Locating a Black Hole in an Un-oriented Ring Using Tokens: The Case of Scattered Agents

Stefan Dobrev[1], Nicola Santoro[2], and Wei Shi[2]

[1] University of Ottawa, Canada, sdobrev@site.uottawa.ca
[2] Carleton University, Canada, {santoro,swei4}@scs.carleton.ca

Abstract. Black hole search in a ring network has been studied in a *token model*. It is known that locating the black hole in an anonymous ring using tokens is feasible, if the team of agents is initially *co-located*. When dealing with the *scattered* agents, the problem was so far solved only when the orientation of the ring is known.
In this paper, we prove that a black hole can be located in a ring using tokens with scattered agents, 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 fewer moves. Moreover, when k ($k \geqslant 4$) is a constant number, the move cost can be made optimal. These results hold even if both agents and nodes are *anonymous*.

Key words: Black Hole, Mobile Agent, Token, Ring, Anonymous, Asynchronous, Scattered, Un-oriented.

1 Introduction

1.1 The Problem and Related Work

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 a 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 [1–6].

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 Bh. In presence of a Bh, 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 Bh. 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 computability and complexity of Bhs depend on a variety of factors, first and foremost on whether the system is *synchronous* [1, 2, 6] or *asynchronous* [3,

4]. Indeed the nature of the problem changes drastically and dramatically: the former is a great simplification of the later. For example, if there is a doubt on whether or not there is a BH 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 BH [3]. In this paper we continue the investigation of black hole search problem in the asynchronous case.

The existing investigations on BHS in asynchronous systems have assumed the presence of a powerful inter-agent communication mechanism, *whiteboards* [3, 4], at all nodes. The availability of whiteboards at all nodes is a requirement that is practically expensive to guarantee and theoretically (perhaps) not necessary. In this paper, we consider a less demanding and less expensive inter-communication and synchronization mechanisms that would still empower the team of agents to locate the BH: the *token model*. 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.

The problem of BHS 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 [5]. In [5] 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.

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. In the token model, in particular, it is known that in an oriented ring it is possible to locate a BH with $O(1)$ tokens per agent performing $\Theta(n \log n)$ moves [7].

1.2 Main Results

In this paper 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 is $O(n^2)$ 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*.

Due to space limitations, the proofs of all lemmas are omitted.

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 *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 in the middle of a node or/and on a port, or remove. Several tokens can be placed on the same location. 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. Most importantly, 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*, meaning that one agent catches up with the agent in front of it in the same direction. Here *catch up* means finding the token(s) of the other agent in the same 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, and *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 a 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 cannot distinguish between a slow node and the Bh. From this we get:

Lemma 1. *[3] 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. *[3] Two agents are not sufficient to locate the* BH *in scattered case without knowing the orientation of the ring.*

3 Algorithm *Shadow Check*

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[3]. 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.

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

- 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 via 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.

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

The details of the algorithm are explained 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".

3.2 Procedure "Initialize" and "Junior Explorer"

Once an agent wakes up, it becomes a *JE* that will go to the next node to the right immediately. There are 6 possible situations a *JE* may encounter once in the right neighbor node. 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*.

- 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.

3.3 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.

3.4 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*.

3.5 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.

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

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

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

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

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.

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.*

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 that 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 [3]. 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 orientation of the ring 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 obtained in this paper, we observe that there is a trade-off 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*. (The *distance identity*, presented in [3], is the crucial technique used 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 black hole search in an un-oriented ring. And memorywise our algorithms represent a considerable improvement on the whiteboard model.

References

1. Cooper, C., Klasing, R., Radzik., T.: 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) 320–332
2. Czyzowicz, J., Kowalski, D., Markou, E., Pelc, A.: Complexity of searching for a black hole. Fundamenta Informatica. **71**(2-3) (2006) 229–242
3. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. Algorithmica **to appear** (2007)
4. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Searching for a black hole in arbitrary networks: Optimal mobile agent protocols. Distributed Computing **to appear** (2007)
5. Dobrev, S., Kralovic, R., Santoro, N., Shi, W.: Black hole search in asynchronous rings using tokens. In: Proc. of 6^{th} Conference on Algorithms and Complexity (CIAC'06). (2006) 139–150
6. Klasing, R., Markou, E., Radzik, T., Sarracco, F.: Hardness and approximation results for black hole search in arbitrary networks. Structural Information and Communication Complexity **3499** (2005) 200–215
7. Dobrev, S., Santoro, N., Shi, W.: 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) to appear