Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 29, no. 2, pp. 127–166 (2025)
DOI: 10.7155/jgaa.v29i2.3042

# Searching for a Black Hole in a Dynamic Cactus[1]

*Adri Bhattacharya*[1] *Giuseppe F. Italiano*[2] *Partha Sarathi Mandal*[1]

[1]Indian Institute of Technology Guwahati, Guwahati, India
[2]Luiss University, Rome, Italy

Submitted: May 2024 Accepted: April 2025 Published: May 2025

Article type: Regular paper Communicated by: R. Uehara, K. Yamanaka, and H.-C. Yen

**Abstract.** In this paper, we study the problem of black hole search 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. This problem was first investigated in the context of dynamic rings by Di Luna et al. (Proc. of ICDCS 2021, IEEE, pp. 987-997). In this paper, we extend the same problem to a dynamic cactus. We introduce two categories of dynamicity. Firstly, we examine the scenario where the underlying graph has at most one dynamic edge, i.e., at most, one edge can disappear or reappear at any round. Secondly, we consider the problem for at most $k$ dynamic edges. In both cases, the underlying graph must be connected irrespective of which edge (or edges) is dynamic. Now for each case of dynamicities, we establish lower and upper bounds on the number of agents, moves and rounds required to locate the black hole.

## 1 Introduction

We study the black hole search problem (also termed as BHS problem) by a team of mobile agents. A mobile agent in a graph network is a software code that migrates from one node to another while performing specific actions. Moreover, the agent, while traversing from one node to another, also interacts with the host environment at each node that it visits. One of the fundamental problems in the domain of mobile agents is *exploration* of networks. The exploration problem terminates once each node of the network is visited at least once by any exploring agent. The black hole search problem is a unique variation of this exploration problem. A black hole is a malicious node that

[1]A preliminary version of this work appeared in the proceedings of the 18th International Conference and Workshops on Algorithms and Computation (WALCOM) 2024 [3].

*E-mail addresses:* a.bhattacharya@iitg.ac.in (Adri Bhattacharya) gitaliano@luiss.it (Giuseppe F. Italiano) psm@iitg.ac.in (Partha Sarathi Mandal)

can delete any visiting agent without leaving any trace of its existence. In practicality, a black hole can be modeled into various real-life failure models. It can be malware that attacks a node in the network, where the behavior of that node resembles to that of a black hole, and hence it destroys any incoming agents. This scenario frequently arises within networked systems, particularly in situations requiring the safeguarding of agents from potential host attacks. In literature, the study of the BHS problem focuses on efficiently determining the position of the black hole in an unknown network under several assumptions such as communication models, agent's capabilities and level of synchrony being some of them. A complete survey of each of these assumptions is discussed by Markou et al. [25]. Note that each of these assumptions is explored when the underlying graph is *static*, i.e., the graph network remains the same over time.

Recent advancements show that the study of mobile agents is getting more traction on *highly dynamic graphs*, i.e., graphs that change over time. It is because due to the increasing prevalence of internet connectivity and mobile devices with connectivity, networks are becoming predominantly dynamic. Such networks change over time, i.e., a node can join, leave, or move around, or the links may appear or disappear over time. So, in this paper, we consider our underlying graph to be a synchronous *evolving graph*. These dynamic graphs can be visualized as a collection of static graphs with the additional constraint that at any round, whichever or howsoever many edges are reappeared or disappeared by the adversary, the underlying graph must remain connected. This constraint of the graph remaining connected always is termed as *1-interval connectivity* property.

Apart from the research paper by Di Luna et al. [12] concerning dynamic ring networks, there is limited knowledge about finding the black hole when the underlying network exhibits dynamic behaviour. Therefore, we focus on expanding our findings in this context and in our investigation, we consider a set of mobile agents, all of whom are initially co-located at a node that is free from any black hole threat (this initial node is termed as *home*) and also executes the same algorithm synchronously. The primary objective is to efficiently determine the location of the black hole within the network in the shortest possible time.

## 1.1 Related Work

The black hole search problem (BHS) is well-studied in the domain of mobile agents. This problem has been studied under varying underlying topologies such as rings, grids, and torus, as well as in arbitrary topology. The problem was first introduced by Dobrev et al. [14], in which they solved the BHS problem, 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 [8], rings [2, 6, 15, 16], tori [5, 24] and in graphs of arbitrary and unknown topology [7, 14]. Mainly, two variations of this problem are studied: first, when the agents are initially co-located [8] and second, when the agents are initially scattered [5, 6, 15] in the underlying network.

Most of these studies have been done for static networks; there is little knowledge about this BHS problem for networks that are not static. 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 [10, 21], dynamic tori [20], dynamic cactuses [22] and in dynamic general graphs [19]. In addition to the exploration problem, there are other problems regarding mobile agents which are as well studied in dynamic networks, such as gathering [11], compacting of oblivious agents [9], dispersion of mobile agents [1, 23]. Compared to that, the only papers regarding BHS on dynamic networks are as follows [4, 12, 13, 17]. Flocchini et al. [17]

studied the BHS problem in *carrier graphs* (which they specifically term as *subway* network), it is a special graph under 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 $\gamma + 1$ ($\gamma$ is denoted to be the minimum number of carrier stops at black holes). On the other hand, Di Luna et al. [12, 13] explored the BHS problem in a dynamic ring. In [12], they considered that the agents are initially co-located, whereas in [13], 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. Bhattacharya et al. [4] extended the BHS problem in the 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 this work, we aim to solve a similar problem where the agents are initially co-located, and we want to determine the position of the black hole using a minimum number of agents, where the underlying topology is considered to be a dynamic cactus.

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

- We establish the impossibility of finding the black hole in a dynamic cactus with 2 agents.
- 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. Further, we propose a BHS algorithm that works with 3 agents in $O(n^2)$ rounds and $O(n^2)$ moves.
- 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.

- We establish the impossibility of finding the black hole with $k + 1$ agents.
- Next, we show that any BHS algorithm with $k+2$ agents requires $\Omega(n^{1.5})$ rounds and $\Omega(n^{1.5})$ moves, respectively.
- With $2k + 3$ agents we give an improved lower bound of $\Omega(n)$ rounds and $\Omega(n)$ moves, respectively.
- Lastly with $2k + 3$ agents, we establish an upper bound of $O(kn)$ rounds and $O(k^2n)$ moves.

Table 1 summarizes the list of obtained results.

| # DE | # Agents | Moves | Rounds | Reference |
|---|---|---|---|---|
| 1 | 3 | $\Omega(n^{1.5})$ | $\Omega(n^{1.5})$ | LB (Cor 2 & Thm 3) |
| | 3 | $O(n^2)$ | $O(n^2)$ | UB (Thm 11) |
| | 4 | $\Omega(n)$ | $\Omega(n)$ | LB (Thm 5) |
| $k$ ($> 1$) | $k + 2$ | $\Omega(n^{1.5})$ | $\Omega(n^{1.5})$ | LB (Cor 7 & Thm 8) |
| | $2k + 3$ | $\Omega(n)$ | $\Omega(n)$ | LB (Thm 9) |
| | $2k + 3$ | $O(k^2n)$ | $O(kn)$ | UB (Thm 12 & Thm 13) |

Table 1: Summary of results, where $LB$, $UB$ and $DE$ represent Lower Bound, Upper Bound and Dynamic Edge, respectively.

**Organization:** The rest of the paper is organized as follows. We discuss the model and preliminaries in Section 2. Next in Section 3, we give the lower bound results for both cases of dynamicity. Further, in Section 4 we present the algorithm and its correctness for both the single and multiple dynamic edges case. Lastly, we give our concluding remarks in Section 5.

## 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 $G_i = (V, E_i)$, $E_i \subseteq E_0$. $E_0$ defines the collection of edges present in $\mathcal{G}$ at round 0, and we assume that at round 0 no missing edge exists in $\mathcal{G}$. 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 work, our graph is considered to be a dynamic cactus $\mathcal{G} = (V, \mathcal{E})$, where $|V| = n$ and $\mathcal{E} = \cup_{i=0}^{\infty} E_i$. Note that a *cactus graph* is 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 $\Delta$. 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 $0, \cdots, deg(u) - 1$. 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. 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 to all incoming agents restricts 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 black hole, are termed as *safe nodes*.

**Agent:** We consider $\mathcal{A} = \{a_1, \cdots, a_m\}$, 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 $\Delta$ 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, it can see the IDs of the other agents present at the current node, and can also communicate with them. Further, an agent, while traversing 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:

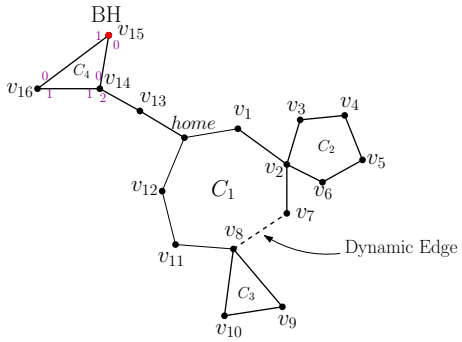


Figure 1: An example dynamic cactus graph with the edge $(v_7, v_8)$ is missing, where the port labeling is indicated in cycle $C_4$. BH represents the black hole.

*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 and their IDs as well as contains 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 $u \in \mathcal{G}$ in that case $dir \in \{0, 1, \ldots, deg(u) - 1\}$. 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.

Before defining some walks or movements, an agent performs while executing our BHS algorithms, we first define what a BHS algorithm really means.

**Definition 1** *Given a dynamic cactus $\mathcal{G}$, an algorithm $A$ 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 has 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.*

**Cautious Walk [14]:** It is a movement strategy for the agents in a network with a black hole. 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.

The strategy works as follows: suppose two agents, namely $a_1$ and $a_2$ are located at $u$, then according to this strategy the lowest ID agent among them, say $a_1$ (also can be called the explorer agent among these two agents executing this walk), travels to an adjacent node $v$ (which is 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 [12]:** From a high-level perspective, 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 back 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, $a_1$ after exploring a new node, reports back to the witness agent, which increments the hop count by one.

**Marking Walk:** This walk is a special case of *cautious* walk. An agent executing *marking* walk performs a similar movement as explained in *cautious walk*, but unlike *cautious* walk, no other agent is waiting for the explorer 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 $v$. Next, from $v$, it moves one hop towards the next unexplored node $w$ along the edge $(v, w)$. If $w$ is safe, it returns to $v$ and marks $(v, w)$ as safe and then moves to $w$.

# 3 Lower Bound Results

In this section, 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 the case where only one edge can be dynamic at any round and then, the case where at most $k$ edges can be dynamic at any round.

## 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 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 [12]. Note that the proof of Lemma 1 in [12], 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 2 (Lower bound 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 1 ([12])** *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 [12].

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 3** *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 Theorem 6 in [12], which uses Lemma 1. The following observation gives an 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 counter-clockwise, the adversary can prevent the agents from visiting further nodes inside the cycle.

**Observation 4** *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 counter-clockwise 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 to visit a node outside $U$.*

The above observation follows from Observation 1 of [12], where a *cut* is defined to be a partition of vertices of the graph in to 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 5** *Any BHS algorithm with 4 agents requires at least $\Omega(n)$ rounds and $\Omega(n)$ moves on a dynamic cactus graph $\mathcal{G}$ with 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. 2). Let the black hole be somewhere in the cycle $C'_1$, (in Fig. 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

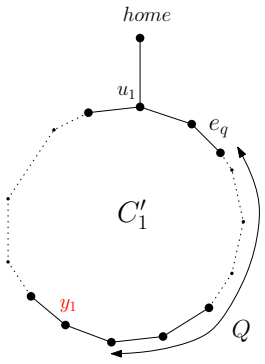


Figure 2: A cactus graph, with a cycle $C'_1$ with $n-1$ nodes, where the red node is the black hole location.

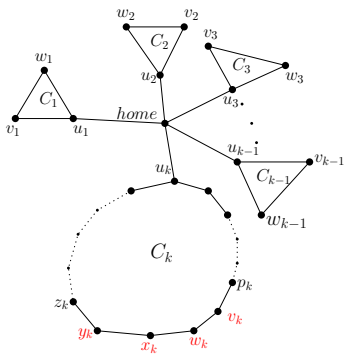


Figure 3: A cactus graph, with $k$ different cycles, where the red nodes in $C_k$ depicts possible locations of the black hole.

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 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, 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 $|C'_1|-|Q|$ $(= n-1-O(1))$ rounds, one among these three remaining agents detect 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. □

## 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 6 (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. 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 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 whiteboard, that $x_k$ is safe. Hence, in any case, the remaining $k-1$ agent 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 7 (Lower bound for $k$ dynamic edges)** *Any BHS algorithm operating on a dynamic cactus with at most $k$ dynamic edges at any round requires at least $k+2$ agents.*

The following two theorems give lower bound complexity of any BHS algorithm with $k+2$ and $2k+3$ agents, respectively.

**Theorem 8** *Any BHS algorithm operating on $\mathcal{G}$ with $k+2$ agents require at least $\Omega(n^{1.5})$ 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. 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 $\mathcal{A} = \{a_1, a_2, \cdots, a_{k+2}\}$ 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 [12], 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 9** *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 in $\mathcal{G}$.*

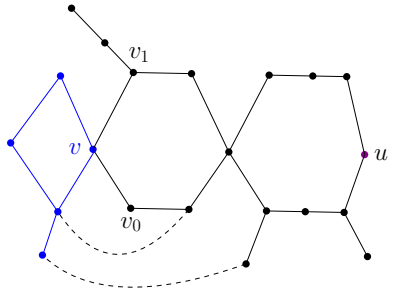


Figure 4: Represents a cactus graph, where blue vertices belong to *Half-2* and rest in *Half-1*. Dashed edges represent edge from *Half-1* to *Half-2*.

**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. 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 5, we know that it takes four agents among the $k+4$ agents to successfully locate the black hole in $\Omega(n+2-3k) = \Omega(n)$ (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 2** *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), 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 which 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 2.

**Corollary 10** *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 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 move and round complexities required by each algorithm.

## 4.1 Black Hole Search in Presence of Single Dynamic Edge

In this section, we present a BHS algorithm that works in the presence of, at most, one dynamic edge. Our algorithm requires three agents, namely $\mathcal{A} = \{a_1, a_2, a_3\}$, where without loss of generality, we assume $a_3$ to be the *Leader*, and other two agents are just termed as agents. In this section, we always call $a_3$ to be 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 uses the presence of a whiteboard present 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: $f : \{0, \cdots, deg(v) - 1\} \longrightarrow \{\perp, 0, 1\}^*$,

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

The symbol "$\circ$" refers to the concatenating 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 our algorithms SingleEdgeBHSAgent and SingleEdgeBHSLeader, we, in the following part, discuss a brief idea about the protocol an agent executes while executing their respective algorithms. Each agent (i.e., $a_1$ and $a_2$) executes the protocol of $t$-Increasing-DFS [18] while exploring the graph $\mathcal{G}$. This protocol of $t$-Increasing-DFS helps the agent to choose the next port. In addition, this protocol also helps the agent 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 2 for the value of $t$). While performing this protocol, the movement of the agents can be subdivided into the following categories: *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 $a_1$. In addition to that, the sub-graph originating from this port is not yet fully explored, i.e., it is still not marked 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.

**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*. Next, as we have defined earlier, $a_1$, $a_2$ are termed as agents and $a_3$ as the *Leader*. After being elected *Leader*, $a_3$ executes the algorithm SingleEdgeBHSLeader, whereas the agents 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 describe of our two algorithms in detail.

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 to 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 $r$ ($\geq 0$), $a_1$ 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 be $i$, 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 the port $i$ at $u$ and the 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 sub-graph originating from the port $i$ or $m$ is not yet fully explored, or $a_1$ is stuck somewhere along the sub-graph originating from the 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 $r+2$. At round $r+3$, if the edge $(u, v)$ exists, $a_1$ accompanies the *Leader* 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 at 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 has 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 still 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. [2]

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

In this situation, 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.

Before describing the pseudo codes of our algorithm, we first define the purpose of the states that $a_1$ or $a_2$ or *Leader* can attain while executing their respective algorithm.

$a_1$ 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 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. 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.

[2] 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.

**Algorithm 1:** SINGLEEDGEBHSAGENT($a_i$)

```
1  Input: {n, home}
2  States: {Cautious, Pendulum}
3  Initialize Move1 = 0, Move2 = 0, Apath_{a_i} = NULL. // Move1 variable is used for state Cautious
       and Move2 variable is used for state Pendulum.
4  In State: Cautious
5  if a missing edge is encountered then
6      Wait
7  else
8      if Move1 = 0 // Move1 = 0 implies the agent is ready to explore
9       then
10         if current node has at least one port marked as ⊥ then
11             Choose min_{j∈deg(current−node)}{j} such that f(j) = ⊥
12             Traverse along this port. Update f(j) = 0 ∘ {a_i} at the current node and f(m) = 0 ∘ {a_i} at
                   the adjacent port. // j, m are the outgoing and incoming port of the edge
                   connecting current node and adjacent node, respectively.
13             Update Move1 = 1
14         else
15             Backtrack while accompanying the Leader, along a path previously traversed.
16     else if Move1 = 1 // Move1 = 1 implies the agent has reached an unexplored node.
17      then
18         if a missing edge appears then
19             Wait, until the missing edge reappears.
20         else
21             Move along the port m and set Move1 = 2. // Move1 = 2 means that the agent has
                   started backtracking after exploring a new node.
22     else
23         if current node does not have Leader then
24             Follow Last.Leader.
25         else
26             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.
```

**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.

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 2:** 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 $W_{time1} = 0$.
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 $Apath_{a_m}$ and $Alen_{a_m}$ 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 $Wtime1 = 0$.
    **else**
        Instruct the agent at current node to perform state **Cautious**. Change to state **Assign** and set $Wtime1 = 0$.
**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 $Wtime1 = 0$.
    **else**
        If previous state was **Fail-to-Report**, then continue performing **Fail-to-Report**. Else change to state **Fail-to-Report**.

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

```
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 · (Alen_{a_m} + Llen_{a_m} + 1) rounds, while performing pendulum walk then
    if Wtime1 > 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} ∪ (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} ∪ (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
        Wtime1 = Wtime1 + 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 · (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 Σ_{i=1}^{2}(2 · (Alen_{a_i} + Llen_{a_i} + 1))
  round then
    Change to state Fail-to-Report-Movement.
```

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

```
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∈{1,2}}(2 · (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∈{1,2}}(2 · (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∈{1,2}}(2 · (Alen_{a_i} + Llen_{a_i} + 1)).
    if the agent is found then
        Set Alen_{a_m} = Llen_{a_m} = 0 and Apath_{a_m} = Lpath_{a_m} = NULL and instruct a_m to change to
          state Pendulum.// a_m be that agent found.
        Change to state Fail-to-Report.
    else
        Conclude the last visited node of the agent with min_{i∈{1,2}}(2 · (Alen_{a_i} + Llen_{a_i} + 1)) is the black
          hole.
```

*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 currently 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*, on the other hand, 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 1 and Algorithm 2, respectively.

### 4.1.1 Correctness and Complexity

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

**Lemma 3** *Our algorithms* SingleEdgeBHSAgent *and* SingleEdgeBHSLeader *on a dynamic cactus graph $\mathcal{G}$ with at most one edge being dynamic at any round, ensures that between $a_1$, $a_2$ and Leader, at most 2 among them can be stuck or waiting due to 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 *cautious* walk, or when it is moving towards an agent $a_1$ (say) which fails to report, while performing *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 2). 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 found stuck or waiting 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 discus 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 $a_1$.

So, 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, 2 among them can be stuck or waiting due to a missing edge. □

**Lemma 4** *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 both the 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 which has performed *cautious* walk. So, at round $r$, whenever it has chosen to travel through the edge $(u, v)$, it has found 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 5** *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 1 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 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 1, cannot explore any cycle infinitely according to the earlier argument. So, this shows that the *Leader* also can never explore any cycle infinitely. □

**Lemma 6** *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 [3] 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 1, not only explore each port in $\mathcal{G}$ but also stores the ports that it visits. 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, it may be visited, but during which the sequence $Apath_{a_i}$ is not updated. Each node in $\mathcal{G}$ can have at most $\Delta$ 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 [18], an agent with $O(n \log \Delta)$ bits of memory is sufficient to explore any static graph with diameter $n$ and maximum degree $\Delta$. So, $O(n\Delta \log \Delta)$ bits are more than sufficient for an agent to explore $\mathcal{G}$. Also, Lemma 4 states that eventually, *Leader* and one among $a_1$ or $a_2$ remain stuck (or wait) 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 until (which is indeed a static graph, as either $a_1$ or $a_2$ and *Leader* occupy both sides of a missing edge) it gets destroyed by the black hole or the *Leader* terminates the algorithm by either detecting the position of the black hole, or the path where visiting the last node, indicates the black hole node. □

**Lemma 7** *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

[3] 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 $f(j) = 0 \circ A \setminus \{a_1\}$

destroyed by the black hole, as it never explores an unexplored node while accompanying with 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 8** *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 9** *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 (refer to Algorithm 1). Hence, we have the following cases based on the movement strategies followed by $a_1$ and $a_2$ while executing Algorithm 1.

- 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 it is 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 performing *pendulum* walk, is discussed in the next case. Let us consider (for case (1)) that $a_2$ is performing *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 both still are 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 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 the possibilities, and in each case, we have proved our claim. □

**Lemma 10** *The Leader executing* SingleEdgeBHSLeader *correctly terminates the algorithm.*

**Proof:** It may be noted that according to our Definition 1, the surviving agent can terminate the algorithm when it either knows the exact black hole node or knows the path that it needs to visit to determine the black hole node. In our BHS algorithm, the *Leader* is the one that 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, so 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 traveled to reach $u$, so accordingly, the *Leader* locates the black hole and terminates the algorithm. Otherwise, if the edge $(u, v)$ is missing, at that 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 concludes 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 correctly terminates the algorithm. □

**Theorem 11** *The agents $a_1$ and $a_2$ executing* SingleEdgeBHSAgent *and the Leader executing* SingleEdgeBHSLeader *correctly terminates 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 8 and 9 state that our Algorithms 1 and 2, 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 at each round. This implies that the agents following Algorithm 1 and the *Leader* following Algorithm 2 solves the BHS problem within $O(n^2)$ moves. □

## 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 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 $\mathcal{A} = \{a_1, a_2, \ldots, a_{2k+3}\}$ 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 storage, 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 same as the function $f$, defined in Section 4.1. The function $g_2$ is defined as follows, $g_2 : \{0, 1, \ldots, deg(u) - 1\} \to \{\perp, 0, 1\}$,

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

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 execute MultiEdgeBHS. Similar to Algorithm 1, 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 some agent that has been performing the action *explore*.

Now, we provide a brief idea 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 exploring continues until the agent gets destroyed by the black hole, or it 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 *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 2). Next, while traversing on $\mathcal{G}$, if an agent discovers a missing edge along its path of movement, and currently, no agent is waiting for that edge, this agent waits until the edge reappears.

**Algorithm 3:** MULTIEDGEBHS($a_i$)

Input = $\{n, home\}$
States: {**Initial**,**Cautious**, **Marking**, **MissingCautious-Wait**, **Cautious-Wait**}
Initially set $Move1 = W_{time1} = Move2 = W_{time3} = 0$
In State: **Initial**
**if** *the current node has more than one agent* **then**
    Change to state **Cautious**.
**else**
    Change to state **Marking**.
In State: **Cautious**.
**if** *a port exists at the current node marked as* $(\perp, \perp)$ **then**
    **if** $Move1 = 0$ **then**
        **if** *the edge with respect to lowest such port exists* **then**
            **if** $a_i$ *is the lowest ID agent at the current node, in state* ***Cautious*** **then**
                Set $Move1 = 2$ and traverse along this port, while marking $(0 \circ \{a_i\}, 0)$ at both the ports of the edge to this adjacent node, update $Apath_{a_i} = Apath_{a_i} \cup (j, m)$.// $j-$`th port is the lowest such outgoing port at the current node,` $m-$`th port is the incoming port of the adjacent node`
            **else**
                Set $W_{time2} = 0$ and change to state **Cautious-Wait**.
        **else**
            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**.
    **else if** $Move1 = 1$ **then**
        **if** *there exists an agent with* $Move1 = 0$ **then**
            Set $Move1 = 0$ and change to state **Cautious**. Also set $W_{time2} = 0$.
    **else if** $Move1 = 2$ **then**
        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.
    **else**
        Move along port $j$ and then set $Move1 = 0$ and change state to **Initial**. Otherwise, if edge is missing then wait.
**else if** *a port* $j$ *exists marked with* $(0 \circ A, 0)$ *exists* **then**
    **if** $Move1 = 0$ **then**
        **if** *the edge is missing and no agent is waiting* **then**
            If $a_i$ is the lowest ID agent with $Move1 = 0$, then wait. Otherwise, change to state **Initial**.
        **else if** *the edge exists and no agent is waiting* **then**
            **if** $W_{time1} > 1$ **then**
                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.
            **else**
                $W_{time1} = W_{time1} + 1$.
        **else**
            Choose a different port. Otherwise, if no port exists to choose from, then backtrack and change to state **Initial**.
    **else if** $Move1 = 1$ **then**
        **if** *there exists an agent with* $Move1 = 0$ **then**
            Set $Move1 = 0$ and change to state **Cautious** also set $W_{time2} = 0$.
    **else if** $Move1 = 2$ **then**
        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.
    **else**
        Move along port $j$ and then set $Move1 = 0$ and change state to **Initial**. Otherwise, if edge is missing, then wait.

**else if** *a port $j$ exists marked as* $(0 \circ A, 1)$ **then**
  **if** $Move1 = 0$ **then**
    **if** *the set $A$ contains ID of an agent present at current node which is not waiting for any missing edge* **then**
      Choose a different port or backtrack and change to state **Initial**.
    **else**
      **if** *the edge is missing and no agent is waiting for this edge* **then**
        If $a_i$ is the lowest ID agent, then wait. Otherwise change to state **Initial**.
      **else if** *the edge exists but no agent is waiting for this edge* **then**
        Move along this port and update $g_1(j) = g_1(m) = 0 \circ (A \cup \{a_i\})$ and $Apath_{a_i} = Apath_{a_i} \cup (j, m)$, set $Move1 = 0$ then change to state **Initial**.
      **else**
        Choose a different port and if no port exists to choose then backtrack and change to state **Initial**.
  **else if** $Move1 = 1$ **then**
    If another agent with $Move1 = 0$ is present, then set $Move1 = 0$ and change to state **Cautious** also set $W_{time2} = 0$.
  **else if** $Move1 = 2$ **then**
    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.
  **else**
    Move along port $j$ and then set $Move1 = 0$ and change state to **Initial**. Otherwise, if the edge is missing then wait.
**else**
  **if** $Move1 = 0$ **then**
    Backtrack to an already traversed node, and change to state **Initial**.
  **else if** $Move1 = 1$ **then**
    If there exists another agent with $Move1 = 0$, then set $Move1 = 0$ and change to state **Cautious** and set $W_{time2} = 0$.
  **else if** $Move1 = 2$ **then**
    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.
  **else**
    Move along port $j$ and then set $Move1 = 0$ and change state to **Initial**. Otherwise, if the edge is missing, then wait.

In State: **MissingCautious-Wait**
**if** *the edge is missing* **then**
  Wait.
**else**
  Change state to **Initial**

In State: **Cautious-Wait**
**if** $W_{time2} > 1$ **then**
  **if** *the edge exists* **then**
    **if** *there is an agent which returned along port $j$ with $Move2 = 3$* **then**
      **if** *the edge exists* **then**
        Move along this port update $g_1(j) = g_1(m) = 0 \circ (A \cup \{a_i\})$ and $Apath_{a_i} = Apath_{a_i} \cup (j, m)$, set $Move1 = 1$ and change to state **Initial**.
      **else**
        Change to state **Initial**.
    **else**
      Conclude the adjacent node to be black hole.
  **else**
    If the agent is of lowest ID among all the agents which are in state **Cautious-Wait**, then wait. Otherwise change to state **Initial**.
**else**
  $W_{time2} = W_{time2} + 1$

In State: **Marking**
**if** *a port exists at current node marked as* $(\perp, \perp)$ **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 $(0 \circ \{a_i\}, 0)$ at both the outgoing port $j$ and incoming port $m$, update $Apath_{a_i} = Apath_{a_i} \cup (j, m)$.
    **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 $g_2(m) = g_2(j) = 1$, 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* $(0 \circ A, 0)$ **then**
  **if** $Move2 = 0$ **then**
    **if** *the edge exists* **then**
      **if** $W_{time3} > 1$ **then**
        If $g_2$ 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 $g_2(m) = g_2(j) = 1$, 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* $(0 \circ A, 1)$ **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 $g_2(j) = g_2(m) = 0 \circ (A \cup \{a_i\})$, $Apath_{a_i} = Apath_{a_i} \cup (j, m)$ 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 $g_2(m) = g_2(j) = 1$, 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 $g_2(m) = g_2(j) = 1$, 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.

Otherwise, if some agent is already waiting for the same missing edge to reappear, then it simply ignores this port and moves along some 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 still 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[4] 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 $Move1$ 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**.

If the agent has $Move1$ value to be 1, then that means the agent has earlier performed *cautious* walk, and it 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 is checking if the explorer agent is also present at the current node; if so, then it changes $Move1$ 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 and sets $Move2 = 3$. 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,

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

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 the agent with $Move1 = 0$ waits 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 terminates 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 $Move1$ value 0, and the edge with respect to the lowest such port (i.e., the port marked as $(0 \circ A, 1)$) is missing, also 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 state **Initial**. 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 which is also trying to traverse through this edge and has $Move1$ value 0, then all these agents with $Move1 = 0$, those 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 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 for 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 among the follower agents performing *cautious* walk, where the explorer agent has traversed along the port, say $j$, and is 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 may be noted that this updation is done on 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 state **Initial**.

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 3.

### 4.2.1 Correctness and Complexity

In this section, we analyze the correctness and complexity of our algorithm MultiEdgeBHS.

**Lemma 11** *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 performs, 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 3). 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 12** *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 10, 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 3, 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 2 rounds, either of these agents at $v_0$ or $v_1$ can successfully detect the black hole location without destroying more agents. Hence, this shows that our algorithm ensures that at most 2 agents can be destroyed by the black hole. □

**Lemma 13** *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 3 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 14** MULTIEDGEBHS *ensures that any agent which 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, so by Lemma 11, 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 [18], $O(n \log \Delta)$ bits of memory are required to explore any static graph of diameter $n$ and maximum degree $\Delta$. 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 3, 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 and unless it gets destroyed by the black hole or detects it. □

**Theorem 12** *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 11, 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 12, it is shown that at most 2 agents can get destroyed by a black hole. Also, Lemma 14 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 13** *A set of* $2k + 3$ *agents executing* MultiEdgeBHS*, locates the black hole in a dynamic cactus graph* $\mathcal{G}$ *within* $O(kn)$ *rounds and* $O(k^2 n)$ *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 $k$ many edges can be dynamic at any round. This shows that as per Theorem 12, to execute MultiEdgeBHS on $\mathcal{G}$, $2k + 3$ agents are required.

We first analyze the round complexity that our algorithm requires in order to explore the cycle $C_i \in \mathcal{G}$. Let us suppose $\alpha$ 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 $u$, so 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 3, 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). In order 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 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 $\alpha$ 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 done 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 [26]). Moreover, it may be noted 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 1** *There is a fundamental difference between the algorithm presented in Section 4.1 and that of the one presented in Section 4.2. It is because the algorithm presented for single dynamic edge case is optimal in terms of agents. So, there exist certain 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, then in that case as well the algorithm terminates.*

*On the other hand, the algorithm, explained for multiple dynamic edges case, does not face these issues. As there are a sufficient number of agents present, so in this case, 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.*

# 5 Conclusion

In this paper, 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 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 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. An interesting future work may be to design an optimal algorithm in terms of a number of agents when the underlying graph has at most $k$ dynamic edges. Further, it will be interesting to find an optimal algorithm in terms of complexity in both cases of dynamicity.

## Acknowledgement

We extend our gratitude to the anonymous reviewers and editors for their valuable time and in detail suggestions, which has significantly improved the paper. We also thank Government of India for supporting Adri Bhattacharya through CSIR, Grant Number: 09/731(0178)/2020-EMR-I.

## References

[1] A. Agarwalla, J. Augustine, W. K. Moses Jr, S. K. Madhav, and A. K. 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] B. Balamohan, P. Flocchini, A. Miri, and N. Santoro. Time optimal algorithms for black hole search in rings. *Discrete Mathematics, Algorithms and Applications*, 3(04):457–471, 2011.

[3] A. Bhattacharya, G. F. Italiano, and P. S. Mandal. Black hole search in dynamic cactus graph. In *International Conference and Workshops on Algorithms and Computation*, pages 288–303. Springer, 2024.

[4] A. Bhattacharya, G. F. Italiano, and P. S. 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.

[5] J. Chalopin, S. Das, A. Labourel, and E. 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.

[6] J. Chalopin, S. Das, A. Labourel, and E. Markou. Tight bounds for black hole search with scattered agents in synchronous rings. *Theoretical Computer Science*, 509:70–85, 2013.

[7] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71(2-3):229–242, 2006.

[8] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in synchronous tree networks. *Comb. Probab. Comput.*, 16(4):595–619, 2007.

[9] S. Das, G. A. Di Luna, D. Mazzei, and G. Prencipe. Compacting oblivious agents on dynamic rings. *PeerJ Computer Science*, 7, 2021.

[10] G. Di Luna, S. Dobrev, P. Flocchini, and N. Santoro. Distributed exploration of dynamic rings. *Distributed Computing*, 33:41–67, 2020.

[11] G. A. Di Luna, P. Flocchini, L. Pagli, G. Prencipe, N. Santoro, and G. Viglietta. Gathering in dynamic rings. *theoretical computer science*, 811:79–98, 2020.

[12] G. A. Di Luna, P. Flocchini, G. Prencipe, and N. Santoro. Black hole search in dynamic rings. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 987–997. IEEE, 2021.

[13] G. A. Di Luna, P. Flocchini, G. Prencipe, and N. 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.

[14] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. *Distributed Computing*, 19, 2006.

[15] S. Dobrev, N. Santoro, and W. 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.

[16] S. Dobrev, N. Santoro, and W. 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.

[17] P. Flocchini, M. Kellett, P. C. Mason, and N. Santoro. Searching for black holes in subways. *Theory of Computing Systems*, 50:158–184, 2012.

[18] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.

[19] T. Gotoh, P. Flocchini, T. Masuzawa, and N. Santoro. Exploration of dynamic networks: tight bounds on the number of agents. *Journal of Computer and System Sciences*, 122:1–18, 2021.

[20] T. Gotoh, Y. Sudo, F. Ooshita, H. Kakugawa, and T. Masuzawa. Exploration of dynamic tori by multiple agents. *Theoretical Computer Science*, 850:202–220, 2021.

[21] T. Gotoh, Y. Sudo, F. Ooshita, and T. Masuzawa. Dynamic ring exploration with (h, s) view. *Algorithms*, 13(6):141, 2020.

[22] D. Ilcinkas and A. M. Wade. Exploration of dynamic cactuses with sub-logarithmic overhead. *Theory of Computing Systems*, 65:257–273, 2021.

[23] A. D. Kshemkalyani, A. R. Molla, and G. 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.

[24] E. Markou and M. 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.

[25] E. Markou and W. Shi. Dangerous graphs. *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 455–515, 2019.

[26] D. B. West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.