

Black Hole Search in Dynamic Cactus Graph

Adri Bhattacharya^{1*}, Giuseppe F. Italiano², and Partha Sarathi Mandal^{1**}

¹ Indian Institute of Technology Guwahati, India
 {a.bhattacharya, psm}@iitg.ac.in

² Luiss University, Rome, Italy
 gitaliano@luiss.it

Abstract. We study the problem of black hole search by a set of mobile agents, where the underlying graph is a dynamic cactus. A black hole is a dangerous vertex in the graph that eliminates any visiting agent without leaving any trace behind. Key parameters that dictate the complexity of finding the black hole include: the number of agents required (termed as *size*), the number of moves performed by the agents in order to determine the black hole location (termed as *move*) and the *time* (or round) taken to terminate. This problem has already been studied where the underlying graph is a dynamic ring [9]. In this paper, we extend the same problem to a dynamic cactus. We introduce two categories of dynamicity, but still the underlying graph needs to be connected: first, we examine the scenario where, at most, one dynamic edge can disappear or reappear at any round. Secondly, we consider the problem for at most k dynamic edges. In both scenarios, we establish lower and upper bounds for the necessary number of agents, moves and rounds.

Keywords: Black hole search, Dynamic cactus graph, Dynamic networks · Time-varying graphs · Mobile agents computing

1 Introduction

We study the black hole search problem (also termed as BHS) in a dynamic cactus graph, where edges can appear and disappear i.e., goes missing over time so the underlying graph remains connected. More precisely, the network is a synchronous cactus graph where one of the vertices (or nodes) is a malicious node that eliminates visiting agents without any trace of their existence upon arrival on such nodes; that node is termed as *Black Hole* [10]. This scenario frequently arises within networked systems, particularly in situations requiring the safeguarding of agents from potential host attacks. Presently, apart from the research paper concerning ring networks [9], there exists limited knowledge regarding this phenomenon when the network exhibits dynamic characteristics. Therefore, the focus of our study is to expand upon our findings in this context.

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

** This work was done while Partha Sarathi Mandal was in the position of Visiting Professor at Luiss University, Rome, Italy

In our investigation, we consider a collection of mobile agents, all of whom execute the same algorithm synchronously. Initially, these agents are positioned at a node that is confirmed to be free from any black hole threat; these nodes are referred to as ‘safe nodes’. The primary objective is to efficiently determine the location of the black hole within the network in the shortest possible time while ensuring at least one agent survives and possesses knowledge of the black hole’s whereabouts.

Related Work. The black hole search problem is well-studied in varying underlying topologies such as rings, grids, torus, etc. This problem is first introduced by Dobrev et al. [10], where they solved in static arbitrary topology. Further, they give tight bounds on the number of agents and provide cost complexity of the size-optimal solution. After this seminal paper, there has been a plethora of work done in this domain under different graph classes such as trees [5], rings [2,11,12], tori [3,22] and in graphs of arbitrary topology [4,10]. Moreover, two variations of this problem are mainly studied in the literature. First, when the agents are initially co-located [5] and second, when the agents are initially scattered [3,11] in the underlying network. Now, in all the above literature, the study has been performed when the underlying graph is static.

While most of the study has been done on static networks, very little literature is known about black hole search especially in dynamic graphs. The study of mobile agents in dynamic graphs is a fairly new area of research. Previously, the problem of exploration has been studied in dynamic rings [7,17,21], torus [16], cactuses [18] and in general graphs [15]. In addition to exploration, there are other problems related to mobile agents studied in dynamic networks such as, gathering [8], compacting of oblivious agents [6], dispersion of mobile agents [1,19]. Further, Flocchini et al. [13] studied the black hole search problem in a different class of dynamic graphs, defined as periodically varying graphs, they showed the minimum number of agents required to solve this problem is $\gamma + 1$, where γ is the minimum number of carrier stops at black holes. Di Luna et al. [9], studied the black hole search problem in a dynamic ring, where they established lower bounds and give size-optimal algorithms in terms of agents, moves and rounds in two communication models. In this paper, we aim to solve a similar problem, where we want to determine the position of a black hole with the least number of agents, but in our case, we have considered the underlying topology to be a dynamic cactus graph.

Our Contributions. We obtain the following results when the cactus graph has at most one dynamic edge at any round.

- Establish the impossibility to find a black hole in a dynamic cactus with 2 agents.
- With 3 agents we establish lower bound of $\Omega(n^{1.5})$ rounds, $\Omega(n^{1.5})$ moves, and we also establish upper bound of $O(n^2)$ rounds and $O(n^2)$ moves.
- With 4 agent improved lower bounds are $\Omega(n)$ rounds and $\Omega(n)$ moves.

Next, when the cactus graph has at most k ($k > 1$) dynamic edge at any round.

- Establish the impossibility to find the black hole with $k + 1$ agents.
- With $k + 2$ agents we establish lower bound of $\Omega((n + 2 - 3k)^{1.5})$ rounds and $\Omega((n + 2 - 3k)^{1.5} + 2k)$ moves.
- With $2k + 3$ agents improved lower bounds are $\Omega(n + 2 - 3k)$ rounds, $\Omega(n + 2 - k)$ moves, and we establish an upper bound of $O(kn)$ rounds and $O(k^2n)$ moves.

# DE	# Agents	Moves	Rounds	
1	3	$\Omega(n^{1.5})$	$\Omega(n^{1.5})$	LB (Cor 1 & Thm 2)
	3	$O(n^2)$	$O(n^2)$	UB (Thm 9)
	4	$\Omega(n)$	$\Omega(n)$	LB (Thm 3)
k	$k + 2$	$\Omega((n + 2 - 3k)^{1.5} + 2k)$	$\Omega((n + 2 - 3k)^{1.5})$	LB (Cor 2 & Thm 6)
	$2k + 3$	$\Omega(n + 2 - k)$	$\Omega(n + 2 - 3k)$	LB (Thm 7)
	$2k + 3$	$O(k^2n)$	$O(kn)$	UB (Thm 10 & Thm 11)

Table 1: Summary of Results ($k > 1$), where LB , UB and DE represent lower bound, upper bound and dynamic edge, respectively.

Organization: Rest of the paper is organized as follows. In section 2, we discuss the model and preliminaries. Section 3, we give the lower bounds. In section 4, we present the algorithm and its correctness for both the single and multiple dynamic edge cases and finally concluding in section 5.

2 Model and Preliminaries

Dynamic Graph Model: We adapt the synchronous dynamic network model by Kuhn et al. [20] to define our dynamic cactus graph \mathcal{G} . The vertices (or nodes) in \mathcal{G} are static, whereas the edges are dynamic i.e., the edges can disappear (or in other terms go missing) and reappear at any round. The dynamicity of the edges holds as long as the graph is connected. The dynamic cactus graph $\mathcal{G} = (V, \mathcal{E})$ is defined as a collection of undirected cactus graphs $\langle G_0, G_1, \dots, G_r, \dots \rangle$, where $G_r = (V, E_r)$ is the graph at round r , $|V| = n$ and $\mathcal{E} = \cup_{r=0}^{\infty} E_r$, where $|E_r| = m_r$ denotes the number of edges in G_r . The adversary maintains the dynamicity of \mathcal{G} , by disappearing or reappearing certain edges at any round r such that the underlying graph is connected. This model of dynamic networks is studied in [20] and is termed as a 1-interval connected network. The degree of a node $u \in \mathcal{G}$ is denoted by $\deg(u)$, in other words, $\deg(u)$ denotes the degree of the node u in G_0 . The maximum degree of the graph \mathcal{G} is denoted as Δ . The vertices (or nodes) in \mathcal{G} are anonymous, i.e., they are unlabelled, although, the edges are labelled, an edge incident to u is labelled via the port numbers $0, \dots, \deg(u) - 1$. The ports are labelled in ascending order along the counter-clockwise direction, where a port with port number i denotes the port number

corresponding to the i -th incident edge at u in the counter-clockwise direction. Any edge $e = (u, v)$ is labelled by two ports, one among them is incident to u and the other incident to v , they have no relation in common (refer to Fig. 1). Any number of agents can pass through an edge concurrently. Each node in \mathcal{G} has local storage in the form of a *whiteboard*, where the size of the whiteboard at a node $v \in V$ is $O(deg(v)(\log deg(v) + k \log k))$, where $deg(v)$ is the degree of v and k is the atmost number of missing edge. The whiteboard is essential to store the list of port numbers attached to the node. Any visiting agent can read and/or write travel information corresponding to port numbers. Fair mutual exclusion to all incoming agents restricts access to the whiteboard. The network \mathcal{G} contains a malicious node termed as *black hole*, which eliminates any incoming agent without leaving any trace of its existence.

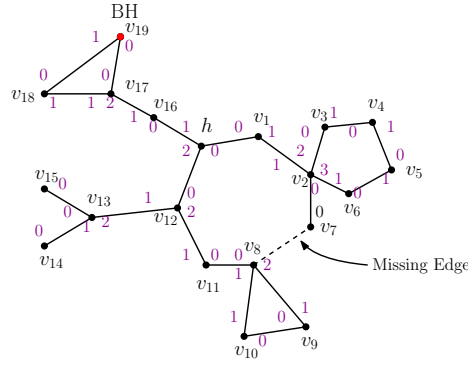


Fig. 1: A port labelled Cactus Graph is depicted where an edge (v_7, v_8) is missing.

Agent: Let $\mathcal{A} = \{a_1, \dots, a_m\}$ be a set of $m \leq n$ agents, they are initially co-located at a safe node termed as *home*. Each agent has a distinct Id of size $\lceil \log m \rceil$ bits taken from the set $[1, m]$ where, each agent is a t -state automata, with local storage of $O(n \log \Delta)$ bits of memory, where $t \geq \alpha n \log \Delta$ and α is any positive integer. The agents visiting a node knows the degree of that node and also can determine the missing edges at that particular node, based on the whiteboard information. The agent while moving from u to v along the edge e knows the port along which it left u and the port along which it entered v . Further, the agents can see the *Ids* of other agents residing at the same node and can communicate with them.

Round: The agents operate in *synchronous* rounds, where each agent gets activated in each round. At any time an agent $a_i \in \mathcal{A}$ gets activated, and they perform the following steps in a round: “Communicate-Compute-Move” (CCM), while it is active. The steps are defined as follows:

- *Communication*: Agents can communicate among themselves when they are at the same node at the same time. They can communicate via whiteboard as well. In this step, agents can also observe their memory.
- *Compute*: An agent based on the gathered information, local snapshot (i.e., information gathered on whether any other agent is present at the current node), internal memory and contents of a whiteboard, either decides to stay or choose the port number in case it decides to move.
- *Move*: The agent moves along the chosen port to a neighboring port, if it decides to move. While it starts to move, the agent writes the information in its memory and also writes on the whiteboard of its current node.

An agent takes one unit of time to move from a node u to another node v following the edge $e = (u, v)$.

Time and Move Complexity: Since the agents operate in synchronous rounds, each agent gets activated at each round to perform one CCM 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 execution of the algorithm.

Configuration: We define C_r to be the *configuration* at round r which holds the following information: the contents in the whiteboard at each node, the contents of the memory of each agent and the locations of the agent at the start of round r .

So, C_0 is the initial configuration at the start of an algorithm \mathcal{H} , whereas C_r is the configuration obtained from C_{r-1} after execution of the algorithm \mathcal{H} at round r .

We recall next the notions of *cautious*, *pendulum* and *pebble* walk.

Cautious Walk [9]. This is a movement strategy for agents in a network with a black hole, which ensures that at least two agents can travel together so that only one of them may be destroyed by the black hole, while the other survives. This is done as follows. If two agents a_1 and a_2 are at a safe node u , a_1 goes to an adjacent node v and returns, while a_2 waits at u . If a_1 returns to node u at the next round, then both a_1 and a_2 can safely travel together to node v . Otherwise, if at the next round, a_1 has not returned to node u , a_2 knows that either edge (u, v) disappeared (which can be discovered by looking at the corresponding port) or that v contains the black hole.

Pendulum Walk [9]. From a high-level perspective, an agent a_1 travels back and forth, increasing the number of hops at each movement, and always reports back to another agent a_2 (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 decides to move one hop along the edge (u, v) and reaches a node v . If v is safe, then a_1 returns back to u to inform a_2 that v is safe. Next, a_1 decides to move two hops along the edge (u, v) and (v, w) , thus reaching node w . If w is safe, then a_1 returns back to u via v . In general, a_1 in each movement incre-

ments the hop count by one, and at some round reaches a node z along the path $P = u \rightarrow v \rightarrow w \rightarrow \dots \rightarrow z$. If z is safe, then it returns to u .

Pebble Walk: This walk is a special case of *pendulum* walk. In this case, as well as an agent a_1 travels to and fro, but unlike *pendulum* walk whenever a_1 reaches a new node it does not report back to another agent (or witness agent).

More precisely, let us consider an agent a_1 is currently at a node u . Now, a_1 decides to move one hop to an adjacent node v using the edge (u, v) . If v is safe, then it returns back to u . Further, a_1 again reaches v and decides to move another one hop from v to a new node w , following a similar strategy. So, the path followed by a_1 to reach w is denoted as $P = u \rightarrow v \rightarrow u \rightarrow v \rightarrow w$. In general, a_1 moves one hop to a new node by similar to and fro movement from the last explored node.

3 Lower Bound Results

In this section, we first study the lower bound on the number of agents, move and round complexities required to solve the BHS problem when at most one edge can disappear or appear in a round. Then we generalize this idea for the case when at most k edges can disappear or appear in a round such that the underlying graph remains connected.

3.1 Lower Bound results on Single Dynamic Edge

Here, we present all the results related to a dynamic cactus graph when at most one edge missing at any round.

Theorem 1 (Impossibility for single dynamic edge). *Given a dynamic cactus graph \mathcal{G} of size $n > 3$ with at most one edge which can disappear at any round such that the underlying graph is connected. Let the agents have the knowledge 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 locate the black hole position. The impossibility holds even if the nodes are equipped with a whiteboard.*

The above theorem is a consequence of Lemma 1 in [9].

Corollary 1 (Lower bound for single dynamic edge). *To locate the black hole in a dynamic cactus graph \mathcal{G} with at most one edge missing at any round, any algorithm requires at least 3 agents to solve the black hole search problem in \mathcal{G} .*

Lemma 1 ([9]). *If an algorithm solves black hole search 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 other node 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.

This lemma holds even if the nodes are equipped with whiteboards.

In the next theorem, we give a lower bound on the move and round complexity required by any algorithm in order to solve the black hole search problem in a dynamic cactus.

Theorem 2. *Given a dynamic cactus graph \mathcal{G} , with at most one missing edge at any round. In the presence of a whiteboard, any algorithm \mathcal{H} solves the black hole search problem with three agents in $\Omega(n^{1.5})$ rounds and $\Omega(n^{1.5})$ moves, when the agents have distinct IDs and they are co-located.*

The above theorem is a consequence of theorem 6 in [9]. The next theorem, gives an improved lower bound on the move and round complexity when 4 agents try to locate the black hole instead of 3.

Theorem 3. *Given a dynamic cactus graph \mathcal{G} , with at most one dynamic edge at any round. In the presence of whiteboard, any algorithm \mathcal{H} solves the black hole search problem with four agents in $\Omega(n)$ rounds and $\Omega(n)$ moves, when the agents have distinct IDs and they are co-located.*

Proof. Let a_1, a_2, a_3 and a_4 are the four agents trying to find the black hole in \mathcal{G} . Consider C_k be a cycle in \mathcal{G} (refer to Fig. 3) and y_k be the black hole node. Now, suppose a_1 enters the black hole and Q be the set of consecutive nodes along which a_1 , before entering the black hole has written its exact location on the whiteboard, so whenever an agent visits any of these nodes, it knows the exact location of the black hole and terminates. Let e_q be the edge separating the black hole and Q with the rest of the graph. Now, the adversary has the ability to restrict any agent from visiting the set of nodes in Q by removing e_q . On the contrary, at any round, if no agent is trying to visit a node in Q , whereas establishing the black hole location along the counter-clockwise direction, then the adversary can restrict the agent visiting such a node by removing an edge, such as e_{cc} . So, the only possibility is while an agent always tries to visit a node in Q , the remaining two agents can correctly locate the black hole location while traversing a counter-clockwise direction in C_k , in at least n rounds. Moreover, since at any round, a constant number of agents are moving. Hence, in order to successfully locate the black hole location with 4 agents, any algorithm requires $\Omega(n)$ rounds and $\Omega(n)$ moves. \square

The next observation gives a brief idea about the movement of the agents on a cycle inside a dynamic cactus graph. It states that, when a single agent is trying to explore any unexplored cycle, the adversary has the power to confine the agent on any single edge of the cycle. Moreover, in case of multiple agents trying to explore a cycle inside a cactus graph, but their movement is along one direction, i.e., either clockwise or counter-clockwise, then also the adversary has the power to prevent the team of agents from visiting further unexplored nodes.

Observation 4 *Given a dynamic cactus graph \mathcal{G} , and a cut U (with $|U| > 1$) of its footprint connected by edges e_1 in the clockwise direction and e_2 in the counter clockwise direction to the nodes in $V \setminus U$. If we assume that all the agents at round r are at U , and there is no agent which tries to cross to $V \setminus U$ along e_1 and an agent tries to cross along e_2 , then the adversary may prevent agents to visit nodes outside U .*

The above observation follows from observation 1 of [9]. The next lemma follows from the structural property of a cactus graph.

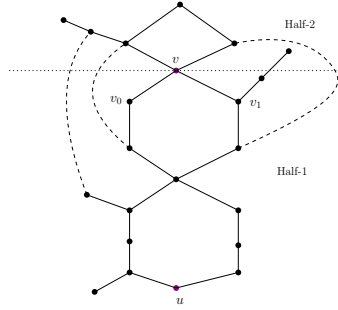


Fig. 2: Represents that any u to v path either passes through v_0 or v_1 .

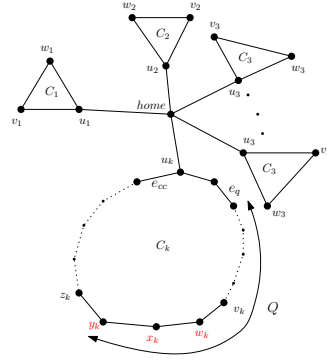


Fig. 3: A cactus graph, with k different cycles, where red nodes in C_k depicts possible location of the black hole.

Lemma 2. *Consider three consecutive nodes $\{v_0, v, v_1\}$ in a cactus graph \mathcal{G} , then any path from u to v in \mathcal{G} must pass through either v_0 or v_1 .*

Proof. We prove the above claim by contradiction. Suppose there exists a u to v path which neither passes through v_0 nor v_1 , so in order to have an alternate path which does not pass through v_0 or v_1 , implies that there must be at least one edge or a path passing from half-1 to half-2 (refer to Fig. 2), where we define half-1 to be the subgraph on and above the horizontal half-line passing through v , whereas half-2 is the subgraph below the horizontal half-line passing through v . Now, the presence of such an edge or path, implies that there is at most one common edge between two cycles, and this violates the characteristic of a cactus graph. Hence, there cannot be any such u to v path which neither passes through v_0 nor v_1 . \square

3.2 Lower bound results for multiple dynamic edges

Here, we present all the lower bound results for a dynamic cactus graph G , when at most k edges are missing at any round.

Theorem 5 (Impossibility for multiple dynamic edges). *Given, a dynamic cactus graph \mathcal{G} with at most k dynamic edges at any round. It is impossible for $k + 1$ agents, to successfully locate the black hole position, regardless of the knowledge of n , k and the presence of a whiteboard at each node.*

Proof. We prove the above statement by contradiction. Let us consider a dynamic cactus graph \mathcal{G} (refer to Fig. 3) of n vertices, in which at most k edges are dynamic at any round. The graph \mathcal{G} contains C_i cycles, where $i \in \{1, \dots, k\}$, in which except the last cycle C_k which of length $n + 2 - 3k$, every other cycle is of length 3. Now, suppose there exists an algorithm \mathcal{H} , which successfully locates the black hole position in \mathcal{G} with a set of $k + 1$ agents, $\{a_1, \dots, a_{k+1}\}$. Each agent is initially located at *home*, and 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 , where $i \in \{1, 2, \dots, k - 1\}$ and the remaining two agents enter the cycle C_k . Suppose, the black hole is located at any one among the three consecutive nodes $S = \{w_k, x_k, y_k\}$, of which the agents have no idea. Since the adversary can disappear and reappear at most k edges at any round, hence, it has the ability to 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 $i \in \{1, \dots, k - 1\}$). So, the remaining agents a_k and a_{k+1} have no other 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, 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 ($i \in \{1, \dots, k - 1\}$). Let us consider, without loss of generality a_k is the first to enter S , i.e., a node w_k at some round r or both agents enter a node in S at round r (a_k enters w_k and a_{k+1} enters y_k). At this point, regardless of the position of a_{k+1} , the adversary removes the edge (v_k, w_k) . Now, in any case, a_k cannot communicate with a_{k+1} in the presence of a whiteboard as well, as on one side there is a black hole and on the other side, there is a missing edge. So, if w_k is a safe node, then a_1 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 ($i \in \{1, \dots, k - 1\}$), respectively. Suppose if y_k is safe, then a_{k+1} can convey this information to at least one of the agents a_i by writing this information in the whiteboard of either u_i or w_i (where $1 \leq i \leq k - 1$). So, now the only possibility for a_{k+1} is to visit the next node x_k , if x_k is the black hole, then the remaining $k - 1$ agents cannot identify among x_k and w_k which is indeed the black hole node. The reason is, if w_k is the black hole, then a_k has already been eliminated, in this situation, the adversary can remove the edge (y_k, x_k) , which in turn restricts a_{k+1} to come out of C_k and convey this information to remaining $k - 1$ agents. Now, since none of the remaining agents can come out of their respective cycles, regardless of the whiteboard information, left by the agents a_k and a_{k+1} before entering the black hole node, none of the remaining agents cannot figure out the exact location among x_k and w_k which is the correct black hole position. This leads to a contradiction. \square

Corollary 2 (Lower bound for k dynamic edges). *To locate the black hole in a dynamic cactus graph \mathcal{G} with at most k dynamic edges at any round, any*

algorithm requires at least $k + 2$ agents to solve the black hole search problem in \mathcal{G} .

The next two theorems give lower bound and improved lower bound complexity with $k + 2$ and $2k + 3$ agents, respectively.

Theorem 6. *Given a dynamic cactus graph \mathcal{G} , with at most k dynamic edges at any round. In presence of whiteboard, any algorithm \mathcal{H} which solves the black hole search problem with $k + 2$ agents requires $\Omega((n + 2 - 3k)^{1.5})$ rounds and $\Omega((n + 2 - 3k)^{1.5} + 2k)$ moves when the agents have distinct Ids and they are co-located.*

Proof. We prove the above statement by contradiction, we suppose that there exists an algorithm \mathcal{H} which finds the black hole in $o((n + 2 - 3k)^{1.5})$ rounds. Now, consider the graph \mathcal{G} (refer to Fig. 3) of k -cycles, where $|C_i| = 3$ ($1 \leq i \leq k - 1$) and $|C_k| = n + 2 - 3k$, in which the set of agents $\mathcal{A} = \{a_1, a_2, \dots, a_{k+2}\}$ 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 ($1 \leq i \leq k - 1$), whereas a_k, a_{k+1} and a_{k+2} enter C_k . Now, by theorem 6 in [9], a set of 3 agents requires $\Omega((n + 2 - 3k)^{1.5})$ rounds to correctly locate the black hole inside C_k , this leads to a contradiction. Moreover, a constant number of agents move in $\Omega((n + 2 - 3k)^{1.5})$ rounds, whereas to enter this configuration starting from *home*, at least $2k$ moves are required. Hence, the team of $k + 2$ agents require $\Omega((n + 2 - 3k)^{1.5})$ rounds and $\Omega((n + 2 - 3k)^{1.5} + 2k)$ moves to find the black hole. \square

Theorem 7. *Any algorithm \mathcal{H} in the presence of whiteboard which solves the black hole search problem with $2k + 3$ agents in a dynamic cactus graph with at most k dynamic edges at any round requires $\Omega(n + 2 - 3k)$ rounds and $\Omega(n + 2 - k)$ moves when the agents have distinct Ids and they are co-located.*

Proof. We prove the above statement by contradiction, we suppose \mathcal{H} finds the black hole in $o(n + 2 - 3k)$ rounds. Now, suppose \mathcal{G} be the graph (refer to Fig. 3) with k -cycles, where $|C_i| = 3, \forall 1 \leq i \leq k - 1$ and $|C_k| = n + 2 - 3k$, and consider the agents to be initially co-located at *home*. Now, suppose the agents while executing \mathcal{H} enter a configuration, where a_i gets stuck in C_i , where $1 \leq i \leq k - 1$. In this situation, the remaining $k + 4$ agents try to explore C_k , so by theorem 3 we know that it takes four agents among the $k + 4$ agents to successfully locate the black hole in $\Omega(n + 2 - 3k)$ rounds. The bound on the number of moves comes from the fact that at least $2k$ additional moves are required to attain this configuration, whereas a constant number of agents moves in $\Omega(n + 2 - 3k)$ rounds. Hence, any algorithm requires $\Omega(n + 2 - 3k)$ rounds and $\Omega(n + 2 - k)$ moves. \square

4 Black Hole Search in Dynamic Cactus

In this section, we first present an algorithm to find the black hole in the presence of at most one dynamic edge, and then we present an algorithm to find the black

hole in the presence of at most k dynamic edges. The number of agents required to find the black hole is presented and further, the move and round complexities are analyzed for each algorithm.

4.1 Black Hole Search in Presence of Single Dynamic Edge

In this section, we present two black hole search algorithms, one for the agents and the other for the *LEADER*, in the presence of at most one dynamic edge in a cactus graph. Our algorithms require each node v to have some local storage space, called *whiteboard*. Our algorithms find the black hole search problem with the help of 3 agents a_1 , a_2 and a_3 , respectively. Among these three agents, we consider a_3 to be the *LEADER*. The task of the *LEADER* is different from the other two agents. The fundamental task of *LEADER* is either to instruct an agent to perform a certain movement or to conclude that a certain agent has entered the black hole. In contrast, the agent's task is to visit nodes that are not explored. More precisely, our algorithm for *LEADER* is SINGLEEDGEBH-SLEADER and for the agents a_1 and a_2 is SINGLEEDGEBH-AGENT. Next, we discuss the contents of the whiteboard.

Whiteboard: For each node $v \in \mathcal{G}$, a whiteboard is maintained with a list of information for each port of v . For each j , where $j \in \{0, \dots, \deg(v) - 1\}$, an ordered tuple $(f(j), \text{Last.LEADER})$ is stored on it, where the function f is defined as follows: $f : \{0, \dots, \deg(v) - 1\} \rightarrow \{\perp, 0, 1\}^*$,

$$f(j) = \begin{cases} \perp, & \text{if } j \text{ is yet to be explored by any agent} \\ 0 \circ A, & \text{if the set of agents in } A \text{ has visited } j \text{ but cannot} \\ & \text{fully explore the sub-graph originating from } j \\ 1, & \text{if the sub-graph corresponding to } j \text{ is fully explored} \\ & \text{and no agent is stuck} \end{cases}$$

The symbol ' \circ ' refers to the concatenation of two binary strings. We define A to be the set of agents which has visited that particular port. More precisely, if a_1 and a_2 both visits the port j , then we have $A = \{a_1, a_2\}$. We discuss the entries on the whiteboard with the help of the following example. Consider a port j at a node u , along which only a_1 has passed, but is unable to completely explore the sub-graph originating from j . In this case, the function $f(j)$ returns the binary string 001, where the first 0 represents that the sub-graph originating from j is not fully explored, and the next 01 represents the *Id* of a_1 , so we have $A = \{a_1\}$.

The entry *Last.LEADER* stores the bit 1 if j is the last visited port in v by the *LEADER*, otherwise, it stores 0.

Each agent (i.e., a_1 and a_2) performs a t -INCREASING-DFS [14], where the movement a_1 and a_2 can be divided in to two categories *explore* and *trace*:

- In *explore*, an agent performs either *cautious* walk or *pendulum* walk depending on the instruction of the *LEADER*. In this case, an agent visits a node for the first time, i.e., it only chooses a port j , such that $f(j) = \perp$.

- In *trace*, an agent performs *pendulum* walk, where it visits a node that has already been visited by the other agent. In this case, an agent a_1 (or a_2) chooses a port j , where $f(j) = 0 \circ A$ and $A = \{a_2\}$ (or $A = \{a_1\}$).

The task of the *LEADER* can be explained as follows:

- Instructs a_1 or a_2 to perform either *cautious* or *pendulum* walk.
- Maintains the variables $length_{a_i}$ and P_{a_i} ($a_i \in \{1, 2\}$), while a_i is performing *pendulum* walk. The *LEADER* increments $length_{a_i}$ by 1, whenever a_i traverses a new node and report this information to *LEADER*. Moreover, P_{a_i} is the sequence port traversed by a_i from the initial node from which it started its *pendulum* walk to the current explored node.
- It also maintains the variables L_{a_i} and PL_{a_i} , where L_{a_i} calculates the length of the path traversed by *LEADER* away from the initial position of a_i and PL_{a_i} stores the sequence of these ports.
- Lastly, terminates the algorithm whenever it knows the black hole position.

An agent a_1 or a_2 *fails to report* to *LEADER*, for one of the following reasons: it has either entered the black hole or it has encountered a missing edge along its forward movement.

The algorithm SINGLEEDGEBHSLEADER assigns only the *LEADER* to communicate with the remaining agents, in order to instruct them regarding their movements, whereas the agents a_1 and a_2 do not communicate among themselves. The agents a_i (where $i \in \{1, 2\}$) stores P_{a_i} while performing *pendulum* walk. For example, let a_1 and *LEADER* are co-located at h in Fig. 1. Now, suppose a_1 starts *pendulum* walk and reaches v_1 . Correspondingly, P_{a_2} gets updated to $P_{a_2} \cup 0$.

Next, we discuss in detail the description of the algorithm.

Algorithm Description: In this section, we give a high-level description of the algorithms SINGLEEDGEBHSAGENT and SINGLEEDGEBHSLEADER.

Description of SINGLEEDGEBHSAGENT(): This algorithm is executed by the agents a_1 and a_2 , in which they are either instructed to perform *cautious* or *pendulum* walk. The agents perform *t-INCREASING-DFS* [14] strategy for deciding the next port, further that port is indeed chosen by the agent for its movement based on the whiteboard information. Before commencing the algorithm, each node is labelled as $(\perp, 0)$. Initially, the agents start from *home*, where without loss of generality, a_1 is instructed to perform *cautious* and a_2 is instructed to perform *pendulum* walk. Let us consider a_1 is at a node u , then the decision taken by a_1 based on the contents of the whiteboard is as follows:

- If \exists at least one port with $f()$ value is \perp and i being the minimum among them, then choose that port and move to its adjacent port u' via i from u .
- If there is no such port i at u , with $f(i) = \perp$, then it backtracks accompanying the *LEADER* to a node where \exists such i with $f(i) = \perp$.

Now, suppose a_1 reaches u' through the i -th port and it is safe, then it returns back to u in the subsequent round, on condition that the edge (u, u') remains.

Otherwise, it stays at u' until the edge reappears. Now, while it returns back to u , if *LEADER* is found, then it accompanies *LEADER* to u' in the next round. Otherwise, it moves towards *LEADER* following *Last.LEADER* entries on whiteboard.

Next, let us consider a_2 is at a node v , then the decision taken by a_2 based on the whiteboard contents at v is as follows:

- If \exists at least one port with $f()$ value is \perp and i being the minimum among them, then choose that port and move to its adjacent port v' via i from v .
- If there is no such port i with $f(i) = \perp$, but \exists a port j with $f(j) = 0 \circ A$, where $A = \{a_1\}$ then it chooses that port and moves to its adjacent node v' . Otherwise, if $A = \{a_2\}$ or $A = \{a_1, a_2\}$, then it chooses a different available port, or backtracks to a node where there exists an available.
- If all ports have the value 1, then it backtracks to a port where each port value is not 1.

Suppose, a_2 travels to v' using the port i , then first it stores the port i , and after visiting v' it moves towards *LEADER*, based on the stored ports, if it is unable to find *LEADER*, then it follows *Last.LEADER* to meet the *LEADER*. Now, whenever it meets the *LEADER* it provides the sequence of ports P_{a_i} it has traversed from its initial position to the *LEADER*. Moreover, if at any moment it encounters a missing edge and no other agent is waiting for that missing edge, then it waits until the edge reappears. Now, irrespective of *cautious* or *pendulum* walk, whenever an agent a_1 (or a_2) moves along a port i (say), it updates $f(i) = f(i) \circ a_1$ (or $f(i) = f(i) \circ a_2$) in whiteboard with respect to i .

Description of SINGLEEDGEBHSLEADER(): This algorithm is executed by the *LEADER*. It initially instructs a_1 to perform *cautious* and a_2 to perform *pendulum* walk. Whenever an agent, suppose a_2 while performing *pendulum* walk, explores a new node and meets with *LEADER*, it increments $length_{a_2}$ by 1 and stores the sequence of port P_{a_i} from a_i . On the other hand, if the *LEADER* moves from its current position away from a_2 , it increments L_{a_2} by 1 at each such movement while updating the sequence of ports PL_{a_i} after each such movement. Suppose, if the *LEADER* moves away from a_i from its current node u to a node v along the port i . It does the following things: first, it updates *Last.LEADER* at u corresponding to i as 1, while the rest to 0. Second, it increments L_{a_i} by 1 and lastly, updates $PL_{a_i} = PL_{a_i} \cup \{i\}$. Further, whenever *LEADER* finds a missing edge along its path, it stops until the edge reappears. The instructions made by *LEADER* related to the movement of a_1 and a_2 are as follows:

- If a_2 without loss of generality (w.l.o.g), fails to report while performing *pendulum*, while a_1 is performing *cautious* walk, then *LEADER* instructs a_1 to perform *pendulum* walk.
- If *LEADER* is stuck at one end of the missing edge, then it instructs both a_1 and a_2 to perform *pendulum* walk if not already performing the same.
- If *LEADER* finds a missing edge reappear, while both a_1 and a_2 performing *pendulum* walk, then it instructs either a_1 or a_2 to perform *cautious* walk, based on the fact which among a_1 or a_2 is faster to report to *LEADER*.

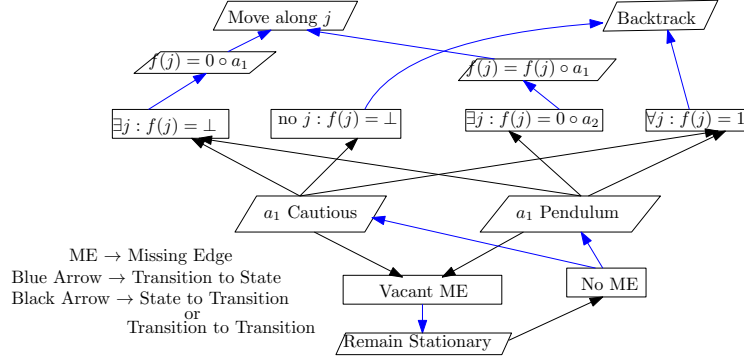


Fig. 4: An illustration of algorithm `SINGLEEDGEBHSAGENT()` in terms of the agent a_1

Next, we discuss the situations, when *LEADER* decides that either a_1 or a_2 has entered the black hole and terminates the algorithm.

- If a_1 w.l.o.g, while performing *cautious* walk fails to return, but the edge between *LEADER* and a_1 remains, then *LEADER* identifies a_1 to be in black hole.
- If a_1 w.l.o.g, while performing *cautious* walk fails to return, but the edge between *LEADER* and a_1 is missing, on the other hand a_2 which is performing *pendulum* walk also fails to return, then *LEADER* identifies a_2 to be in black hole.
- If both a_1 and a_2 is performing *pendulum* walk, and *LEADER* is stuck at one end of the missing edge. In this situation if both a_1 and a_2 fail to report, then *LEADER* moves towards the agent a_1 (or a_2) with the help of P_{a_1} and PL_{a_1} (or P_{a_2} and PL_{a_2}) based on the fact which among them is last to report to the *LEADER*, suppose that agent is a_1 , then we have the following cases:
 - If a_1 is found, then the *LEADER* understands a_2 in black hole.
 - If a_1 cannot be found and there is no missing edge, then *LEADER* identifies a_1 to be in a black hole.
 - If a_1 cannot be found, and there is a missing edge, then *LEADER* waits, and if then also a_2 fails to report then *LEADER* identifies a_2 to be in a black hole.

Figure 4, represents all possible states that an agent a_1 or a_2 attain, while executing `SINGLEEDGEBHSAGENT()`, whereas Figure 5, represents all possible states the *LEADER* attains while executing `SINGLEEDGEBHSLEADER()`. The pseudocode of `SINGLEEDGEBHSAGENT()` is explained in Algorithm 1, whereas the pseudocode of `SINGLEEDGEBHSLEADER()` is explained in Algorithm 2.

Correctness and Complexity: In this section, we prove the correctness of our algorithm, as well as give the upper bound results in terms of move and round complexity.

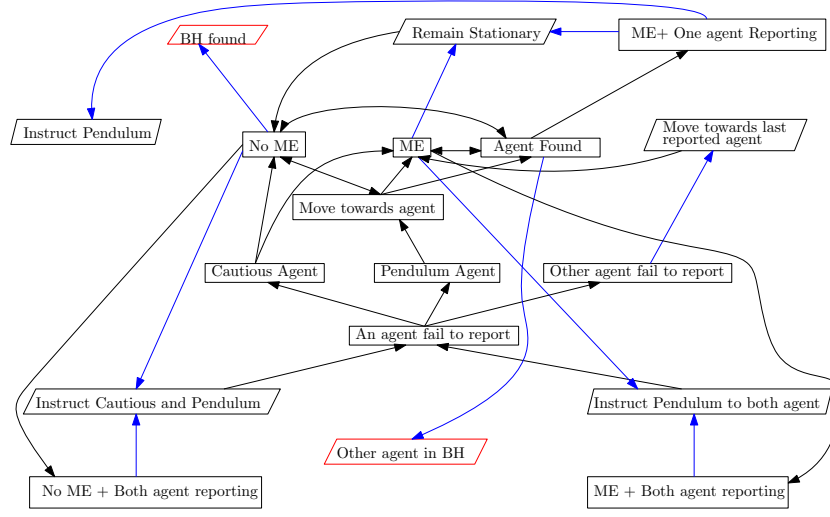


Fig. 5: An illustration of algorithm SINGLEEDGEBHSLEADER()

Lemma 3. *Given a dynamic cactus graph \mathcal{G} with at most one dynamic edge at any round r . Our Algorithms SINGLEEDGEBHSAGENT() and SINGLEEDGEBHSLEADER(), ensures that at most 2 agents are stuck due to the missing edge.*

Proof. In order to prove this claim, we first discuss the decisions that *LEADER* takes following Algorithm 2, whenever it either finds an agent a_1 (w.l.o.g) or a missing edge.

- *LEADER* finds a_1 at u stuck due to a missing edge (u, v) : In this case, if the edge does not reappear, then *LEADER* instructs a_1 to perform *pendulum* walk, whereas it remains stationary at u until the edge (u, v) reappears.
- *LEADER* while tracing for a_1 finds a missing edge (u, v) : If a_1 has not already entered the black hole, then it is at v while the *LEADER* is at u , because a_1 while reporting back to *LEADER* gets stuck at u , whereas *LEADER* after finding a_1 is not reporting back, moves towards a_1 and gets stuck at u .

So, in any case, *LEADER* does not allow more than one agent to occupy one end of the missing edge. Next, we discuss the decisions taken by the agent a_1 (w.l.o.g) following the Algorithm 1, whenever it either finds another agent or encounters a missing edge.

- If a_1 finds u to be vacant and (u, v) is a missing edge: In this case, a_1 remains stationary at u , until either the edge reappears or until *LEADER* appears at u .
- If the node u of the missing edge (u, v) is occupied by a_2 : In this case, if a_1 does not find any available ports at u then it backtracks, otherwise, it moves along the available port at u .

Algorithm 1: SINGLEEDGEBHSAGENT(a_i)

```

1 Set  $P_{a_i} = \emptyset$ .
2 while  $a_i$  performs cautious walk do
3   if each port at  $u$  has  $f(j) \neq \perp$  then
4     Backtrack with LEADER to a node  $w$ , where  $\exists j$ , such that
        $f(j) = \perp$ ,  $j \in \{0, 1, \dots, \deg(w) - 1\}$ .
5   else
6     Choose the minimum  $j$  at  $u$  with  $f(j) = \perp$ , and update
        $P_{a_i} = P_{a_i} \cup \{j\}$  and  $f(j) = 0 \circ a_1$ .
7     Travel to the adjacent node  $v$  via  $j$ .
8     if  $v$  is safe and  $e = (u, v)$  remains then
9       Return back to  $u$  and take LEADER to  $v$ .
10    else if  $v$  is safe and  $e = (u, v)$  is missing then
11      Remain at  $v$  until  $e$  reappears.
12 while  $a_i$  performs pendulum walk do
13   if each port at  $u$  has  $f(j) = 1$  then
14     Backtrack to a node  $w$  where  $\exists j$  such that
        $f(j) \neq 1$ ,  $j \in \{0, 1, \dots, \deg(w) - 1\}$ .
15     if a missing edge is encountered while backtracking then
16       Remain stationary until the edge reappears.
17   else
18     Choose the minimum port  $j$  such that  $f(j) \neq 1$  or  $0 \circ A$ , where  $a_i \in A$ .
19     Store  $P_{a_i} = P_{a_i} \cup \{j\}$  and move to adjacent node  $v$  via  $j$ .
20     Update  $f(j) = f(j) \circ a_i$  and return to LEADER following  $P_{a_i}$  and
       Last.LEADER.
21     if a vacant missing edge is encountered then
22       Remain stationary until the edge reappears.
23     else
24       Choose a different available port or backtrack.
25     Return back to  $v$  using  $P_{a_i}$ .

```

So, irrespective of the situation from the above steps, we conclude that at most 2 agents can occupy a node on each side of the missing edge. All the above scenarios will hold as well if instead of a_1 , the agent is a_2 . This proves our statement. \square

Lemma 4. *The LEADER following the algorithm SINGLEEDGEBHSLEADER(), ensures that among the two agents stuck due to a dynamic edge, eventually, one must be the LEADER.*

Proof. We prove the above claim by contradiction. We claim that, whenever two agents a_1 and a_2 are at either side of a missing edge, they invariably need to hold these positions until the missing edge reappears. Consider the scenario,

Algorithm 2: SINGLEEDGEBHSLEADER($a_1, length_{a_2}, L_{a_2}, P_{a_2}, PL_{a_2}$)

```

1 Instruct  $a_1$  to perform cautious and  $a_2$  to perform pendulum walk.
2 Set  $length_{a_1} = L_{a_1} = 0$  and  $PL_{a_1} = \emptyset$ .
3 while the black hole is not found do
4   if  $a_1$  does not return after a round then
5     if no missing edge then
6       └ Conclude  $a_1$  has entered the black hole and terminate.
7     else
8       Remain stationary until the edge reappears.
9       if  $a_2$  fail to return within  $2(L_{a_2} + length_{a_2} + 1)$  round then
10        if the edge is still missing then
11          └ Conclude  $a_2$  has entered the black hole and terminate.
12        else
13          Instruct  $a_1$  to perform pendulum walk and move towards  $a_2$ 
            following  $P_{a_2}$  and  $PL_{a_2}$  while each time incrementing  $L_{a_1}$ 
            by 1 and updating  $PL_{a_1}$  by storing each port traversed.
14          Set  $Last.LEADER = 1$  for each port traversed and
             $Last.LEADER = 0$  for rest of the ports at each node.
15          PENDULUM( $a_2, length_{a_1}, length_{a_2}, L_{a_1}, L_{a_2}, P_{a_1}, P_{a_2}, PL_{a_1}, PL_{a_2}$ ).
16        else
17          └ Update  $length_{a_2} = length_{a_2} + 1$  and store  $P_{a_2}$  from  $a_2$ .
18      else
19        Update  $L_{a_2} = L_{a_2} + 1$  and  $PL_{a_2} = PL_{a_2} \cup \{j\}$ , where  $j$  is the port
            taken by  $LEADER$ .
20        if  $a_2$  fail to return within  $2(L_{a_2} + length_{a_2} + 1)$  round then
21          Instruct  $a_1$  to perform pendulum walk and move towards  $a_2$ 
            following  $P_{a_2}$  and  $PL_{a_2}$  while each time incrementing  $L_{a_1}$  by 1
            and updating  $PL_{a_1}$  by storing each port traversed.
22          Set  $Last.LEADER = 1$  for each port traversed and
             $Last.LEADER = 0$  for rest of the ports at each node.
23          PENDULUM( $a_2, length_{a_1}, length_{a_2}, L_{a_1}, L_{a_2}, P_{a_1}, P_{a_2}, PL_{a_1}, PL_{a_2}$ ).
24        else
25          └ Update  $length_{a_2} = length_{a_2} + 1$  and store  $P_{a_2}$  from  $a_2$ .

```

where an edge $(u, v) \in \mathcal{G}$ is missing for a finite but sufficiently large number of rounds, and a_1 and a_2 are stuck at u and v , respectively. In the first case, we consider the scenario where both the agents are performing *pendulum*. Now, this scenario has arisen because $LEADER$ is occupying one end of an earlier missing edge, i.e., either at u' or v' (refer to Algorithm 3). Since, at a round at most one edge can be missing, hence this earlier missing edge has reappeared, making way for the new missing edge (u, v) , which has stuck both a_1 and a_2 at u and v , respectively. Now, as both of them fail to report, the $LEADER$ moves towards

Algorithm 3: PENDULUM($a_2, length_{a_1}, length_{a_2}, L_{a_1}, L_{a_2}, P_{a_1}, P_{a_2}, PL_{a_1}, PL_{a_2}$)

```

1 while black hole is not found do
2   if  $a_2$  is found then
3     Set  $length_{a_2} = L_{a_2} = 0$  and  $PL_{a_2} = \emptyset$ .
4     if a forward edge is missing then
5       Remain stationary and instruct  $a_2$  to perform pendulum walk.
6       if  $a_1$  does not return after  $2(L_{a_1} + length_{a_1} + 1)$  round then
7         Wait for  $2(L_{a_2} + length_{a_2} + 1)$  round.
8         if  $a_2$  does not return to LEADER then
9           Traverse towards  $a_2$  following  $P_{a_2}$  and  $PL_{a_2}$ , while
              updating  $Last.LEADER$  and incrementing  $L_{a_1}$  and
              updating  $PL_{a_1}$  for each port traversed.
10          if  $a_2$  is found then
11            Conclude the  $a_1$  has entered black hole and terminate.
12          else
13            If a forward edge is missing, then wait for
               $2(length_{a_1} + 1 + L_{a_1})$  rounds.
14            If the edge is still missing and  $a_1$  does not return,
              conclude  $a_1$  is in black hole and terminate.
15            Otherwise, conclude  $a_2$  has entered black hole and
              terminate.
16          else
17            Increment  $length_{a_2} = length_{a_2} + 1$  and update  $P_{a_2}$  from  $a_2$ .
18            PENDULUM( $a_2, length_{a_1}, length_{a_2}, L_{a_1}, L_{a_2}, P_{a_1}, P_{a_2}, PL_{a_1}, PL_{a_2}$ ).
19        else
20          Increment  $length_{a_1} = length_{a_1} + 1$  and update  $P_{a_1}$  from  $a_1$ .
21      else
22        SINGLEEDGEBHSLEADER( $a_2, length_{a_1}, L_{a_1}, P_{a_1}, PL_{a_1}$ ).
23  else if it encounters a missing edge then
24    MISSING( $a_2, length_{a_1}, length_{a_2}, L_{a_1}, L_{a_2}, PL_{a_1}, P_{a_1}, PL_{a_2}, P_{a_2}$ ).
25  else
26    Wait for one round, and conclude  $a_2$  has entered black hole and
      terminate.

```

the agent which has last reported (say a_1 without loss of generality) and finds a_1 . Whenever it finds a_1 , it instructs a_1 to continue performing *pendulum* walk while it remains stationary at u . This leads to a contradiction. In the second case, suppose a_1 at u finds a missing edge (u, v) , while it is performing *cautious* walk with *LEADER*, whereas a_2 at v also finds a missing edge (u, v) while it is performing *pendulum* walk. In this scenario, a_1 and *LEADER* are stuck at u whereas a_2 is stuck at v . In this case, *LEADER* following the Algorithm 2 instructs a_1 to perform *pendulum* walk, whereas it remains stationary at u . In

Algorithm 4: MISSING($a_2, length_{a_1}, length_{a_2}, L_{a_1}, L_{a_2}, PL_{a_1}, P_{a_1}, PL_{a_2}, P_{a_2}$)

```

1 while  $a_2$  is not found do
2   Remain stationary at one end of the missing edge.
3   if  $a_1$  fails to report within  $2(L_{a_1} + length_{a_1} + 1)$  rounds then
4     Move towards  $a_1$  following  $P_{a_1}$  and  $PL_{a_1}$  while updating  $L_{a_2}$ ,  $PL_{a_2}$ 
       and  $Last.LEADER$ .
5     if  $a_1$  is not found then
6       if there is no missing edge then
7         Conclude  $a_1$  has entered black hole and terminate.
8       else
9         Remain stationary.
10        if  $a_2$  does not report after  $2(L_{a_2} + length_{a_2} + 1)$  rounds then
11          Conclude  $a_2$  has entered black hole and terminate.
12        else
13          Conclude  $a_1$  has entered black hole and terminate.
14      else
15        Increment  $length_{a_1} = length_{a_1} + 1$  and update  $P_{a_1}$  from  $a_1$ .
16 if the forward edge is missing then
17   PENDULUM( $a_2, length_{a_1}, length_{a_2}, L_{a_1}, L_{a_2}, P_{a_1}, P_{a_2}, PL_{a_1}, PL_{a_2}$ ).
18 else
19   SINGLEEDGEBHSLEADER( $a_2, length_{a_1}, L_{a_1}, P_{a_1}, PL_{a_1}$ ).

```

this case, also, our claim leads to a contradiction. So we conclude that eventually among the two agents stuck due to a missing edge, one must be the *LEADER*. \square

Lemma 5. *An agent a_1 or a_2 following the SINGLEEDGEBHSAGENT(), does not enter an infinite cycle.*

Proof. Suppose C_1 be a cycle in \mathcal{G} , where u be the node along which an agent a_1 (w.l.o.g) enters the cycle and moves to the adjacent node v using the port j . While it moves to v via j , it updates $f(j) = f(j) \circ a_1$. Now, after exploring C_1 , whenever the agent returns back to u and again tries to take the port j , it finds that a_1 has already visited the port j based on the whiteboard information, so irrespective of the agent performing *cautious* or *pendulum* walk, it does not take the port j again (refer to steps 3-4 and 18 of Algorithm 1). This phenomenon of an agent, concludes that it never enters an infinite cycle while performing SINGLEEDGEBHSAGENT(). \square

Lemma 6. *Algorithm SINGLEEDGBHSAGENT(), ensure that in the worst case, every node in \mathcal{G} is explored by either a_1 or a_2 until the black hole node is detected.*

Proof. Observe, each agent is a t -state finite automata, where $t \geq \alpha n \log \Delta$, the agents a_1 and a_2 performs t -INCREASING-DFS while *explore* or *trace*. In

addition, the whiteboard information helps a_1 and a_2 find the set of ports it hasn't visited yet at each node. Now, as stated in theorem 6 and corollary 7 in [14], an agent with $O(n \log \Delta)$ -bits of memory can explore any static graph with maximum degree Δ and diameter at most n . Moreover, lemma 5 ensures that a_1 or a_2 never enters an infinite cycle. Further, lemma 4 ensures that either *LEADER* and one among a_1 or a_2 gets stuck by a missing edge, whereas the other agent can explore unobstructively until it either enters the black hole or detects it, as the underlying graph can have at most one dynamic edge. So, combining all these phenomenon, we can guarantee that a_1 or a_2 , can explore the underlying graph until the black hole is detected. \square

Observation 8 *Algorithm SINGLEEDGEBHSLEADER, ensures that LEADER does not enter the black hole.*

Lemma 7. *In the worst case, a_1 and a_2 executing algorithm SINGLEEDGBHSAGENT(), enters black hole in $O(n^2)$ rounds.*

Proof. Recall, *LEADER* either instructs both a_1 and a_2 to perform *pendulum* walk, or it instructs one among them to perform *cautious* walk while the other *pendulum* walk.

- a_1 is instructed cautious walk, whereas a_2 pendulum walk. Since, at most one edge is missing, then if a_2 is not blocked at all by any missing edge, then in at most $2n$ rounds, it explores a new node.
- Both a_1 and a_2 are instructed pendulum walk. This scenario arises when the *LEADER* is stationary at one end of the missing edge. Now, in the worst case the missing edge for which *LEADER* is stationary reappears, whereas both a_1 and a_2 while exploring a cycle along disjoint paths, gets blocked by a new missing edge (refer to Fig. 1, where suppose a_1 is at v_7 and a_2 at v_8). In this case, *LEADER* finding no missing edge adjacent to itself, reaches either of them in at most n rounds. Now, if a_1 is reached, then *LEADER* instructs a_1 to continue *pendulum* walk, while *LEADER* remains stationary. In this scenario as well, a_1 explores a new node each in at most $2n$ rounds.

Hence, in any case, in at most $2n$ rounds, an agent explores at least one new node. This implies in $O(n)$ rounds, at least a new node is explored. This concludes the statement that either a_1 or a_2 enters the black hole in $O(n^2)$ rounds. \square

Lemma 8. *Let us consider, the agents a_1 or a_2 enter the black hole at round r while executing SINGLEEDGEBHSAGENT(), then the LEADER following SINGLEEDGEBHSLEADER() detects the black hole node in $r + O(n^2)$ rounds.*

Proof. Suppose after r rounds either a_1 or a_2 enters the black hole. Now in each of the following scenarios, we show that the *LEADER* takes additional $O(n^2)$ rounds after round r to detect the black hole:

- *Scenario-1:* Suppose w.l.o.g, a_1 while performing *cautious* walk with the *LEADER* enters the black hole at round r . Hence, a_1 does not return back

to *LEADER*, whereas the edge e between the *LEADER* and black hole is present. So, after waiting for one more round, *LEADER* terminates the algorithm by concluding the black hole position. On the contrary, suppose the edge e remains missing from the round r onwards. Since, there is at most one missing edge at any round, and moreover the other end of the missing edge contains the black hole. So, a_2 can unobstructively explore a new node in at most $2n$ rounds while executing *pendulum* walk, until it reaches the black hole. Hence, in $O(n^2)$ rounds, the agent a_2 enters black hole. Therefore, a_2 will also fail to report, and *LEADER* will conclude the black hole position in $O(n)$ additional rounds from the round when a_2 enters the black hole. In conclusion, the *LEADER* takes $O(n^2)$ rounds to conclude the black hole position, after a_1 has entered the black hole.

- *Scenario-2*: Consider w.l.o.g, a_2 while performing *pendulum* walk reaches the black hole at round r , while a_1 is performing *cautious* walk with *LEADER*. So after a_2 fails to report, *LEADER* performs the following task.
 - If a_1 is with *LEADER* when a_2 fails to report whereas the forward edge is missing, then *LEADER* instructs a_1 to perform *pendulum* walk whenever this edge reappears. Subsequently, *LEADER* moves towards a_2 .
 - If a_1 is with *LEADER* when a_2 fails to report and there is no missing edge in the forward direction, then *LEADER* instructs a_1 to perform *pendulum* walk, whereas *LEADER* moves towards a_2 .

So, in at most n rounds, *LEADER* reaches the last marked node of a_2 following P_{a_2} and PL_{a_2} . Now, if a_2 is not found and there is no forward missing edge, then *LEADER* concludes that a_2 has reached the black hole. This conclusion is executed in additional $O(n)$ rounds after a_2 has entered the black hole, i.e., after round r .

Now on the contrary, if a_2 is not found by *LEADER*, and there exists a missing edge (u, v) along the path towards a_2 in the forward direction, then *LEADER* remains stationary at u (say), but a_1 invariably performs *pendulum* walk until it either reaches the black hole or gets stuck at v . The *LEADER* in $O(n^2)$ rounds, understands the failure of a_1 's return as well, leaves u and moves towards a_1 . The possibilities are: either a_1 is found, or the *LEADER* encounters another missing edge, or a_2 is not found. In each case, *LEADER* concludes the black hole position in $O(n^2)$ rounds after a_2 has entered the black hole.

- *Scenario-3*: Both a_1 and a_2 are performing *pendulum* walk, in which a_1 (say) enters the black hole. Now, again a_2 unobstructively explores the graph in $O(n^2)$ rounds while the *LEADER* is stationary at one end of the missing edge, until a_2 either enters black hole or gets stuck by a missing edge. Now, the *LEADER* invariably travels towards a_2 , while it faces the following instances: either a_2 is found, or a_2 is not found, or encounters a missing edge. In any case, the *LEADER* concludes the black hole position in $O(n^2)$ rounds after a_1 has entered the black hole.

□

Lemma 9. *The LEADER following the algorithm SINGLEEDGEBHSLEADER() correctly locates the black hole position.*

Proof. We discuss all possible scenarios that can occur while executing algorithm SINGLEEDGEBHSLEADER().

a_1 enters black hole at u during *cautious* walk: In this case, according to Algorithm 2, if the edge $e = (u, v)$ between *LEADER* and a_1 is missing, then *LEADER* waits at v until e reappears. Meanwhile, a_2 is performing *pendulum* walk, and since there is at most one dynamic edge at any round, hence a_2 will eventually reach u , i.e., enter the black hole from an alternate path. Now in this case, the *LEADER* after waiting $2(length_{a_2} + 1 + L_{a_2})$ rounds, concludes that a_2 has entered black hole. The conclusion is indeed correct, as one end of the edge which is missing edge, is occupied by *LEADER* and the other end contains black hole. Since, there is at most one missing edge in \mathcal{G} at any round. So, a_2 faces no obstruction while exploring the graph, until it enters the black hole. Hence, within $2(length_{a_2} + 1 + L_{a_2})$ rounds a_2 otherwise will have reported back to *LEADER*.

a_1 enters black hole during *pendulum* walk: a_1 which is initially performing *cautious*, performs *pendulum* walk for two reason.

- a_2 fails to report. This situation arises when a_1 is initially performing *cautious* walk with *LEADER*, and a_2 which is performing *pendulum* walk fails to report. In this scenario, the *LEADER* according to Algorithm 2, instructs a_1 to perform *pendulum* walk, while it moves towards a_2 following P_{a_2} and PL_{a_2} . Now we have two possibilities: first, a_2 is found and second a_2 is not found. If a_2 is found, then it is instructed to either perform *cautious* or *pendulum* walk, based on the fact that there is a forward missing edge or not. Now, in this situation, since a_1 has entered black hole and fails to report, then *LEADER* moves towards a_1 while instructing a_2 to perform *pendulum* walk. If the *LEADER* does not find a missing edge corresponding to the last visited port of a_1 , then it concludes that a_1 has entered black hole. Otherwise, if *LEADER* gets stuck at a node v due to a missing edge $e = (u, v)$, then it remains stationary, and while a_2 will either get stuck at u or eventually enter black hole. So, ultimately a_2 also fails to report and *LEADER* not knowing the reason behind a_2 's failure to report, moves towards a_2 leaving the node v . If a_2 is found, i.e., it got stuck at u then it correctly concludes a_1 has entered the black hole, whereas if *LEADER* encounters another missing edge while moving towards a_2 , then after waiting an additional $2(L_{a_1} + length_{a_2} + 1)$ round, it correctly concludes a_1 has entered black hole, as otherwise a_1 will have reported back to *LEADER*. Otherwise, if a_2 is not found, and there is no missing edge, then also *LEADER* correctly concludes a_2 has entered black hole, as otherwise a_2 will have reported back to *LEADER*.
- a_1 performs *pendulum* walk with a_2 : This situation arises because, a_2 initially stopped reporting due to a missing edge, while a_1 is performing *cautious* walk. Now, as *LEADER* moves towards a_2 it instructs a_1 to

change its movement to *pendulum* walk. Moreover, it finds a_2 and the missing edge persists, in this situation, a_2 is further instructed to continue *pendulum* walk whereas *LEADER* remains stationary at one end of the missing edge. If both a_1 and a_2 fail to return, then *LEADER* identifies at least one among them has entered black hole. It is because, the *LEADER* holds one end of the missing edge, and at most one agent may be stuck at the other end, whereas the remaining agent must return if it has not entered the black hole. So, *LEADER* moves towards the last reported agent a_1 (say w.l.o.g). If a_1 is not found and there is no missing edge in the forward direction, then *LEADER* correctly identifies the black hole position. Otherwise, if *LEADER* finds a missing edge along its path towards a_1 and further waits for $2(\text{length}_{a_2} + 1 + L_{a_2})$ rounds, within which if a_2 also fails to return, then *LEADER* concludes a_2 has entered black hole. Since, *LEADER* has moved towards a_1 and encountered a new missing edge, this implies that the earlier missing edge has reappeared. Hence, a_2 has no other obstruction along its path towards *LEADER*, if it is originally stuck due to the earlier missing edge.

The explanation are similar, when a_2 enters black hole while performing either *cautious* or *pendulum* walk. So, we have shown that in each case the *LEADER* correctly determines the black hole location. \square

Theorem 9. *The agent following algorithms SINGLEEDGEBHSAGENT() and SINGLEEDGEBHSLEADER(), correctly locates the black hole in a dynamic cactus graph \mathcal{G} with at most one dynamic edge at any round with $O(n^2)$ moves and in $O(n^2)$ rounds.*

Proof. Lemmas 7 and 8 states that our Algorithms 1 and 2, correctly find the black hole in $O(n^2)$ rounds, and since at each round each agent can move at most once. Hence, there can be at most 3 moves in each round of our algorithm. This implies, that the agent following Algorithms 1 and 2, solves the black hole search problem in $O(n^2)$ moves. \square

4.2 Black Hole Search in Presence of Multiple Dynamic Edges

In this section, we present an algorithm MULTIEDGEBHS() for the agents to locate the black hole position, where the underlying graph is a dynamic cactus graph \mathcal{G} but unlike the earlier section, where at most one edge can be missing at any round, in this section, we discuss the case in which there can be at most k dynamic edges, such that the underlying graph remains connected. As discussed earlier, each node $v \in \mathcal{G}$ is equipped with a whiteboard of $O(\deg(u)(\log \deg(u) + k \log k))$ bits of memory. Moreover, there are a team of $2k + 3$ agents, $\mathcal{A} = \{a_1, \dots, a_{2k+3}\}$ which executes the algorithm MULTIEDGEBHS(), starting from a safe node also termed as *home*. Next, we define the contents of information that can be present on a whiteboard.

Whiteboard: For each node $v \in \mathcal{G}$, a whiteboard is maintained with a list of information for each port of v . For each port j , where $j \in \{0, \dots, \deg(v) - 1\}$,

an ordered tuple $(g_1(j), g_2(j))$ is stored on the whiteboard. The function g_1 is defined to be exactly the same as the function f in section 4.1.

On the other hand, the function g_2 is defined as follows, $g_2 : \{0, \dots, \deg(v) - 1\} \rightarrow \{\perp, 0, 1\}$,

$$g_2(j) = \begin{cases} \perp, & \text{if an agent is yet to visit the port } j \\ 0, & \text{if no agent has returned to the node } v \text{ along } j \\ 1, & \text{otherwise} \end{cases}$$

Each agent performs a t -INCREASING-DFS [14], where the movement of each agent can be divided into two types *explore* and *trace*:

- In *explore*, an agent performs either *cautious* walk or *pebble* walk depending on the situation.
- In *trace*, an agent walks along the safe ports of a node v , i.e., all such port $j \in v$ with $g_2(j) = 1$.

Our algorithm MULTIEDGEBHS() requires no *LEADER*, unlike our previous algorithms in section 4.1. In this case each a_i (where $i \in \{a_1, a_2, \dots, a_{2k+3}\}$) executes their operations, based on the whiteboard information they gather at each node.

Next, we give a detailed description of the algorithm MULTIEDGEBHS().

Outline of Algorithm: The team of agents $\mathcal{A} = \{a_1, \dots, a_{2k+3}\}$ are initially located at a safe node, termed as *home*. Initially, the whiteboard entry corresponding to each port at each node in \mathcal{G} is (\perp, \perp) . The lowest *Id* agent present at *home*, i.e., a_1 in this case decides to perform *cautious* walk. At the first round, it chooses the 0-th port and if the edge corresponding to 0-th port exists, then it moves along it to an adjacent node v while updating $(g_1(0), g_2(0))$ at *home* from (\perp, \perp) to $(0 \circ a_1, 0)$. Now, if v is safe, and the edge (home, v) exists then it returns to *home* at the second round, while updating $g_2(0)$ at *home* to 1, i.e., marking the edge (home, v) as safe. Further, if at the third round the edge (home, v) exists, then a_1 accompanies $\mathcal{A} \setminus \{a_1\}$ to v while updating $g_1(0) = g_1(0) \circ A'$, where $A' = \{a_2, \dots, a_{2k+3}\}$. Otherwise, if the edge has gone missing, at the second round, then a_1 and a_2 remain at v and *home*, respectively until the edge reappears, whereas the remaining agent continues to perform their respective movement.

Now, consider a scenario where the edge (home, v) goes missing at the third round when all the \mathcal{A} agents are at *home*. In this case, a_1 remains at *home* until the edge reappears, whereas the remaining agents continue to perform *cautious* walk along the other available ports. Whenever the edge reappears suppose at the r -th round, then it starts *pebble* walk. The movement of a_1 performing *pebble* walk is as follows: at the $r + 1$ -th round a_1 moves to v , further the agent moves as follows:

- If there exists a port i with $g_2(i) = 0$, and the edge remains, then at the $r + 2$ -th round a_1 stays at v . If no agent returns along i -th port, a_1 concludes that the node w.r.to the port i is the black hole node. Otherwise, if an agent a_j

returns (for some $j > 0$), then both a_1 and a_j start *cautious* walk. Moreover, if the edge does not exist, and there is no other agent at v , then a_1 waits until the edge reappears. Otherwise, if there is already an agent waiting, then a_1 decides to move from v to some adjacent node, based on the whiteboard entry.

- If there exists a port i with $g_2(i) = \perp$, then at $r + 2$ round a_1 chooses that port and moves to the adjacent node while updating $(g_1(i), g_2(i))$ to $(0 \circ a_1, 0)$.
- If each port at v is having its $g_2()$ value 1, then at $r+2$ -th round a_1 backtracks to *home*, if $(home, v)$ exists, otherwise stays at v until the edge reappears.

In general, whenever multiple agents meet at a node, they start *cautious* movement. Moreover, when a single agent waiting for a missing edge reappears, then it starts *pebble* walk. In addition, whenever an agent finds a port i at some node in \mathcal{G} with $g_2(i) = 0$ and the edge exists, then after waiting for a round, it concludes the adjacent node w.r.to i is the black hole node.

The pseudo code of MULTIEDGEBHS() is explained in Algorithm 2.

Algorithm 5: MULTIEDGEBHS(u, a_i)

```

1 if more than one agent is available at  $u$  then
2   | MULTIPLEAGENT( $u, a_i$ ).
3 else
4   | Perform SINGLEAGENT( $u, a_i$ ).

```

Correctness and Complexity: In this section we analyze the correctness and complexity of our algorithm 5.

Lemma 10. *Given a dynamic cactus graph \mathcal{G} with at most k dynamic edges at any round r . Our algorithm, MULTIEDGEBHS() ensures that at most 2 agents are stuck due to a missing edge at any round.*

Proof. An agent executing algorithm 5 may encounter a missing edge in two possible ways:

- First, when the agent is performing *cautious* walk.
- Second, when the agent is performing *pebble* walk.

In either case, the agent remains stationary when it encounters a vacant missing edge (refer to Algorithms 6 and 7). Moreover, if any agent at a node u finds another agent waiting due to a missing edge w.r.to a port j (where $j \in \{0, 1, \dots, \deg(u) - 1\}$), then it either chooses a different available port or if there are no such available ports then the agent backtracks. Hence, either end of a missing edge can be occupied by at most 2 agents. \square

Algorithm 6: MULTIPLEAGENT(u, a_i)

```

1  if  $a_i$  is the minimum  $Id$  agent at  $u$  then
2    if a port  $j$  with  $(\perp, \perp)$  exists then
3      Traverse to  $v$  via  $j$  and update  $g_1(j) = 0 \circ a_i$  and  $g_2(j) = 0$  at  $u$ .
4      if  $v$  is safe and  $(u, v)$  exists then
5        Return to  $u$ , and update  $g_2(j) = 1$  at  $u$ .
6        if  $(u, v)$  exists then
7          Move all available agents to  $v$  via and update  $g_1(j) = 0 \circ A'$ ,
            where  $A'$  is the set of available agents at  $u$ .
8          Perform MULTIEDGEBHS( $v, a_i$ ).
9        else
10         Stay at  $u$  until  $(u, v)$  reappears, if no agent is waiting for  $(u, v)$ .
11         Perform MULTIEDGEBHS( $u, a_m$ ), where  $a_m$  is the next
            available minimum  $Id$  agent at  $u$  after  $a_i$ .
12      else if  $v$  is safe but  $(u, v)$  goes missing then
13        Remain at  $v$ , until the edge reappears.
14    else if a port  $j$  of  $u$  with  $(0 \circ A, 0)$  exists then
15      if the edge is missing then
16        Stay at  $u$  until  $(u, v)$  reappears, if no agent is waiting for  $(u, v)$ .
17        Perform MULTIEDGEBHS( $u, a_m$ ), where  $a_m$  is the next available
            minimum  $Id$  agent at  $u$  after  $a_i$ .
18      else
19        Conclude the adjacent node to be the black hole and terminate.
20    else if a port  $j$  of  $u$  with  $(0 \circ A, 1)$  exists then
21      if the edge is missing then
22        Stay at  $u$  until  $(u, v)$  reappears, if no agent is waiting for  $(u, v)$ .
23        Perform MULTIEDGEBHS( $u, a_m$ ), where  $a_m$  is the next available
            minimum  $Id$  agent at  $u$  after  $a_i$ .
24      else
25        if  $A$  contains  $Id$  of at least one available agent at  $u$  then
26          Choose a different available port or backtrack.
27        else
28          Move all available agents to  $v$  via  $j$ -th port and update
             $g_1(j) = g_1(j) \circ A'$ , where  $A'$  are the set of all available agents
            then perform MULTIEDGEBHS( $v, a_i$ ).
29    else
30      Backtrack with all available agents to a node  $w$  with a port  $j$ ,
             $g_1(j) \neq 1$  perform MULTIEDGEBHS( $w, a_i$ ).
31  else
32    Follow instruction of the minimum  $Id$  agent.

```

Algorithm 7: SINGLEAGENT(u, a_i)

```

1 if a port  $j$  at  $u$  with  $(\perp, \perp)$  exists then
2   If that edge exists, then visit that node  $v$  while updating  $g_1(j) = 0 \circ a_i$  and
    $g_2(j) = 0$ .
3   Else, if that edge is missing, stay at  $u$  until  $(u, v)$  reappears, if no agent is
   waiting for  $(u, v)$ .
4   if  $v$  is safe then
5     If the edge  $(u, v)$  exists, then return to  $u$  and update  $g_2(j) = 1$ .
6     Else, if the edge  $(u, v)$  is missing, then wait until it reappears.
7     if  $(u, v)$  exists then
8       Visit  $v$  and perform MULTIEDGE BHS( $v, a_i$ ).
9     else
10      Stay at  $u$  until  $(u, v)$  reappears, if no agent is waiting for  $(u, v)$ .
      Else, choose a different port and continue MULTIEDGE BHS( $u, a_i$ ).
11 else if a port  $j$  at  $u$  with  $(0 \circ A, 0)$  exists then
12   if  $A$  does not contain  $a_i$  then
13     If the edge exists, and no agent returns after one round, then
       terminate by concluding the node w.r.to the port  $j$  is the black hole.
14     Else, if the edge is missing, then stay at  $u$  until  $(u, v)$  reappears, if no
       agent is waiting for  $(u, v)$ . Else, choose a different port and perform
       MULTIEDGE BHS( $u, a_i$ ).
15   else
16     Choose a different port and perform MULTIEDGE BHS( $u, a_i$ ).
17 else if a port  $j$  at  $u$  with  $(0 \circ A, 1)$  exists then
18   if  $A$  does not contain  $a_i$  then
19     Move along  $j$  to  $v$  while updating  $g_2(j) = g_2(j) \circ a_i$  at  $u$  and perform
       MULTIEDGE BHS( $v, a_i$ ).
20   else
21     Choose a different port and perform MULTIEDGE BHS( $u, a_i$ ).
22 else
23   Backtrack to a node  $w$  with a port  $j$ ,  $g_1(j) \neq 1$  and perform
       MULTIEDGE( $w, a_i$ ).

```

Lemma 11. *In the worst case at most 2 agents are consumed by the black hole, while the agents are following the algorithm MULTIEDGE BHS().*

Proof. By lemma 2, it has been shown that if v is the black hole node and the node u is the *home*, then any path from *home* to v must either pass along v_0 or v_1 , where v_0 and v_1 belong to the same cycle as v in \mathcal{G} . This implies that an agent passing through these unexplored edges (v_0, v) and (v_1, v) cannot mark the nodes v_0 and v_1 to be safe, as they enter the black hole. So, any subsequent agent visiting either v_0 or v_1 finds that $g_2()$ value corresponding to the edges (v_0, v) and (v_1, v) are 0. As any cycle can have at most one missing edge at

any round such that the underlying graph remains connected, the adversary can either disappear (v_0, v) or (v_1, v) at any round. So, any set of agents trying to visit these unsafe nodes while executing `MULTIEDGEBHS()`, can find these nodes v_0 and v_1 unsafe, and can successfully locate the black hole node without any further agents entering the black hole. Hence, this guarantees that at most two agents can be consumed by the black hole. \square

Lemma 12. *Our algorithm `MULTIEDGEBHS()` ensures that no agent enters an infinite cycle.*

Proof. Consider C to be a cycle in \mathcal{G} and u is the node in C along which an agent a_i while executing Algorithm 5 traverses to an adjacent node $v \in C$ via the port i . Now, a_i may be traversing with other agents performing *cautious* walk (in that case $g_1(i) = g_1(i) \circ A'$ refer to step 28 of Algorithm 6) or traversing alone performing *pebble* walk (in this case $g_1(i) = g_1(i) \circ a_i$ refer to step 19 of Algorithm 7). In either case, a_i after exploring C whenever reaches u and again decides to travel along i , then with the help of $g_1(i)$ at the whiteboard, identifies that it has already traversed this port before. In this situation, it decides not to take this port, and either chooses another available port or backtrack (refer to step 25-26 of Algorithm 6 when a_i is performing *cautious* walk, otherwise step 12-16 of Algorithm 7). \square

Lemma 13. *`MULTIEDGEBHS()` ensures that any agent which is not stuck due to a missing edge can explore the remaining graph until it either enters the black hole or detects it.*

Proof. As stated, each agent is a t -state finite automata, where $t \geq \alpha n \log \Delta$. An agent while executing either during *cautious* or *pebble* walk, performs the t -INCREASING-DFS algorithm, which in turn ensures that any static graph of diameter at most n and maximum degree Δ can be explored with $O(n \log \Delta)$ bits of memory (refer to Theorem 6 and to Corollary 7 in [14]). Now, in this situation of dynamic graph, the whiteboard in addition helps the agent in determining the ports it has yet to visit from a node ($g_1(i) = \perp$ or $0 \circ A$, where A does not contain a_i , refers that i -th port not visited by a_i). Moreover, the whiteboard also restricts the agent from entering in an infinite cycle loop (refer to Lemma 12). So, an agent which is not stuck due to a missing edge, can explore each node of the underlying graph until it either enters the black hole or detects the black hole. \square

Theorem 10. *Given a dynamic cactus graph \mathcal{G} with at most k dynamic edges at any round. Our algorithm `MULTIEDGEBHS()` ensures that it requires at most $2k + 3$ agents, to successfully locates the black hole position.*

Proof. By lemma 10, we have shown that at most 2 agents are stuck due to a dynamic edge. Now at any round, there can be at most k dynamic edges, which in turn implies that at most $2k$ agents can be stuck due to these dynamic edges. Moreover by lemma 11, it is shown that at most 2 agents get eliminated by the black hole. Moreover, lemma 13, ensures that any agent which is not stuck

can explore the graph until it either detects the black hole or gets eliminated by it. Hence, the remaining agent among $2k + 3$ agents can traverse along the graph unobstructively, and whenever it finds a vertex adjacent to a black hole, along which a port is marked unsafe (i.e. $g_2()$ value with respect to a port is 0) in whiteboard, it waits for one round and if no agent returns then it correctly concludes that the node w.r.to the unsafe port is the black hole position. \square

Theorem 11. *The team of agents $\mathcal{A} = \{a_1, a_2, \dots, a_{2k+3}\}$ following MULTIEDGEBHS(), locates the black hole in a dynamic cactus graph \mathcal{G} with at most k dynamic edges at any round with $O(kn)$ rounds and in $O(k^2n)$ moves.*

Proof. Consider a graph with k -cycles, C_1, C_2, \dots, C_k , each cycle can have at most one dynamic edge at a round, such that the graph remains connected. Now, we analyze the complexity with the help of the following cases based on the size of C_i 's, $1 \leq i \leq k$.

- *Case-1:* Suppose $|C_i| \leq \frac{n}{k}$, $\forall 1 \leq i \leq k$, consider a cycle C_1 with the nodes $u, v, v', w, w' \in C_1$. Let us assume, the set of $2k + 3$ agents enter C_1 from u and moves along a clockwise direction and encounters a missing edge (v, v') , which separates a_1 (say) from \mathcal{A} . In this scenario, the algorithm MULTIEDGEBHS() instructs a_2 to remain stationary at v , whereas the remaining agents leave a_1 and a_2 and start moving in a counter-clockwise direction, either to *trace* or to *explore*. Now, in the meantime suppose the adversary revives the edge (v, v') , which reunites a_1 and a_2 and they continue to move forward, whereas disappears another edge (w, w') in the counter-clockwise direction, which separates further two agents a_3 and a_4 (say) in a similar manner. Now, in order to separate these 4 agents from \mathcal{A} , the adversary requires $O(\frac{n}{k})$ rounds. Continuing in this manner, in order to separate $2k + 3$ agents by reappearing and disappearing edges in C_1 , it takes $O(k\frac{n}{k}) = O(n)$ rounds. Hence, in order to explore each of k such cycles, in the worst case $O(kn)$ rounds are required. Moreover, in each round at most $2k + 3$ agents move, hence in the worst case $O(k^2n)$ moves are required.
- *Case-2:* Suppose $|C_i| = 3$, $\forall 2 \leq i \leq k$, and $|C_1| = n + 2 - 3k$. As discussed in the earlier case it requires $O(k(n + 2 - 3k))$ round to separate $2k + 3$ agent in C_1 , whereas since the other cycles are of size 3, it takes an additional $O(k)$ rounds to perform the same task in the remaining $k - 1$ cycles. So, in general, it requires $O(k(n - 2k))$ rounds to explore \mathcal{G} in this case. Moreover, at most $2k + 3$ agents move in each round, hence in the worst case $O(k^2(n - 2k))$ moves.

So, we analyze the two extreme possibilities, from the above cases, and conclude that in the worst case the algorithm MULTIEDGEBHS() requires $O(kn)$ rounds and $O(k^2n)$ moves to locate the black hole. \square

5 Conclusion

In this paper, we studied the black hole search problem in a dynamic cactus for two types of dynamicity. We propose algorithms and lower bound and upper

bound complexities in terms of number of agents, rounds and moves in each case of dynamicity. First, we studied at most one dynamic edge case, where we showed with 2 agents it is impossible to find the black hole, and designed a black hole search 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, also we propose a black hole search algorithm with $2k + 3$ agents. Further, we propose that it is impossible to find the black hole with $k + 1$ agents in this scenario. A future work is to design an algorithm which has a tight bound in terms of 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.

References

1. Agarwalla, A., Augustine, J., Moses Jr, W.K., Madhav, S.K., Sridhar, A.K.: Deterministic dispersion of mobile robots in dynamic rings. In: Proceedings of the 19th International Conference on Distributed Computing and Networking. pp. 1–4 (2018)
2. Balamohan, B., Flocchini, P., Miri, A., Santoro, N.: Time optimal algorithms for black hole search in rings. *Discrete Mathematics, Algorithms and Applications* **3**(04), 457–471 (2011)
3. Chalopin, J., Das, S., Labourel, A., Markou, E.: 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. pp. 432–446. Springer (2011)
4. Czyzowicz, J., Kowalski, D., Markou, E., Pelc, A.: Complexity of searching for a black hole. *Fundamenta Informaticae* **71**(2-3), 229–242 (2006)
5. Czyzowicz, J., Kowalski, D., Markou, E., Pelc, A.: Searching for a black hole in synchronous tree networks. *Comb. Probab. Comput.* **16**(4), 595–619 (2007)
6. Das, S., Di Luna, G.A., Mazzei, D., Prencipe, G.: Compacting oblivious agents on dynamic rings. *PeerJ Computer Science* **7** (2021)
7. Di Luna, G., Dobrev, S., Flocchini, P., Santoro, N.: Distributed exploration of dynamic rings. *Distributed Computing* **33**, 41–67 (2020)
8. Di Luna, G.A., Flocchini, P., Pagli, L., Prencipe, G., Santoro, N., Viglietta, G.: Gathering in dynamic rings. *theoretical computer science* **811**, 79–98 (2020)
9. Di Luna, G.A., Flocchini, P., Prencipe, G., Santoro, N.: Black hole search in dynamic rings. In: 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS). pp. 987–997. IEEE (2021)
10. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. *Distributed Computing* **19** (2006)
11. Dobrev, S., Santoro, N., Shi, W.: Scattered black hole search in an oriented ring using tokens. In: 2007 IEEE International Parallel and Distributed Processing Symposium. pp. 1–8. IEEE (2007)
12. Dobrev, S., Santoro, N., Shi, W.: 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)
13. Flocchini, P., Kellett, M., Mason, P.C., Santoro, N.: Searching for black holes in subways. *Theory of Computing Systems* **50**, 158–184 (2012)

14. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. *Theoretical Computer Science* **345**(2-3), 331–344 (2005)
15. Gotoh, T., Flocchini, P., Masuzawa, T., Santoro, N.: Exploration of dynamic networks: tight bounds on the number of agents. *Journal of Computer and System Sciences* **122**, 1–18 (2021)
16. Gotoh, T., Sudo, Y., Ooshita, F., Kakugawa, H., Masuzawa, T.: Exploration of dynamic tori by multiple agents. *Theoretical Computer Science* **850**, 202–220 (2021)
17. Gotoh, T., Sudo, Y., Ooshita, F., Masuzawa, T.: Dynamic ring exploration with (h, s) view. *Algorithms* **13**(6), 141 (2020)
18. Ilcinkas, D., Wade, A.M.: Exploration of dynamic cactuses with sub-logarithmic overhead. *Theory of Computing Systems* **65**, 257–273 (2021)
19. Kshemkalyani, A.D., Molla, A.R., Sharma, G.: Efficient dispersion of mobile robots on dynamic graphs. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). pp. 732–742. IEEE (2020)
20. Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: *Proceedings of the forty-second ACM symposium on Theory of computing*. pp. 513–522 (2010)
21. Mandal, S., Molla, A.R., Moses Jr, W.K.: Live exploration with mobile robots in a dynamic ring, revisited. In: *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*. pp. 92–107. Springer (2020)
22. Markou, E., Paquette, M.: Black hole search and exploration in unoriented tori with synchronous scattered finite automata. In: *Principles of Distributed Systems: 16th International Conference, OPODIS 2012, Rome, Italy, December 18–20, 2012. Proceedings 16*. pp. 239–253. Springer (2012)