International Journal of Foundations of Computer Science

Using Scattered Mobile Agents to Locate a Black Hole in an Un-oriented Ring with Tokens

Stefan Dobrev
School of Information Technology and Engineering (SITE)University of Ottawa
800 King Edwaard Ottawa, Ontario, K1N 6N5, Canada

Nicola Santoro
School of Computer Science, Carleton University
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6, Canada

Wei Shi
School of Computer Science, Carleton University
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6, Canada

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

A *black hole* in a network is a highly harmful host that disposes of any incoming agents upon their arrival. Determining the location of a black hole in a ring network has been studied when each node is equipped with a *whiteboard*. Recently, the Black Hole Search problem was solved in a less demanding and less expensive *token model* with co-located agents. Whether the problem can be solved with scattered agents in a token model remains an open problem.
In this paper, we show not only that a black hole can be located in a ring using tokens with scattered agents, but also that the problem is solvable even if the ring is un-oriented. More precisely, first we prove that the black hole search problem can be solved using only three scattered agents. We then show that, with k ($k \geqslant 4$) scattered agents, the black hole can be located in $O(kn + n \log n)$ moves. Moreover, when k ($k \geqslant 4$) is a constant number, the move cost can be reduced to $O(n \log n)$, which is optimal. These results hold even if both agents and nodes are *anonymous*.

Keywords: Black Hole, Mobile Agent, Token, Ring, Scattered, Un-oriented.

1. Introduction

1.1. The Problem and Related Work

The reality of networked systems supporting mobility agents is that these systems are highly *unsafe*. Indeed, the most pressing concerns are all about security issues and mainly in regards to the presence of a *harmful host* (i.e., a network node

damaging incoming agents) or of a *harmful agent* (e.g., a mobile virus infecting the network nodes); for example, see Refs. [2,15,17].

The computational and algorithmic research has just recently started to consider these issues. The computational issues related to the presence of a harmful agent have been explored in the context of intruder capture and network decontamination; in the case of harmful host, the focus has been on the *black hole*, a node that disposes of any incoming agent without leaving any observable trace of this destruction Refs. [3,4,6,7,8,9,10,11,16]. In this paper, we continue the investigation of the black hole search problem.

A *black hole* (*BH*) models a network site in which a resident process (e.g., an unknowingly installed virus) deletes visiting agents or incoming data; furthermore, any undetectable crash failure of a site in an asynchronous network transforms that site into a black hole. In presence of a black hole, the first important goal is to determine its location. To this end, a team of mobile system agents is deployed; their task is completed if, within finite time, at least one agent survives and knows the links leading to the black hole. The research concern is to determine under what conditions and at what cost mobile agents can successfully accomplish this task, called the *black hole search* (*Bhs*) problem. The main complexity parameter is the *size* of the team; i.e., the number of agents used in the search. Another important measure is the amount of *moves* performed by the agents in their search.

The computability and complexity of *Bhs* depend on a variety of factors, first and foremost on whether the system is *asynchronous* Refs. [6,7,8,9,10] or *synchronous* Refs. [3,4,5,16]. Indeed the nature of the problem changes drastically and dramatically. For example, both in synchronous and asynchronous systems, with enough agents it is possible to locate the black hole if we are aware of its existence; however, if there is doubt on whether or not there is a black hole in the system, in absence of synchrony this doubt can *not* be removed. In fact, in an asynchronous system, it is *undecidable* to determine if there is a black hole Ref. [8]. The consequences of this fact are numerous and render the asynchronous case considerably difficult. In this paper we continue the investigation of the asynchronous case.

The existing investigations on *Bhs* in asynchronous systems have assumed the presence of a powerful inter-agent communication mechanism, *whiteboards*, at all nodes. In the whiteboard model, each node has available a local storage area (the whiteboard) accessible in fair mutual exclusion to all incoming agents; upon gaining access, the agent can write messages on the whiteboard and can read all previously written messages. This mechanism can be used by the agents to communicate and mark nodes or/and edges, and has been commonly employed in several mobile agents computing investigations. (e.g. see Refs. [1,14]). Although many research questions are still open, the existing investigations have provided a strong characterization of the asynchronous *Bhs* problem using whiteboard.

The availability of whiteboards at all nodes is a requirement that is practically expensive to guarantee and theoretically (perhaps) not necessary. This leads to the theoretically intriguing and practically important question of whether there are simpler and less expensive inter-communication and synchronization mechanisms

that would still empower the team of agents to locate the black hole. The research focus in particular has been on the *token model* commonly used in the investigations on graph exploration. In this model, each agent has available a bounded number of tokens that can be carried, placed in a node or/and on a port of the node, or removed from them; all tokens are identical (i.e., indistinguishable) and no other form of communication or coordination is available to the agents. Some natural questions immediately arise: is the *Bhs* problem still solvable with this weaker mechanism, and if so under what conditions and at what cost. Notice that the use of tokens introduces another complexity measure: the number of tokens. Indeed, if the number of tokens is unbounded, it is possible to simulate a whiteboard environment; hence the question immediately arises of how many tokens are really needed.

The problem of locating the *BH* using tokens has been examined in the case of *co-located* agents, that is when all the agents start from the same node. In this case, *Bhs* is indeed solvable Refs. [7,11]. In particular, in Ref. [11] it was shown that a team of two or more co-located agents can solve *Bhs* with $O(n \log n)$ moves and two (2) tokens per agent in a *ring* network. Notice that the ring is the sparsest bi-connected graph[a], and for which the number of moves for black hole search with whiteboards is the worst.

The problem becomes considerably more difficult if the agents are *scattered*, that is, when they start from many different sites. In particular, with scattered agents, the presence (or lack) of orientation in the ring and knowledge of the team size are important factors; here, oriented ring means all the agents in this ring are able to agree on a common sense of direction. This is true also in the whiteboard model Ref. [8]. In the token model, in particular, it is known that in an oriented rings it is possible to locate a *BH* with $O(1)$ tokens per agent performing $\Theta(n\ logn)$ moves Ref. [12].

1.2. Main Results

Clearly, communication between mobile agents is considerably more restricted (and complex) in a token model than in a whiteboard one. The question then is whether this additional constraint complicates significantly token-based solutions to the *Bhs*.

In this paper we prove that this is not the case for ring networks. In fact, we show that, for *Bhs* in a ring, the token model is computationally and complexity-wise as powerful as the whiteboard model, regardless of the initial position of the agents and of the orientation of the topology.

More precisely, first we prove that in an un-oriented ring, the *BH* can be located by a team of three or more scattered agents, each using $O(1)$ tokens; the total amount of moves being $O(n^2)$ in the worst case. We then show that, if there are k ($k \geqslant 4$) scattered agents, the *BH* can be located with $O(kn + n \log n)$ moves and $O(1)$ tokens per agent. When k ($k \geqslant 4$) is a constant number, the number of moves used can be reduced to $\Theta(n \log n)$, which is optimal. These results hold even if both agents and nodes are *anonymous*.

[a] bi-connectivity is required for *Bhs* in asynchronous systems Ref. [9]

We also point out that the results we achieve in this paper are an improvement over Ref. [12].

2. Model, Observations and Basic Tool

2.1. The Model and Basic Observations

Let $\mathcal{R}$ be a anonymous ring of n nodes (i.e. all the nodes look the same, they do not have distinct identifiers). Operating on $\mathcal{R}$ is a set of k agents $a_1, a_2, ..., a_k$. The agents are *anonymous* (do not have distinct identifiers), *mobile* (can move from a node to a neighboring node) and *autonomous* (each has computing and bounded memory capabilities). All agents have the same behavior, i.e. follow the same protocol, but start at the different nodes (and they may start at different and unpredictable times), each of which is called a *homebase* ($\mathcal{H}$ for brevity).

The agents can interact with their environment and with each other only through the means of *tokens*. A token is an atomic entity that the agents can see, place it in the middle of a node or/and on a port, or remove it. Several tokens can be placed on the same place. The agents can detect the multiplicity, but the tokens themselves are undistinguishable from each other. Initially, there are no tokens placed in the network, and each agent starts with some fixed number of tokens.

Note that the tokens are the only means of inter-agent communication we consider. There is no read/write memory (whiteboards) for the agents to access in the nodes, nor is there face-to-face recognition. In fact, an agent notices the presence of another agent by recognizing the token(s) it leaves. When we say two agents *meet*, there are two situations: two agents walking in the same direction *meet*, it means that one agent catches up with the agent in front of it in the same direction. Here *catch up* means finds the token(s) of the other agent in the same advancing direction. When two agents walking in the opposite direction meet, we mean that both agents find the token(s) of the other agent in the same node.

One of the nodes of the ring $\mathcal{R}$ is a *BH*. All the agents are aware of the presence of the *BH*, but at the beginning the location of the *BH* is unknown. The goal is to locate the *BH*, i.e. at the end there must be at least one agent that has not entered the *BH* and knows the location of the *BH*.

The primary complexity measure is *team size*: the number of agents needed to locate the *BH*. Other complexity measures we are interested in are *token count*: the number of tokens each agent starts with, *cost*: the total number of moves executed by the agents (worst case over all possible timings and starting locations).

The computation is asynchronous in the sense that the time an agent sleeps or is on transit is finite but unpredictable. The links obey FIFO rule, that is, the agents do not overtake each other when traveling over the same link in the same direction. Because of the asynchrony, the agents can not distinguish between a slow node and the *BH*. From this we get:

Lemma 1 *Ref. [8] It is impossible to find the Black Hole if the size of the ring is not known.*

As the agents are scattered, it could be the case that there is an agent in each neighbor of the *BH*, and both these agents wake up and make their first move towards the *BH*. This shows that:

Lemma 2 *Ref. [8] Two agents are not sufficient to locate the* BH *in scattered case without knowing the orientation of the ring.*

3. Algorithm *Shadow Check*

In this section, we present an algorithms that proves that the *BH* can be located with minimum of 3 scattered agents in an un-oriented ring.

3.1. Basic Ideas, General Description and Communication

We call a node/link *explored* if it is visited by an agent. A *safe (explored) region* consists of contiguous explored nodes and links. We call the last node an agent explored its Last-Safe-Place (LSP for brevity). In the scattered agents case, during the executing of *Bhs*, there are more than one *safe region*s in the ring. Our goal is to merge all the *safe region*s into one, which eventually includes all the nodes and links with the exception of the *BH* and the two links leading to the *BH*. Let us describe how this goal is going to be achieved.

Upon waking up, an agent becomes a *Junior Explorer (JE)*, exploring the ring to the right (from the viewpoint of the agent) until it meets another agent[b]. When two *JE*s meet, they both become *Senior Explorers (SE)*, and start exploring the ring in opposite directions. We call the explored area between these two *SE*s a *safe region* for them. A *SE* explores the ring, growing its *safe region* and checking after each newly explored node whether the *safe region* contains all the nodes except the *BH*. When two *SE*s moving in opposite directions *meet*, the two *safe regions* merge into a bigger *safe region*. The two meeting *SE*s become *Checker*s and check the size of the new *safe region*. There could be more than one such *safe region* in the whole ring. When a *JE* sees a *safe region*(i.e., it encounters a *SE*), it becomes *Passive* (stops being active).

When no unusual event occurs, each *SE* repeats the following cycle: it leaves two (*SE*s use two tokens) tokens on the port (if there is no token on this port) of the unexplored link on which it is going to move next. Once it reaches the node (if it is not the *BH*), the *SE* leaves there two tokens on the port from which it did not enter that node. It returns to the previous node, picks up the token(s) on the port it used, then returns to the last explored node. If, between cycles, an agent notices any unusual event (e.g., token situation changes on certain ports of a node), it stops the cycle and acts according to this interruption. The details of possible interruptions are explained later.

3.2. Using Tokens for Communication and coordination

The communication and coordination between the agents are described as follows:

[b] more precisely, finds a token of another agent

- One token on the port means a *JE* is exploring the link via this port.
- Two tokens on the port means a *SE* is exploring the link vis this port.
- One token on the port and one token in the middle means this is the node in which two opposite direction *JE*s *meet*.
- One token on each port means this is the node in which one *JE* catches up with another *JE* in the same direction.

We are going to explain the details of the algorithm in the next sub-sections. In order to make the algorithm simpler to understand, we describe the procedure "Junior/Senior Explorer" from the viewpoint of the agents, who agree on the same "right" direction. The procedure for all the agents who agree on the same "left" direction can be achieved by changing the word "right" into "left", and "left" into "right".

3.3. Procedure "Initialize" and "Junior Explorer"

A *JE* will eventually either end up in the *BH* or become a *Checker* upon *meet*ing a *SE* or a potential *SE*, or become a *SE* upon *meet*ing another *JE*. A potential *SE* refers to the status of a *JE* after it either met another *JE* in the same direction or a different direction, but before it becomes a *SE*.

Once an agent wakes up, it becomes an *JE* that will immediately go to the next node to its right after putting a token on the right port. There are 6 possible situations a *JE* may encounter upon arriving at its right neighbor node u. Now we can look at the details (expressed with respect the agent at hand) of each case:

- Case 1
  The agent A puts one token in the middle, then goes back to the left node. If there is a *SE* caught up with agent A, then A will become a *Checker* to the left. If the agent it just met (let's say B) in the opposite direction also left A a sign (a token in the middle), then A will become a *SE* to the left. If A is caught up by another *JE* in the same direction, A will pick up all the tokens, then become a *Checker* to the right.
- Case 2
  The agent A goes back to the left node. If A is caught up by another *JE*, it will become a *Checker* to the right. If A notices that the *SE* it just met in the opposite direction left A a sign(a token in the middle), then it will become *Passive* immediately. If the *JE* A just met left A a sign, then A will become a *SE* to the left.
- Case 3
  The agent A puts one token on the left port, then goes back to the left node. If A's token is still there, it will move this token to the left port, add one more token on the left port then it becomes a *SE* to the left. If either A sees

the sign a *SE* it just met left to it, or A is caught up by another *JE*, it will become *Passive* immediately.

- Case 4
  The agent A goes back to the left node. If A's token is still there, then it will pick the token and then become *Passive*. If A is caught up by a *SE*, then it will become *Passive*. If A is caught up by another *JE*, it will pick up the tokens, then become a *Checker* to the right.

- Case 5
  The agent A returns to the left node. If A is caught up by a *SE*, then it becomes *Passive*. If A is caught up by another *JE*, it will become a *Checker* to the right. If it notices that the *JE* it just met left it a sign (a token in the middle), then it will move the two tokens to the left port and become a *SE* to the left.

- Case 6
  The agent A puts a token on the right port then goes back to the left node. If A's token is still there, then it will pick the token and continue as a *JE*. If it is caught up by a *SE*, then it will become *Passive*. If A is caught up by an other *JE*, then it will become a *SE* to the right.

The pseudo code of procedure "Initialization" and "Junior Explorer" are in Algorithm 1, 2, 3, 4.

Algorithm 1 Algorithm *Shadow Check* — Procedure "Initialization" and "Junior Explorer"

```
1: procedure Initialization
2:     wakes up and puts a token on the right port then execute Junior Explorer(right)
3: end procedure
4: procedure Junior Explorer(right)
5:     loop
6:         walk to the right node
7:         if there is 1 token on the left port then
8:             execute Case 1
9:         else if there are 2 tokens on the left port then
10:            execute Case 2
11:        else if there is 1 token on the right port then
12:            execute Case 3
13:        else if there is 1 token in the middle and 1 on the right port then
14:            execute Case 4
15:        else if there is nothing in the node then
16:            execute Case 5
17:        else if there is 1 token on each port then
18:            execute Case 6
19:        end if
20:    end loop
21: end procedure
```

Algorithm 2 Algorithm *Shadow Check* — Procedure "Junior Explorer" — Cases 1 and 2

```
1: procedure Case 1
2:     put 1 token in the middle of the node, go back to the left node
3:     if there are 2 tokens on the right port then
4:         execute Checker(left)
5:     else if there is 1 token on the right port and 1 token in the middle then
6:         execute Senior Explorer(left)
7:     else if there is 1 token on each port then
8:         pick up all the tokens, execute Checker(right)
9:     end if
10: end procedure
11: procedure Case 2
12:     go back to the left node
13:     if there is 1 token on each port then
14:         execute Checker(right)
15:     else if there are 2 token on the left port then
16:         become Passive
17:     else if there is 1 token on the right port and 1 in the middle then
18:         execute Senior Explorer(left)
19:     end if
20: end procedure
```

Algorithm 3 Algorithm *Shadow Check* — Procedure "Junior Explorer" — Cases 3 and 4

```
1: procedure Case 3
2:     put 1 token on the left port, go back to the left node
3:     if there is only 1 token on the right port then
4:         move this token to the left port, add one more token on the left port,
5:         execute Senior Explorer(left)
6:     else if there are 2 tokens on the right port or 1 token on each port then
7:         become Passive
8:     end if
9: end procedure
10: procedure Case 4
11:     go back to the left node
12:     if there is 1 token on the right port then
13:         pick the token then become Passive
14:     else if there are 2 token on the right port then
15:         become Passive
16:     else if there is 1 token on each port then
17:         pick up the tokens, execute Checker(right)
18:     end if
19: end procedure
```

Algorithm 4 Algorithm *Shadow Check* — Procedure "Junior Explorer" — Cases 5 and 6

```
1: procedure Case 5
2:     return to the left node
3:     if there are 2 tokens on the right port then
4:         become Passive
5:     else if there is 1 token on each port then
6:         execute Checker(right)
7:     else if there is 1 token on right port and in the middle then
8:         move the 2 tokens to the left port, execute Senior Explorer(left)
9:     end if
10: end procedure
11: procedure Case 6
12:     put a token on the right port, go back to the left node
13:     if there is 1 token on the right port then
14:         pick the token and execute Junior Explorer(right)
15:     else if there are 2 tokens on the right port then
16:         becomes Passive
17:     else if there are 1 token on each port then
18:         execute Senior Explorer(right)
19:     end if
20: end procedure
```

3.4. Procedure "Checker"

A *Checker* is created when an agent realizes it is in the middle of two *SE*s exploring in different directions. The purpose of the *Checker* is to check the distance between the two *SE*s. A *Checker* keeps walking to the right until it either sees the token of a *SE* going to the right, or a node with one token on each port. If the distance is $n-2$, that means that two agents have died in the *BH*, and the only node left is the *BH*. Otherwise, it keeps walking to the left until it either sees the token of a *SE* going to the right, or a node with one token on each port. If now the distance is $n-2$, then it will become *DONE* (the *BH* is located), otherwise it becomes *Passive* immediately.

The pseudo code of procedure "Checker" is in Algorithm 5

Algorithm 5 Algorithm *Shadow Check* — Procedure "Checker"

```
1: procedure CHECKER
2:     repeat
3:         walk to the right
4:     until meet a node with either 2 tokens on the right port or 1 token on each port
5:     dist = 0
6:     repeat
7:         walk to the left increasing dist
8:     until there are 2 tokens on the left port or 1 token on each port of a node
9:     if dist = n − 2 then
10:        become DONE
11:    else
12:        become Passive
13:    end if
14: end procedure
```

3.5. Procedure "Senior Explorer"

A senior explorer will eventually either end up in the *BH* or locate the *BH*, or become a *Checker* upon *meet*ing another *SE* or a potential *SE*. A potential *SE* means a status of a *JE* after it either met another *JE* in the same direction or different direction, but before it becomes a *SE*. A *SE* walks to the right node. If it meets another *SE* in the different direction (we say: faces a *SE*), it will pick up all the tokens and become a *Checker* to the right. If it realizes it is the node which two *JE*s in the different directions met, it will then become a *Checker* to the right. If it realizes this node is where two *JE*s in the same direction met, it will then go back to the left port, pick up all the tokens and become a *Checker*. If it meets a *JE* going to the left, then it will pick the token on the left port, put two tokens on the right ports and go back to the left node and pick up the two tokens on the right port. Then the *SE* will execute the *check phase* to the left. If it meets a *JE* going to the right, then it will put one more token on the right port, go back to the left node; pick up the two tokens on the right port, then execute the *check phase* to the left. If the node is empty, the *SE* will then put two tokens on the right ports, go back to the left node; pick up the two tokens on the right port, then execute the *check phase* to the left.

Once a *SE* is in the *check phase*, it walks to the left until it either sees the token of a *SE* going to the right, or a token with one token on each port. If there are $n-2$ links in the *safe region*, then it will become *DONE*, otherwise it goes back to its LSP. If there is no token on the right port of its LSP, it then will become *Passive*.

The pseudo code of procedure "Senior Explorer" is in Algorithm 6.

Algorithm 6 Algorithm *Shadow Check* — Procedure "Senior Explorer" — right agents

```
1: procedure SENIOR EXPLORER(right)
2:     loop
3:         walk to the right node
4:         if there are 2 tokens on the left port then                ▷ face to face to a SE
5:             pick up all the tokens, execute CHECKER(right)
6:         else if there is 1 token in the middle of the node and 1 token on the right port
   then
7:             execute CHECKER(right)
8:         else if there is 1 token on each port then
9:             go back to the left port, pick up all the tokens, execute CHECKER(right)
10:        else if there is 1 token on the left port then
11:            put 2 tokens on the right port, pick up the token on the left port
12:            go back to the left node, pick up the two tokens on the right port
13:        else if there is 1 token on the right port then
14:            put 1 more token on the right port, go back to the left node; pick up the 2
   tokens on the right port
15:        else
16:            put 2 tokens on the right port, go back to the left node; pick up the 2 tokens
   on the right port
17:        end if
18:        Walk to the left until found a node with 2 tokens on the left port or 1 token on
   each port, increasing dist
19:        if dist = n − 2 then
20:            become DONE
21:        else
22:            Return dist steps to the right
23:            if there is no token on the right port of the node then
24:                become Passive
25:            end if
26:        end if
27:    end loop
28: end procedure
```

3.6. Analysis of Algorithm Shadow Check

According to Lemma 2, we assume there are at least three agents in the ring network. The following lemmas and corollary hold.

Lemma 3 *Eventually there is at least one* SE.

Proof. Given there are at least three agents in the ring, there are at least two *JE*s exploring the ring in the same direction. Sooner or later, the third *JE* will meet one of the other *JE*. Hence, in such case, at least two *SE*s will be created. But consider the situation of only three agents wake up during the entire execution, two agents died in the black hole as *JE*s. In this case, the third agent will sooner or later sees the token that a *JE* left before it went into the *BH*. The third *JE* then becomes a *SE* and starts exploring in the backward direction. Eventually, it will reach the token that the other dead *JE* left. Given the two *JE* both died in the black hole,

the distance (on the explored segment) between the two tokens is $n-2$. So the only *SE* will be able to tell the location of the black hole correctly. □

Corollary 1 *At most two agents enter the* BH.

Proof. Given before any explorer (*JE* or *SE*) explores a new node, it leaves one or two tokens in the current node as a marker. There is no other agent (a *JE*, a *SE* or a *Checker*) goes beyond the node with marking token(s). If this agent successfully explored a empty node, then it goes back to the node, in which it left a token. If the node is the same as before this agent left, the agent will then pick up the token and continue exploring the next node along the ring. This mechanism insures that there is no more than one agent explores a node via the same link. Given there are only two links adjacent to each node, there are two links leading to the *BH*. Hence there is at most one agent enters the *BH* from either link connects to the *BH*, which means there are at most two agents enters the *BH*. □

Lemma 4 *A* safe region *will be created.*

Proof. Lemma 3 shows there is at least one *SE* during the execution. According to the definition of a *SE*: two *JE* meeting (a *JE* sees the token of the other *JE*) each other will create two *SE*s. These two *SE*s will explore the ring in opposite directions starting from the same node. Hence *safe region* is created. When there are only three agents wake up in the entire execution and two *JE*s die in the *BH* before meet any other agent (see Lemma 1), the third *JE* will sooner or later meet a token which a *JE* left before die in the *BH*. A *safe region* is also created between the third *JE* and the LSP of another *JE* (died in the *BH*). □

Lemma 5 *Whenever the length of a* safe region *increases, it will be checked.*

Proof. In the case of there are at least two *SE*s in the network, The two explorers at each end of a *safe region* keep advancing. Each of them does not stop until meets another *SE* or die in the *BH* or finds out that the length of its *safe region* is $n-2$ (links) during *checking* phase.

When a *SE* meets another *SE*, according to the algorithm, both of them become *Checker*s. Hence, the two *safe region*s merge correctly.

Otherwise, when a *SE* meets a *JE*, the *JE* picks its token if it is still there then becomes *Passive* immediately. The *SE* will keep exploring after picking up the token of the *JE*.

When one *SE* dies in the *BH*, the other will keep exploring the ring until it figures out the length of the segment is $n-2$ during its own checking steps. In either case the segment still keeps grow correctly until reaches length $n-2$. □

Lemma 6 *The length of a* safe region *keeps increasing until contains* $n-2$ *links or* $n-1$ *nodes.*

Proof. A *safe region* is between two *SE*s. We observe the fact that each *SE* goes to check the location of the other *SE* in the same *safe region*, after exploring one more node. A *SE* terminates the algorithm as soon as it notices that the length of the *safe region* between the two LSP is $n-2$. This procedure ensures that if there are only two *SE*s left, they will not both die in the *BH*. If one *SE* died in the *BH*, the surviving agent will sooner or later advance to the node next to the *BH*, then

while seeking for its partner, it will notice the distance between the two LSPs is $n-2$. The only node left between the two unexplored links is the *BH*. □

Theorem 1 *Algorithm* Shadow Check *correctly locates the* BH *with* k *(*$k \geqslant 3$*) scattered agents in an un-oriented anonymous ring network. The total cost is* $O(n^2)$ *moves and,* 5 *tokens per agent.*

Proof. According to the above lemmas, three scattered agents are enough to locate the *BH*.

Now let us analyze the move cost: because there are k scattered agents, there is a maximum of $k/2$ *safe region*s in the ring. In procedure "Senior Explorer", an agent traverses its *safe region* once it explores one more node. There is a maximum $2n$ moves in each such traversal. There are at most n nodes in the ring, which means there are at most n such traversals. So, $O(n^2)$ moves are used. In procedure "Checker", the maximum number of each *check* is $2n$. A *check* can be triggered by either two *safe region*s merging or the *SE* this *Checker* follows exploring one more node. Given, there are no more than $k/2$ merges and n such *check*s, the total number of moves in procedure "Checker" is no more than $2n^2$. Hence, the total move cost is $O(n^2)$.

Now we analyze the token cost: a *JE* uses one token on the port to mark its progress. Once a *JE* meets another *JE*, one extra token is used to mark the node in which the two *JE*s meet and form a pair of *SE*s. This token will stay in the node until the algorithm terminates. A *SE* puts two tokens on the port as soon as it is created. It puts another two tokens on the port of the next node to mark progress. The first two tokens will be picked up and reused when exploring the next node. Hence, at most $1+2+2=5$ tokens are used by each agent. □

4. Algorithm *Modified 'Shadow Check'*

4.1. Motivation

In the previous section, we presented algorithm *Shadow Check* that handles the *Bhs* problem in an un-oriented ring with a minimum of 3 scattered agents and 5 tokens per agents. According to Theorem 1, an agent in one of the $k/2$ *safe regions*, traverses its *safe region* every time it explores one more node in order to check the size of this *safe region*. This is due to requiring minimum team size: Since there are only 3 agents in total, and because of the definition of *Checker*, it is obvious that there is at most one *Checker* formed in algorithm *Shadow Check*. So the explorers have to both explore the ring and check the size of the *safe region*. This cost (n^2) moves in the worst case.

After considering what kind of cost would we obtain if we had one more agent, we modify the algorithm slightly. The modified algorithm *Modified 'Shadow Check'* is such that:

- it can handle 4 or more scattered agents instead of 3;
- eventually there will be two *Checker*s formed and $O(kn + n\log n)$ moves are

used for an arbitrary k. If k ($k \geqslant 4$) is a constant number, the move cost can be reduced to $\Theta(n \log n)$

4.2. Modification

We can obtain algorithm *Modified 'Shadow Check'* by performing the following modifications on algorithm *Shadow Check*:

1. In procedure "Junior Explorer": change all the action "become *Passive*" of a *JE* in algorithm *Shadow Check*, into "become a *SE* in the same direction reusing the two tokens of the caught up *SE*", whenever this *JE* is caught up by a *SE*.

2. In procedure "Senior Explorer", there are two types of *SE*s: a *SE* with a *Checker* and a *SE* without a *Checker*.

   - a *SE* with a *Checker*: marked as three tokens on the port (the extra token is added by its *Checker*).
   - a *SE* without a *Checker*: marked as two tokens on the port.

   In both procedures, the 'check phase' in algorithm *Shadow Check* is deleted; Instead, as soon as it caught up with a *JE*, it will becomes a *Checker* in the opposite direction;

   - In procedure "a *SE* with a *Checker*": as soon as a *SE* with a *Checker* faces another *SE* with/without a *Checker*, it becomes *Passive*. Otherwise, it continues exploring.
   - In procedure "a *SE* without a *Checker*": as soon as a *SE* without a *Checker* faces another *SE* with/without a *Checker*, it becomes a *Checker*. Otherwise it continues exploring.

3. The procedure "Checker" is modified as follows:

   A *Checker* is created when it realizes it is in the middle of two *SE*s exploring in different directions. Once an agent becomes a *Checker*, it *check*s the size (number of nodes) of the *safe region* once, namely, it walks until the LSP of a *SE* with/without a *Checker*, then changes direction, walks and keeps counting the number of nodes it passes, until it arrives in the LSP of another *SE* without a *Checker*. We call this a *check* and this second *SE* the *SE* of this *Checker*. Let L denote the size of a *safe region*. Now this *Checker* puts an extra token on the port where its *SE* left two tokens. But if what this *Checker* meets is a *SE* with a *Checker*, then this *Checker* leaves a token in the middle of the node and becomes *Passive* immediately. There are two situations that can trigger a *Checker* to *check* again:

- Merging *check*: there is a token in the middle of the LSP of the *SE* of this *Checker*. This is caused by two *safe region*s merging. This *Checker* then picks up the token in the middle and performs a *check* in order to update L.
- Dividing *check*: this *Checker* followed its *SE* for $\lfloor (n-L)/2 \rfloor$ steps.

If while a *Checker* is following its *SE*, it notices that its *SE* became *Passive* (i.e., no token or not three tokens on the port in the next node), it then keeps walking until it sees the LSP of another *SE*. If it is a *SE* without a *Checker*, then this *Checker* becomes a *Checker* of this *SE*; otherwise, this *Checker* puts a token in the middle of this LSP and becomes *Passive* immediately.

If while a *Checker* is *check*ing the length of its *safe region* L, it notices that the *safe region* contains $n-1$ links. Then the *BH* is located: the only node left unexplored is the *BH*.

The pseudo code of procedure "Checker" follows:

Algorithm 7 Algorithm *Modified 'Shadow Check'* — Procedure "Checker"

```
1: procedure CHECKER
2:     loop
3:         dist = 0
4:         repeat
5:             keep walking to the right and increase dist
6:         until there are 2 tokens on the left port or 1 token on each port of a node or
   dist = n − 2
7:         if dist = 0 then
8:             become DONE
9:         else
10:            dist = 0
11:            repeat
12:                walk to the left and increase dist
13:            until there are 2 tokens on the left port or 1 token on each port of a node
14:            if dist = n − 2 then
15:                become DONE
16:            else
17:                follow the SE for ⌊(n − dist)/2⌋ nodes
18:            end if
19:        end if
20:    end loop
21: end procedure
```

4.3. Correctness and complexity

Given there are at least 4 agents in the ring, we know:

Lemma 7 *At least two* SE*s are formed in algorithm* Modified 'Shadow Check'.

Lemma 8 *Eventually at least two* Checker*s will be formed.*

Proof. Assume there are only 4 agents in the ring and all the agents are still *JE*s even after two of them died in the *BH*. The *JE*s left will either meet each other then become two *SE*s or eventually sees the token the *JE* died in the *BH*. If it is the first case, the two *SE*s will then explore the ring in opposite directions. Eventually each of them will see the token of the *JE* which died in the *BH*, then the *SE* will become a *Checker*. Hence there are two *Checker*s will eventually be created. If it is the second case, then there would be two pairs of *SE*s were formed if the two *JE*s did not go into the *BH*. This proves Corollary 7. The two surviving *SE*s will then exploring the ring in the opposite directions. Eventually they will meet and form two *Checker*s consequently. The case of more than 4 agents applies the same result. Hence, eventually there are at least two *Checker*s. □

Theorem 2 *Algorithm* Modified 'Shadow Check' *correctly locates the* BH

Proof. According to Corollary 1 and Lemma 7 and 8, eventually there will be two *Checker*s formed/left which keep checking the size of the *safe region* until the only *safe region* in the ring contains $n-1$ nodes or $n-2$ links. Hence the *BH* is correctly located. □

Theorem 3 *Algorithm* Modified 'Shadow Check' *correctly locates the* BH *in an un-oriented ring with* k *(*$k \geqslant 4$*) scattered agents, each having* 5 *tokens. When* k *is arbitrary, the total cost is* $O(kn + n \log n)$*. If* k *is a constant number, then the total move cost is* $\Theta(n \log n)$*.*

Proof. First we analyze the move cost: a *SE* with/without a *Checker* keeps exploring nodes along the ring without turning back. There are at most n such moves in total. In procedure "Checker", the maximum number of moves in each *check* is $2n$, and there are no more than $\log n$ *Dividing check*s, given a *Checker* does not proceed with the next *Dividing check* until it follows its *SE* for $\lfloor (n - L)/2 \rfloor$ steps. There are no more than $k/2$ *Merging Check*s in total, given k scattered agents can form at most $k/2$ *safe region*s. So the total number of moves in procedure "Checker" is no more than $kn + 2n \log n$. When k ($k \geqslant 4$) is a constant number, the total number of moves in procedure "Checker" becomes $O(n \log n)$, and the total move cost is $O(n \log n)$. The lower bound follows from the whiteboard model presented in [8]. Hence the total cost of moves is optimal when four or more ($O(1)$) scattered agents searching for a *BH* in an un-oriented ring.

Now we analyze the token cost: as we know that except for in procedure "Checker", a *Checker* uses one more token compared to a *Checker* in algorithm *Shadow Check*, no other modification affects the number of tokens used by each agent. According to the algorithm, a *Checker* uses a token only once in its lifespan. Also, according to Theorem 1: five tokens per agent are used in algorithm *Shadow Check*. Hence, 5 tokens per agent suffice to locate the *BH* in algorithm *Modified 'Shadow Check'*. □

5. Conclusion

In this paper, we proved that locating the Black Hole in an anonymous ring network using tokens is feasible even if the agents are scattered and the the orientation of the topology is unknown. Thus, we proved that, for the black hole search

problem, the token model is as powerful as the whiteboard regardless of the initial position of the agents.

From the results we obtain in this paper, we observe that there is a tradeoff between the team size (number of agents) and the costs (number of moves and number of tokens used). Since both algorithms we presented require only a constant number of tokens per agent, we are unable to simulate the *distance identity* presented in Ref. [8], which is crucial in order to achieve $\Theta(n \log n)$ moves with optimal team size (3 agents). But with one more agent, the token model is as powerful as the whiteboard with respect to *Bhs* in an un-oriented ring. And memorywise our algorithms represent a considerable improvement on the whiteboard model.

Acknowledgements

This section should come before the References. Funding information may also be included here.

References

1. L. Barriere and P. Flocchini and P. Fraigniaud and N. Santoro, "Rendezvous and election of mobile agents: Impact of sense of direction," *Theory of Computing Systems.* (2007) to appear.
2. D. M. Chess, "Security issues in mobile code systems," in *Proc. of* 1998 *Conf. on Mobile Agent Security (MAS'98)*, **LNCS 1419**, 1998, pp. 1–14.
3. C. Cooper and R. Klasing and T. Radzik, "Searching for black-hole faults in a network using multiple agents," in *Proc. of* 10^{th} *Int. Conf. on Principles of Distributed Systems (OPODIS'06)*, 2006, pp. 320–332.
4. J. Czyzowicz and D. Kowalski and E. Markou and A. Pelc, "Complexity of searching for a black hole," *Fundamenta Informatica.* **71(2-3)**(2006) 229–242.
5. J. Czyzowicz and D. Kowalski and E. Markou and A. Pelc, "Searching for a black hole in synchronous tree networks," *Combinatorics, Probability and Computing.* (2007) to appear.
6. S. Dobrev and P. Flocchini and R. Kralovic and G. Prencipe and P. Ruzicka and N. Santoro, "Optimal search for a black hole in common interconnection networks," *Networks.* **47** (2006) 61–71.
7. S. Dobrev and P. Flocchini and G. Prencipe and N. Santoro, "Exploring a dangerous unknown graph using tokens" in *Proc. of* 5^{th} *IFIP International Conference on Theoretical Computer Science (TCS'06)*, 2006, pp. 169–180.
8. S. Dobrev and P. Flocchini and G. Prencipe and N. Santoro, "Mobile search for a black hole in an anonymous ring," *Algorithmica.* (2007) to appear.
9. S. Dobrev and P. Flocchini and G. Prencipe and N. Santoro, "Searching for a black hole in arbitrary networks: Optimal mobile agent protocols," *Distributed Computing.* (2007) to appear.
10. S. Dobrev and P. Flocchini and N. Santoro, "Cycling through a dangerous network: a simple efficient strategy for black hole search" in *Proc. of* 26^{th} *International Conference on Distributed Computing Systems (ICDCS'06)*, 2006, pp. 57.
11. S. Dobrev and R. Kralovic and N. Santoro and W. Shi, "Black hole search in asynchronous rings using tokens" in *Proc. of* 6^{th} *Conference on Algorithms and Complexity (CIAC'06)*, 2006, pp. 139–150.

12. S. Dobrev and N. Santoro and W. Shi, "Scattered black hole search in an oriented ring using tokens" in *Proc. of* 9^{th} *Workshop on Advances in Parallel and Distributed Computational Models (APDCM'07)*, 2007, pp. to appear.
13. S. Dobrev and N. Santoro and W. Shi, "Locating a black hole in a ring using tokens: The case of scattered agents" in *Proc. of* 13^{th} *International Euro-Par Conference, European Conference on Parallel and Distributed Computing (Euro-Par'07)*, 2007, pp. to appear.
14. P. Fraigniaud and D. Ilcinkas, "Digraph exploration with little memory" in *Proc. of* 21^{st} *Symp. on Theoretical Aspects of Computer Science (STACS'04)*, 2004, pp. 246–257.
15. M. Greenberg and J. Byington and D. G. Harper, "Mobile agents and security," *IEEE Commun. Mag.* **36(7)** (1998) 76–85.
16. R. Klasing and E. Markou and T. Radzik and F. Sarracco, "Hardness and approximation results for black hole search in arbitrary networks," *Structural Information and Communication Complexity.* **3499** (2005) 200–215.
17. R. Oppliger, "Security issues related to mobile code and agent-based systems," *Computer Communications.* **22(12)** (1999) 1165–1170.