Cycling Through a Dangerous Network: A Simple Efficient Strategy for Black Hole Search*

Stefan Dobrev
SITE, University of Ottawa
800 King Edward Avenue
Ottawa, ON K1N 6N5, Canada
sdobrev@site.uottawa.ca

Paola Flocchini
SITE, University of Ottawa
800 King Edward Avenue
Ottawa, ON K1N 6N5, Canada
flocchin@site.uottawa.ca

Nicola Santoro
SCS, Carleton University
1125 Colonel By Drive
Ottawa, ON, K1S 5B6, Canada
santoro@scs.carleton.ca

Abstract

In this paper we consider a dangerous process located at a node of a network (called Black Hole *) and a team of mobile agents deployed to locate that node. The nature of the danger is such that when an agent enters the dangerous node, it is trapped there leaving no trace of its destruction. The goal is to deploy as few agents as possible and to locate the black hole in as few moves as possible.*

We present a simple algorithm that works on any topology (a-priori known by the agents). Our algorithm, based on the pre-computation of an open vertex cover by cycles of the network, uses the optimal number of agents (two); its cost (number of moves) depends on the choice of the cover and it is optimal for several classes of networks.

Keywords: Mobile Agents, Malicious Host, Undetectable Failure, Black Hole Search
Technical Area: Algorithms and Theory

1 Introduction

1.1 The Problem

In networked environments that supports mobile agents, security is an important issue of difficult solution (e.g., see [1, 8, 11, 14, 16]). We consider a particular security problem known as *host attacks*; that is, the presence in a site of a stationary process that harm incoming agents (e.g., see [7, 9, 10, 13, 15, 18]). A first step in solving such a problem should be to determine and report the location of the harmful process, so to avoid the node in the future, or to neutralize the process restoring a safe environment. The task to identify the harmful host is clearly dangerous for the searching agents and, depending on the nature of the harm, it might be impossible to perform.

In this paper, we consider a highly harmful process that disposes of visiting agents upon their arrival, leaving no observable trace of such a destruction. Due to its nature, the site where such a process is located is called a *black hole* (e.g., [2, 3, 5, 4, 6, 12]).

The goal is to deploy the team of agents to locate the black hole (black hole location problem BLACK HOLE SEARCH). The agents start from the same node (the *homebase*), have limited memory, and move *asynchronously* from node to neighbouring node; each agent has a labeled map of the graph with the indication of the homebase. Each network site has a limited amount of storage, called *whiteboard* that the agents use to communicate to each other. The black hole location problem is solved if at least one agent survives, and all surviving agents know the location of the black hole.

The performance of a solution is measured in terms of number of agents deployed (the *size* of the team) and number of moves performed by the agents (the *cost*).

It is known that, with a map of the network, a team of *two agents* suffice, and the number of moves is in the worst case $O(n \log n)$ [4], where n is the number of nodes. Such a cost is optimal in the sense that there exist topologies

*Work partially supported by NSERC, Tecsis Co., and by Dr. Flocchini's University Research Chair

where $\Omega(n \log n)$ moves are necessary (e.g., the ring [5]), but might be unnecessarily high for other topologies. For instance, there are networks in which two agents can locate the black hole with just $\Theta(n)$ moves [3].

This leads to the problem of designing a solution strategy enabling two agents to locate a black hole with at most $O(n \log n)$ moves in any network, and in $O(n)$ moves in many networks.

A step in this direction is the recent result of [6] showing how two agents with a map can locate the black hole performing $O(n + d \log d)$ moves (where d is the diameter of the graph), yielding an $O(n)$ bound for most interconnection networks.

The main drawback of this result is that the approach used is extremely complex. Furthermore, because its cost depends on the value of d, it fails to achieve an $O(n)$ cost in networks with $\Omega(n)$ diameter, requiring instead $O(n \log n)$ moves. However, there exist networks with $\Omega(n)$ diameter in which the black hole might be found using $O(n)$ moves (a simple example is a $2 \times \frac{n}{2}$ mesh).

1.2 Our Result

In this paper we consider an asynchronous environment and we present a simple algorithm that works on an arbitrary network, provided that the agents have a map of the network. Our algorithm employs the optimal number of agents (two), and the number of moves depends on the topology of the network.

Let $\mathcal{C}$ be a set of simple cycles such that each vertex of G is covered by a cycle from $\mathcal{C}$ and the connectivity graph of these cycles (where each cycle is represented by a vertex, and two vertices are connected if the corresponding cycles share an edge) is connected. Such a set of cycle will be formally defined later and is called *Open Vertex Cover* by cycles. The algorithm is based on the pre-computation of such an open vertex cover by cycles of a graph. The idea is to explore the graph G by exploring the cycles of $\mathcal{C}$. As we will see, if an agent becomes stuck on an edge e (because either the transmission delay on e is very high, or it leads to the BH), the other agent will be able to bypass it, using the cycle containing e, and continue the exploration.
Our main result is:

Theorem 1.1. (Main Result)
Let G be a 2-connected graph and let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be an open vertex cover by cycles of G. Then, it is possible for 2 agents to locate the black hole in G using $O(\sum_{i=1}^{k} |C_i| \log |C_i|)$ moves.

On a wide class of networks, such a cover can be precomputed so to obtain $\Theta(n)$ cost for locating the black hole.

This class includes all Abelian Cayley graphs (e.g., hypercubes, (multi-dimensional) tori, etc,) of degree three and more, as well as many non-Abelian cube graphs (e.g., CCC, butterfly, wrapped-butterfly networks, etc.). For some of these networks, this is the only algorithm achieving such a bound.

For other graphs, it produces topology-sensitive bounds that are, in most cases the best available one; e.g., in $2 \times \frac{n}{2}$ mesh.

When compared to existing approaches, our new approach has the additional advantage, to be quite robust with respect to network changes: a local change in the network changes only the cycles involved, and a new cover could be easily constructed in such a case.

1.3 Related Work

The BLACK HOLE SEARCH problem has been investigated in different contexts. In asynchronous settings, it has been studied when the network is an anonymous ring, characterizing the limits and presenting optimal solutions [5]; among other things, it has been shown that the black hole can be located with 2 agents with $\Theta(n \log n)$ moves. In [4] the problem has been considered when the network is arbitrary; optimal solutions have been given under different assumptions on the level of topological knowledge available to the agents. In particular, it has been shown that, for any known arbitrary network, there exists an algorithm that employs 2 agents with $\Theta(n \log n)$ moves, such a cost is optimal in the sense that there exist topologies where $\Omega(n \log n)$ moves are necessary (e.g., the ring). In [3], a different algorithm has been designed that achieves an optimal number of moves ($\Theta(n)$) for many interconnection networks. Finally, the recent result of [6] achieves a bound that depends on the topology; in fact the agents perform $O(n + d \log d)$ moves (where d is the diameter of the graph) yielding an optimal bound for most interconnection networks.

The BLACK HOLE SEARCH problem has been studied also in synchronous settings, where the time for an agent to traverse a link is assumed to be unitary; in this case, tight bounds have been established for some classes of trees [2]. In the case of general networks, approximation algorithms have been described in [2] and subsequently improved [12].

2 Model and Preliminaries

2.1 Model

Let $G = (V, E)$ be a simple 2-(vertex)connected graph; let $n = |V|$ be the size of G, and let $E(x)$ denote the links incident on $x \in V$, $d(x) = |E(x)|$ denote the degree of x, and Δ denote the maximum degree in G. If $(x, y) \in E$ then x and y are said to be neighbours.

Two distinct mobile agents a and b are operating in G. The agents can move from node to neighbouring node in G,

have computing capabilities and limited computational storage*, and obey the same set of behavioral rules (the "protocol"). The agents are *asynchronous* in the sense that every action they perform (computing, moving, etc) takes a finite but otherwise unpredictable amount of time. Both agents start at the same node h, called *home base*. The agents know the topology they are operating in, i.e. each agent has in its local memory a labeled map of the graph, with the *home base* marked on the map.

As in most mobile agents platforms, each node has available a limited amount of storage, called *whiteboard*. Agents communicate by reading from and writing on the whiteboards; access to a whiteboard is gained fairly in mutual exclusion.

A *black hole* (shortly BH) is a node where a stationary process that destroys any agent arriving at that node resides; no observable trace of such a destruction is evident to the other agents. The location of the black hole is unknown to the agents, however the agents know that there is one in the network. The BLACK HOLE SEARCH (shortly BHS) problem is to find the location of the black hole. More precisely, BHS is solved if at least one agent survives, and all surviving agents know the location of the BH (by having its location marked on their maps).

The complexity measure we are interested in is the *cost* of the solution, that is the total number of moves performed by the agents in the worst case.

2.2 Basic Tools

At any time during the search for the black hole, the ports (corresponding to the incident links) of a node can be classified as *unexplored* – no agent has been sent/received via this port, *explored* – an agent has been received via this port, or *dangerous* – an agent has been sent through this port, but no agent has been received from it. Clearly, an explored port does not lead to a black hole; on the other hand, both unexplored and dangerous ports might lead to it.

To minimize the number of agents lost, we employ a technique called *Cautious walk* (from [5]). The main idea is to avoid sending an agent over a dangerous link, while still achieving progress. This is accomplished using the following two rules: (1) No agent enters a *dangerous* link and (2) whenever an agent a leaves a node u through an unexplored port p (transforming it into dangerous), upon its arrival to node v, and before proceeding somewhere else, a returns to u (transforming that port into explored).

Similarly to the classification adopted for the ports, we classify nodes as follows: at the beginning, all nodes except the *home base* are *unexplored*; the first time a node is visited by an agent, it becomes *explored*. Note that, by definition, the BH never becomes explored. Explored nodes and edges are considered *safe*.

The following lemma summarizes some basic limitations on the black hole search problem:

Observation 2.1. *[4, 5]*
1. It is not possible to verify whether there is a BH *in the system or not (due to asynchrony, a very slow node cannot be distinguished from the* BH*).*
2. Single agent is not sufficient to locate the BH *(it may enter the* BH *in its first move).*
3. It is impossible to determine the location of the black hole if the size of G is not known (two slow links might lead to the BH*, or to a whole new part of the graph).*
4. If G has a cut vertex different from the home base*, then it is impossible to determine the location of the* BH *(cannot distinguish between the cut vertex being very slow, and the* BH *being located in the cut vertex).*

As a consequence, we assume that the network is 2-connected, that there are at least 2 agents and that the existence of the black hole is common knowledge to the agents. Recall that we also assume that the agents have full map of the network.

2.3 Covers and Cycles

We now introduce some definitions and properties that will be needed later.

Definition 2.2. Cycle Graph
Let $G = (V, E)$ be a 2-connected graph and let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be a set of simple cycles in G. We will call the Cycle Graph of $\mathcal{C}$ in G *the graph $C(G; \mathcal{C}) = (\tilde{V}, \tilde{E})$ defined as follows:*

- $\tilde{V} = \{\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_k\}$ *and*
- $(\tilde{v}_i, \tilde{v}_j) \in \tilde{E}$ *if and only if C_i and C_j have a common* edge.

In other words, the cycle graph of a set of cycles in a given graph captures the connectivity (and thus the reachability) of the cycles in the original graph: Two cycles are considered connected in the cycle graph if they share an edge, while sharing a node does not make them connected (the motivation comes from the fact that a BH in the shared node can disconnect the cycles, but with a shared edge there will always be a way to reach the neighbouring cycle). Cycle graph is in fact well known concept (introduced in [17]), mostly studied in planar graphs where it is closely related to the dual graph.

The following definition describes a particular set of cycles of a graph (*Open Vertex Cover* by Cycles) that will be heavily used in our algorithm:

*the agents need to store the map of the graph, other then that $O(1)$ counters of $O(\log n)$ bits suffi ce

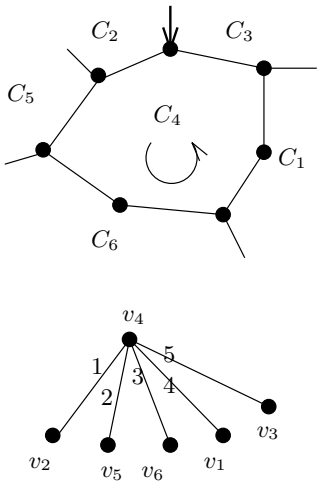

Figure 1. The ordering of neighbouring vertices in the cycle graph of G according to the cycle positions in G

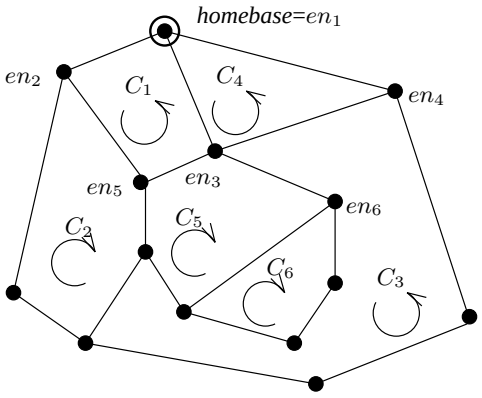

Figure 2. Example of *Cycle-DFT Sequence* for graph G and $\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5, C_6\}$. The cycle directions are shown, as well as the resulting entry nodes for each cycle. The resulting *Cycle-DFT Sequence*: $L = \{1, 2, 5, 3, 6, 3, 4\}$.

Definition 2.3. Open Vertex Cover
Let $G = (V, E)$ be a 2-connected graph and let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be a set of simple cycles in G. We say that $\mathcal{C}$ is an Open Vertex Cover by Cycles *of G (*OVC *for short) if and only if:*

1. $\forall v \in V : v \in \bigcup_{i=1}^{k} C_i$ (each vertex of G is covered by some cycle from $\mathcal{C}$)
2. *The cycle graph $C(G;\mathcal{C})$ is connected.*

Interestingly, an *OVC* can be constructed in every 2-connected graph:

Lemma 2.4. *Every 2-connected graph has an* OVC.

Proof. It follows from the fact that every 2-connected graph has an *open* ear decomposition $\mathcal{E} = \{E_1, E_2, \ldots, E_l\}$ [19]. We can obtain $\mathcal{C}$ from $\mathcal{E}$ by completing each ear into a cycle. The connectivity of $C(G;\mathcal{C})$ follows from the fact that the ear decomposition is open. □

The basic idea of our algorithm is to explore G by exploring the cycles of its *OVC*, guided by depth-first-traversal (DFT) order of $C(G;\mathcal{C})$. For efficiency reasons, not any DFT of $C(G;\mathcal{C})$ will do. In particular, we want the neighbours of a vertex $v_i \in C(G;\mathcal{C})$ to be considered in the same order their corresponding cycles appear when traversing the cycle C_i (see Figure 1). Note that this ordering depends both on the direction of the traversal of C_i and on the starting node (we will call it *entry node*) en_i.

A DFT for a given graph can be unambiguously determined by specifying (1) the starting node and (2) ordering of neighbours for each node. This leads to the following definition:

Definition 2.5. *We say that a sequence of integers $L = \{u_1, u_2, \ldots, u_s\}$ is a* Cycle-DFT Sequence *for G and* OVC *$\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ if the following conditions are satisfied:*

- *C_{u_1} contains the* home base
- *L corresponds to a DFT of $C(G;\mathcal{C})$ with the starting node v_{u_1} and the following ordering of neighbours:*
  - *For each cycle $C_i \in \mathcal{C}$ choose arbitrary, but fixed direction to be considered* clockwise.
  - *The neighbours of v_i are ordered in the order their corresponding cycles are encountered in a clockwise traversal of C_i, starting from the entry node en_i of C_i defined as follows:*
    - *en_{u_1} is the* home base
    - *Let l be the first occurrence of u_l in L (corresponding to visiting v_{u_l} for the first time in the DFT of $C(G;\mathcal{C})$), then en_{u_l} is the first node of $C_{u_{l-1}} \cap C_{u_l}$ encountered in the clockwise traversal of $C_{u_{l-1}}$ starting from $en_{u_{l-1}}$.*

The sequence L can be constructed for any 2-connected graph G and its *OVC* C simply by performing the DFT according to the definition. See Figure 2 for an example of a *Cycle-DFT Sequence* for a given graph.

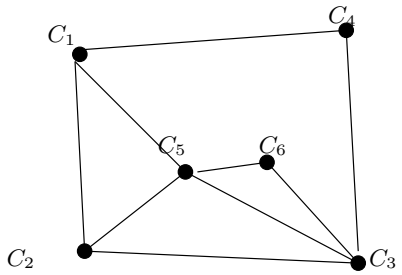

Figure 3. The cycle graph corresponding to the graph from Figure 2.

3 Algorithm EXPLORE

3.1 High Level Description

Let $C(G;\mathcal{C})$ be the cycle graph corresponding to an *OVC* $\mathcal{C}$ of graph G and let $L = \{u_1, u_2, \ldots, u_s\}$ be a *Cycle-DFT Sequence* for G and $\mathcal{C}$. (Precomputed by the agents since G is known.)

The goal of Algorithm EXPLORE is to collaboratively explore G until only one node, the black hole, is left unexplored. To do so, the agents explore G by using the sequence L: Exploring a node $u_i \in L$ means exploring the corresponding cycle C_{u_i}.

The basic step of Algorithm EXPLORE consists of exploring a cycle $C_i \in \mathcal{C}$ (described in Algorithm EXPLORE-CYCLE). Since C_i may contain the BH, exploring it means exploring all of its nodes, or all but one. Moreover, C_i has to be explored in a way ensuring that at least one agent survives. Two basic techniques are used to ensure that: Cautious walk and avoiding the *dangerous node (DN)*. The dangerous node for an agent a is a node where agent b has gone, but from which, to a's knowledge, b has not returned yet. During the algorithm, we will make sure that an agent never enters the node that it considers dangerous. To do so, each agent carries a variable DN, initially empty, which contains the *id* of the current dangerous node. Note that using full topological knowledge, the DN can indeed be avoided, even if it is on the other side of an unexplored link.

A cycle can be explored in two modes: *Alone* and *Together*. In the *Alone* mode single agent (w.l.o.g. assume a) explores C_i using cautious walk and avoiding the DN if necessary until either the whole C_i (possibly except the DN) is explored or a message from b is found that it came to help. In the latter case, the exploration of C_i proceeds in the *Together* mode using the divide-the-unexplored part technique from [5] until a single node remains unsafe (and considered the new DN). Once a cycle C_i has been explored, its parts might need to be traversed several more times (if i has several occurrences in L), we will call that the *Transit* mode.

At the beginning, both agents start exploring C_{u_1} from the *home base* in the *Together* mode. Eventually, one of then (e.g. a) will finish exploring C_{u_1} and will proceed to explore C_{u_2} in the *Alone* mode, avoiding the last unexplored node of C_{u_1} (where b was heading to) which is now the DN for a. Agent a will continue its solo exploration (possibly proceeding to $C_{u_{i+1}}$ after finishing C_{u_i}) until eventually b catches up at some cycle C_j, at which moment the DN is reset to empty and they start cooperatively exploring C_j in mode *Together*. The process is repeated until all cycles have been explored, at which moment the DN contains the BH. While going from one cycle to another, an agent always makes sure that it moves on safe links. Notice that, since u_i and u_{i+1} are neighbours in $C(G;\mathcal{C})$, C_{u_i} and $C_{u_{i+1}}$ share an edge. Therefore $C_{u_{i+1}}$ can always be reached from C_{u_i}, even while avoiding the DN.

The high level description of Algorithm EXPLORE is given below.

Algorithm 1 Algorithm EXPLORE

1: Set $i \leftarrow 1$, set DN to empty.
2: **while** $i \neq s$ **do**
3: Execute Algorithm EXPLORE-CYCLE for C_{u_i}.
4: **repeat**
5: Go to a node $w \in C_{u_i} \cap C_{u_{i+1}}$ using the edges of C_{u_i} and avoiding the DN and set $i \leftarrow i+1$.
6: **until** C_{u_i} is unexplored or $i = s$
7: **end while**
8: *// The last cycle has been explored.*
9: The black hole is in DN. **Terminate.**

3.2 Algorithm EXPLORE-CYCLE in Detail

The *Alone* mode of Algorithm EXPLORE-CYCLE uses Cautious walk to explore the current cycle C_i while avoiding the DN (see Figure 4) and terminates either when the full cycle has been explored, or when only DN remains unexplored, or when a message (Workload; U_a, U_b) left by the other agent is found.

In the *Transit* mode (corresponding to line 5. of Algorithm EXPLORE) the agent simply proceeds clockwise or counterclockwise (if the DN is in the clockwise-way) until it arrives to its destination. Note that cautious walk is not necessary, as C_i has already been explored.

The *Together* mode (lines 7 . . . 19, with line 7 including also the last/following part of the *Alone* mode) starts with one agent (say a) leaving message (Workload: U_a, U_b), and proceeding to explore its part using cautious walk. The crucial property is that $U_a \cup U_b$ contains all unexplored nodes of C_i, $U_a \cap U_b = \emptyset$, $||U_a| - |U_b|| \leq 1$ and both U_a and U_b are contiguous segments of C_i. The idea is that since $U_a \cap U_b = \emptyset$, the BH can be in at most one of the parts and

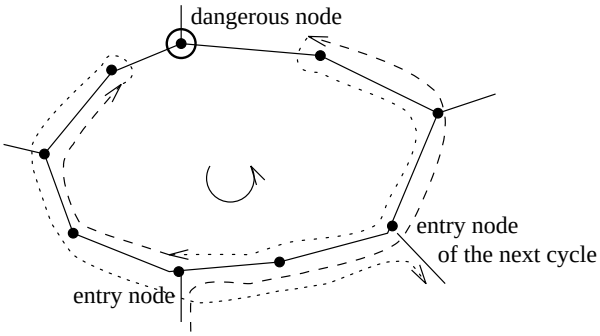

Figure 4. The way alone agent explores a cycle. The dashed path is explored using Cautious walk, the dotted path is a simple traversal of already safe edges.

therefore at least one of the agents (say a) will explore its part. a will then come to the last safe node explored by the other agent and evaluate the currently unexplored part U of C_i:

- If U is empty, C_i has been explored and a proceeds by following the safe links explored by b until catching up with b (b might have explored alone several cycles).
- If U contains a single node x, then x may be the BH and it does not make sense to wait for b. Instead, a remembers x as DN and proceeds to explore the remaining cycles alone.
- If $|U| \geq 2$, a splits U into two disjoint connected (almost) equal-sized partitions U_a and U_b (with U_b containing the node b is currently headed to), leaves a message (Workload: U_a, U_b) for b and proceeds to explore the new U_a. When (if) b returns, it will notice the workload message and update its workload.

The basic idea here is that every time the workload is recomputed, the unexplored area is at least halved, therefore $\log |C_i|$ iterations are sufficient to explore C_i. Each iteration costs $O(|C_i|)$ moves, for the resulting complexity of $O(|C_i| \log |C_i|)$, which is optimal for a ring [5].

3.3 Correctness and Complexity

Because of cautious walk, avoiding the DN, and $U_a \cap U_b = \emptyset$, there will never be the case of both agents heading for the same unexplored node while executing Algorithm EXPLORE-CYCLE. This immediately yields the liveness lemma:

Lemma 3.1. *At most one agent will enter the black hole.*

We now prove the progress lemma:

Algorithm 2 Algorithm EXPLORE-CYCLE for a cycle C_i

1: *// Executed by the first agent (say a) that starts exploring $\mathcal{C}_i$.*
2: Agent a explores C_i using cautious walk and avoiding DN until either the whole C_i has been explored (except the DN), or a message *(Workload: U_a, U_b)* has been found.
3: **if** whole C_i has been explored **then**
4: **return** *// Proceed to the next cycle.*
5: **end if**
6: *// A message* (Workload: U_a, U_b) *from the other agent has been found. Clear the DN as b has returned from there.*
7: Set $DN \leftarrow \epsilon$.
8: Explore U_a using cautious walk.
9: **repeat**
10: *// The second agent joining exploration of C_i starts here.*
11: Follow the safe links explored by the other agent until the last safe node v is reached or you learn that the whole C_i has been explored.
12: **if** the whole C_i has been explored **then**
13: **return** *// Proceed to the next cycle.*
14: **end if**
15: Let U be the smallest contiguous segment of C_i containing all nodes of C_i that are still unexplored.
16: **if** U contains single node x **then**
17: *// Proceed to the next cycle.*
18: Set $DN \leftarrow x$. **return**
19: **else**
20: Decompose U into two connected subsegments U_a and U_b such that (1) $U_a \cap U_b = \emptyset$, (2) $U_a \cup U_b = U$ and (3) $||U_a| - |U_b|| \leq 1$.
21: Leave message *(Workload: U_a, U_b)* at v and explore your part using cautious walk. Make sure to reach the edge of your workload area using only safe links.
22: **end if**
23: **until** false

Lemma 3.2. *Every cycle C_i for $1 \leq i \leq k$ will eventually be explored.*

Proof. There is no waiting specified in the algorithm, therefore the only way the execution of the agent can be permanently stopped before finishing Algorithm EXPLORE is by entering the BH. According to Lemma 3.1 this can happen to at most one agent, therefore there will be an agent completing Algorithm EXPLORE. Since L is a traversal sequence, $\forall i \in \{1, 2, \ldots, k\}$ there is $u_j \in L$ such that $i = u_j$, i.e. each cycle has been explored. □

We are now ready for the main correctness theorem:

Theorem 3.3. *Algorithm* EXPLORE *correctly locates the black hole.*

Proof. From the construction of Algorithm EXPLORE it follows that the only node (if any) remaining unexplored after the exploration of a cycle is DN. From Lemma 3.2 we know that eventually all cycles will be explored. Since at every moment there is at most one DN, at the end there will be at most one unexplored node left. Since the BH is indeed present, at the end there will be exactly one unexplored node (DN) left and the black hole is located there. □

Let us now proceed to the complexity analysis.

Lemma 3.4. *The total number of moves spent in Algorithm* EXPLORE-CYCLE *for C_i is* $O(|C_i| \log |C_i|)$.

Proof. Note that the total number of moves before the first *(Workload: U_a, U_b)* message is placed is $O(|C_i|)$ (see Figure 4). Afterwards, in each round (delimited by placing *(Workload: U_a, U_b)* messages) the size of the unexplored area is at least halved, while $O(|C_i|)$ moves are spent by both agents. Thus, the number of rounds is $O(\log |C_i|)$ and the total number of moves is $O(|C_i| \log |C_i|)$. □

As each cycle of $\mathcal{C}$ is explored only once, from Lemma 3.4 we get that the total number of moves by the agents executing Algorithm EXPLORE-CYCLE is $O(\sum_{C_i \in \mathcal{C}} |C_i| \log |C_i|)$.

The only other place where agent's moves are specified is line 5. of Algorithm EXPLORE, in order to reach the next cycle in L. A cycle C_i may be involved several times, as it may appear many times in L, with each of those transits being either clockwise or counterclockwise. However, all segments traversed clockwise are edge disjoint (from the definition of L - that was our goal in defining *Cycle-DFT Sequence*). Moreover, the DN cannot move from one node of the cycle to another node of the same cycle, as only the last unexplored node of the cycle can become DN. That means that C_i can be transited counterclockwise at most once (when the DN is located between the entry points of neighbouring child cycles in the DFT) and both clockwise and counterclockwise cost of transiting C_i are $O(|C_i|)$. We get:

Lemma 3.5. *The total number of moves corresponding to executing line 5. of Algorithm* EXPLORE *on a cycle $C_i \in \mathcal{C}$ is $O(|C_i|)$*

Combining Lemmas 3.5 and 3.4 we obtain:

Theorem 3.6. *The complexity of Algorithm* EXPLORE *is* $O(\sum_{i=1}^{k} |C_i| \log |C_i|)$.

From Theorem 3.3, Theorem 3.6 and from the fact that every 2-connected graph has an *OVC* (Lemma 2.4) we now get the main theorem:

Theorem 1.1 (Main Result)
Let G be a 2-connected graph and let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be an open vertex cover by cycles of G. Then, it is possible for 2 agents to locate the black hole in G using $O(\sum_{i=1}^{k} |C_i| \log |C_i|)$ moves.

4 BH Location in Specific Networks

In this section we show that Algorithm EXPLORE can be applied to a wide variety of common networks, in many cases obtaining optimal results.

The *OVC* constructed by Lemma 2.4 (using ear decomposition) contains all edges of G. This means that simply applying Theorem 1.1 using this *OVC* might not result in acceptable complexity, especially for dense graphs. However, since *OVC* needs to cover only the vertices, not the edges, for many graphs we might be able to find much sparser *OVC*, possibly with small cycles as well.

That is exactly what we do in this section – showing explicit *OVC* construction for several commonly used classes of graphs, resulting in optimal BH location algorithms.

Perhaps the simplest construction is for 2-connected planar graphs:

Theorem 4.1. *There is a* BH *location algorithm for 2-connected planar graphs of complexity $O(n \log f)$, where f is the size of the second largest face in the graph.*

Proof. Since the graph is 2-connected, the faces of the graph themselves form an *OVC*, even after removing an arbitrary one of them. Applying Theorem 1.1 after removing the largest face, and using the fact that planar graphs have $O(n)$ edges (each of which is used in at most two retained faces) we get the theorem. □

A stronger result can be proven for Abelian Cayley graph of degree at least three, even though they include also dense graphs:

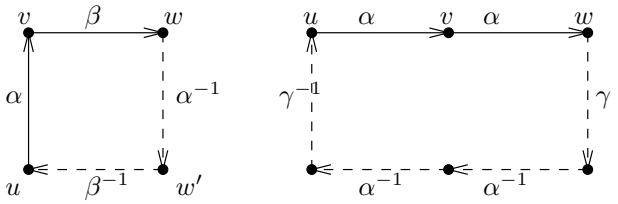

Figure 5. Short cycles in Abelian Cayley graphs, allowing to bypass node v if it is dangerous.

Theorem 4.2. *There is a* BH *location algorithm for Abelian Cayley graphs of degree ≥ 3 of cost $O(n)$.*

Proof. We show that there is an *OVC* with cycles of size 4 or 6. The theorem then follows directly from Theorem 1.1.

Consider an arbitrary spanning tree T of G and its Euler tour L. Let (u, v) and (v, w) be any two consecutive edges of L such that $u \neq w$. We show how to construct a cycle of length 4 or 6 which passes through (u, v) and (v, w). The union of these cycles over all such consecutive edges of L is then the sought *OVC*.

Let $v = \alpha u$ and $w = \beta v$, i.e. we get from u to v by applying generator α and from v to w by applying β. If $\alpha \neq \beta$, the cycle containing (u, v) and (v, w) is simply $\{u, v = \alpha u, w = \beta v, w' = \alpha^{-1}w, u = \beta^{-1}w'\}$. If $\alpha = \beta$, let γ be a generator different from α, α^{-1} and $(\alpha^{-1})^2$ (such γ must exist, as the graph is of degree at least 3). The cycle containing (u, v) and (v, w) is: $\{u, v = \alpha u, w = \alpha v, w' = \gamma w, v' = \alpha^{-1}w', u' = \alpha^{-1}v', u = \gamma^{-1}u'\}$ - see Figure 5. □

Several frequently studied topologies are Abelian Cayley graphs:

Corollary 4.3. *The* BH *can be located with $O(n)$ moves in hypercubes and (multi-dimensional) tori.*

While the following hypercube-related networks are not Abelian Cayley graphs, all of them have short ($O(1)$) cycles (roughly corresponding to some of the hypercube faces); these cycles together form *OVC* even without including the longer ($O(\log n)$) cycles. Thus, also for them we can similarly derive a linear bound on the number of moves using Algorithm EXPLORE:

Theorem 4.4. *The* BH *can be located with $O(n)$ moves in CCC, butterfly and wrapped-butterfly networks.*

The above results might suggest that a regular, highly symmetric structure is needed to locate the BH efficiently. That is not the case, though. In fact, the main advantage of out approach is that it applies equally well to less structured networks, which are nevertheless frequently used in practice.

Consider, for example a 2-dimensional mesh/torus network with several failed/missing nodes. The missing nodes create "holes" that disrupt the regular structure of the network. Nevertheless, such failures do not thwart BH location too much: The "holes" are treated as new faces and as long as the resulting graph remains 2-connected and the holes are not too big, Theorem 1.1 can be applied successfully:

Theorem 4.5. *Let G be a 2-connected 2-dimensional mesh/torus with holes of size at most f. Then the* BH *can be located in G using $O(n \log f)$ moves.*

5 Conclusions

We have presented a simple algorithm for locating BH using an *OVC* of the graph and we have shown that for many frequently studied topologies this approach results in optimal BH location algorithm.

The two main contributions of our technique are simplicity and robustness. The previous approaches were either fairly complex ([6]) or fragile ([3] – the task of finding traversal pairs for a given network is often non-trivial, and the result can change dramatically with a small change in the network). Moreover, there are classes of graphs (e.g. $O(n)$-diameter tori with small irregularities/holes) for which Algorithm EXPLORE is optimal, but neither of the previous approaches is.

Although we have not discussed in this paper the extension of Algorithm EXPLORE to dynamic networks (topic that we will leave for future work), it is foreseeable that this approach could scale well: in fact, a change in the topology involves local changes only in the Open Vertex Cover.

We have been able to construct "good" *OVC* for a relatively small class of graphs. In this regard, many intriguing problems are open; in particular:

Given a graph G, find an *OVC* $\mathcal{C}$ such that
$\sum_{C_i \in \mathcal{C}} |C_i| \log |C_i|$ is minimized.

This, in turn, poses many intriguing questions: What is the complexity of finding such optimal *OVC*? How can it be found? What about approximate solutions? Clearly, the same questions can be asked on more restricted instances, such as on *OVC* for graphs of given diameter, girth, etc.

One might hope that for every graph there is an *OVC* resulting in a asymptotically optimal algorithm. This is not so, unfortunately: Consider a ring of $\log n$ nodes, a central node and $\log n$ paths of length $n/logn$ each connecting the central node with the ring nodes. As there are $logn$ cycles of size $O(n/\log n)$ each, our algorithm would cost $O(\log n \times n/\log n \times \log(n/\log n)) = O(n \log n)$, while

the algorithm from [6] will locate the BH in $O(n)$ as the diameter is $O(n/\log n)$.

References

[1] D. M. Chess. Security issues in mobile code systems. In *Proc. Conf. on Mobile Agent Security*, LNCS 1419, pages 1–14, 1998.

[2] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in tree networks. In *Proc. 8th Int. Conference on Principle of Distributed Systems (OPODIS'04)*, pages 35–45, 2004.

[3] S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, and N. Santoro. Black hole search in in common interconnection networks. *Networks*, 2005. To appear.

[4] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Finding a black hole in an arbitrary network: optimal mobile agents protocols. In *Proc. of 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 153–162, 2002.

[5] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 2005. To appear.

[6] S. Dobrev, P. Flocchini, and N. Santoro. Improved bounds for optimal black hole search in a network. In *Proc. of 10th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2004)*, pages 111–122, 2004.

[7] O. Esparza, M. Soriano, J.J. Munoz, and J. Forne. Host revocation authority: A way of protecting mobile agents from malicious hosts. In *Proc. Int. Conf. onWeb Engineering (ICWE'03)*, LNCS 2722, 2003.

[8] M.S. Greenberg, J.C. Byington, and D. G. Harper. Mobile agents and security. *IEEE Commun. Mag.*, 36(7):76 – 85, 1998.

[9] F. Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Proc. of Conf on Mobile Agent Security*, LNCS 1419, pages 92–113, 1998.

[10] F. Hohl. A framework to protect mobile agents by using reference states. In *Proc. of the 20th Int. Conf. on Distr. Computing Systems (ICDCS 2000)*, 2000.

[11] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[12] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary graphs. In *Proc. 12th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO'05)*, pages 200–215, 2005.

[13] S.K. Ng and K.W. Cheung. Protecting mobile agents against malicious hosts by intention spreading. In *Proc. 1999 Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 725–729, 1999.

[14] R. Oppliger. Security issues related to mobile code and agent-based systems. *Computer Communications*, 22(12):1165 – 1170, 1999.

[15] T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts. In *Proc. of Conf on Mobile Agent Security*, LNCS 1419, pages 44–60, 1998.

[16] K. Schelderup and J. Ones. Mobile agent security - issues and directions. In *Proc. 6th Int. Conf. on Intelligence and Services in Networks*, LNCS 1597, pages 155–167, 1999.

[17] M.M. Syslo. An efficient cycle vector space algorithm for listing all cycles of a planar graph. *SIAM Journal on Computing*, 10(4):797–808, 1981.

[18] Jan Vitek and Giuseppe Castagna. Mobile computations and hostile hosts. In D. Tsichritzis, editor, *Mobile Objects*, pages 241–261. University of Geneva, 1999.

[19] H. Whitney. Non-separable and planar graphs. *Trans. Amer. Math. Soc.*, 34:339–362, 1932.