. In *Theoretical Computer Science 4th GI Conference: Aachen, March 26–28, 1979*, pages 266–275. Springer, 2005.

[113] Ashish Saxena, Barun Gorain, Subhrangsu Mandal, and Kaushik Mondal. Brief announcement: Pebble guided rendezvous despite fault. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 121–125. Springer, 2024.

[114] Ashish Saxena and Kaushik Mondal. Path connected dynamic graphs with a study of dispersion and exploration. *Theoretical Computer Science*, page 115390, 2025.

[115] Thomas C Schelling. *The Strategy of Conflict: with a new Preface by the Author*. Harvard university press, 1980.

[116] Hirokazu Seike and Yukiko Yamauchi. Separation of unconscious colored robots. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 328–343. Springer, 2023.

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

[118] Takahiro Shintaku, Yuichi Sudo, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Efficient dispersion of mobile agents without global knowledge. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 280–294. Springer, 2020.

[119] Yuichi Sudo, Masahiro Shibata, Junya Nakamura, Yonghwan Kim, and Toshimitsu Masuzawa. Near-Linear Time Dispersion of Mobile Agents. In *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:22, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[120] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[121] Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In *Symposium on Discrete Algorithms: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, volume 7, pages 599–608, 2007.

[122] Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms (TALG)*, 10:1–15, 2014.

[123] David W Wall. Messages as active agents. In *Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 34–39, 1982.

[124] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[125] Yukiko Yamauchi and Masafumi Yamashita. Pattern formation by mobile robots with limited visibility. In *International Colloquium on Structural Information and Communication Complexity*, pages 201–212. Springer, 2013.

[126] Yan Yang, Samia Souissi, Xavier Défago, and Makoto Takizawa. Fault-tolerant flocking for a group of autonomous mobile robots. *Journal of systems and Software*, 84(1):29–36, 2011.

Research Articles from the Thesis Work

Journal Publication

- Adri Bhattacharya, Barun Gorain, Partha Sarathi Mandal. "Treasure Hunt in Euclidean plane: Cost vs. Pebble Trade-off" in *International Journal of Foundations of Computer Science (IJFCS)*, pp. 1-37, August 2025.

- Adri Bhattacharya, Giuseppe F. Italiano, Partha Sarathi Mandal. "Searching for a black hole in a Dynamic Cactus" in *Journal of Graph Algorithms and Applications (JGAA)*, Vol. 29 No. 2, pp. 127-166, May 2025.

Conference Publications

- Adri Bhattacharya, Pritam Goswami, Evangelos Bampas, Partha Sarathi Mandal. "Perpetual exploration in anonymous synchronous networks with a Byzantine black hole". In *Proceedings of the 39th International Symposium on Distributed Computing (DISC 2025)*, (LIPIcs), Berlin, Germany, October 27 - 31, 2025.

- Pritam Goswami, Adri Bhattacharya, Raja Das, Partha Sarathi Mandal. "Perpetual Exploration of a Ring in Presence of Byzantine Black Hole". In *Proceedings of the 28th International Conference on Principles of Distributed Systems (OPODIS 2024)*, (LIPIcs), Volume , pp. Lucca, Italy, December 11-13, 2024.

- Adri Bhattacharya, Giuseppe F. Italiano, Partha Sarathi Mandal. "Black Hole Search in Dynamic Cactus Graph". In *Proceedings of the 18th International Conference and Workshops on Algorithms and Computation (WALCOM 2024)*, (LNCS-14549) (Springer),

Kanazawa, Japan, March 18-20, 2024.

- Adri Bhattacharya, Barun Gorain, Partha Sarathi Mandal. "Pebble Guided Treasure Hunt in Plane". In *Proceedings of the 11th International Conference on Networked Systems (NETYS 2023)*, (LNCS-14067), (Springer), Marrakech, Morocco, May 22-24, 2023.

Manuscripts Submitted/ Under Preparation

- Pritam Goswami, Adri Bhattacharya, Raja Das, Partha Sarathi Mandal."Perpetual Exploration of a Synchronous Ring in Presence of a Byzantine Black Hole" [**Submitted** to *Theoretical Computer Science (Elsevier)*].
- Adri Bhattacharya, Pritam Goswami, Evangelos Bampas, Partha Sarathi Mandal."Almost optimal perpetual exploration in anonymous synchronous networks with a Byzantine black hole" [**Under preparation** for submitting to Journal].

Other Publications

Journal Publication

- Subhajit Pramanick, Saswata Jana, Adri Bhattacharya, Partha Sarathi Mandal. "Mutual visibility of luminous robots despite angular inaccuracy" in *Theoretical Computer Science (TCS)*, Volume 1011, October 2024, 114723.

Conference Publications

- Prajyot Pyati, Navjot Kaur, Saswata Jana, Adri Bhattacharya, Partha Sarathi Mandal. "Separation of Unconscious Robots with Obstructed Visibility". In *Proceedings of the 22nd International Conference on Distributed Computing and Intelligent Technologies (ICDCIT 2026)*, Bhubaneswar, India, January 16-19, 2026.

- Saswata Jana, Subhajit Pramanick, Adri Bhattacharya, Partha Sarathi Mandal. "Time-optimal Asynchronous Minimal Vertex Covering by Myopic Robots on Graph". In *Proceedings of International Symposium on Algorithmics of Wireless Networks (ALGOWIN 2025)*, Warsaw, Poland, September 18-19, 2025.

- Subhajit Pramanick, Saswata Jana, Adri Bhattacharya, Partha Sarathi Mandal. "Distributed Uniform Partitioning of a Region using Opaque ASYNC Luminous Robots". In *Proceedings of the 25th International Conference on Distributed Computing and Networking (ICDCN 2024)*, (ACM), Chennai, India, January 4-7, 2024.

- Adri Bhattacharya, Giuseppe F. Italiano, Partha Sarathi Mandal. "Black Hole Search in Dynamic Tori". In *Proceedings of the 3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*(LIPIcs), Volume 292, pp. 6:1-6:16, Patras, Greece, June 5-7, 2024.

- Subhajit Pramanick, Saswata Jana, Adri Bhattacharya, Partha Sarathi Mandal. "Mutual Visibility with ASYNC Luminous Robots Having Inaccurate Movements". In *Proceedings of the 19th International Symposium on Algorithmics of Wireless Networks (ALGOWIN 2023)*,LNCS-14061 (Springer), Amsterdam, Netherlands, September 4-8, 2023.

- Adri Bhattacharya, Barun Gorain, Partha Sarathi Mandal. "Treasure Hunt in Graph Using Pebbles". In *Proceedings of the 24th International Symposium on Stabilizing, Safety, and Security of Distributed Systems (SSS 2022)*, LNCS-13751 (Springer), Clermont-Ferrand, France, November 15-17, 2022.