Scattered Black Hole Search in an Oriented Ring using Tokens

Stefan Dobrev†, Nicola Santoro§, Wei SHI§

†University of Ottawa
School of Information Technology and Engineering
Ottawa, K1N 6N5, Canada
sdobrev@site.uottawa.ca

§Carleton University
School of Computer Science
1125 Colonel By Drive,Ottawa, Canada
{santoro,swei4}@scs.carleton.ca

Abstract

A black hole *is a highly harmful host that disposes of visiting agents upon their arrival without any observable trace of the destruction. The problem of locating the black hole in a asynchronous* ring *network is known to be solvable by a team of mobile agents if each node is equipped with a* whiteboard*. A simpler and less expensive inter-communication and synchronization mechanism is provided by* tokens*: 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. All tokens are identical and no other form of communication or coordination is available to the agents.*
It is known that locating the black hole in an anonymous ring network using tokens is feasible when the team of agents is initially colocated *(i.e. they all start from the same host). Recently, the more difficult case when the agents are* scattered *(i.e., when the agents do not start from the same host) has also been examined and solutions requiring only $O(1)$ tokens per agent but using a total of $O(n^2)$ moves have been presented. The number of moves can be reduced to $O(kn + n \log n)$ if the number k of agents is known.*
In this paper, we study the impact of orientation and knowledge of team size on the cost of black hole location by scattered agents with tokens. We prove that, in oriented rings, the number of moves can be reduced from $O(n^2)$ to the optimal $\Theta(n \log n)$ using only $O(1)$ tokens per agent, without any knowledge of the team size. This result holds even if both agents and nodes are anonymous*. Interestingly, the proposed algorithm solves, with the same cost, also the* Leader Election *problem and the* Rendezvous *problem for the scattered agents despite the presence of a* BH*.*

Keywords: Mobile Agents, Token, Scattered, Ring, Asynchronous, Anonymous, Election, Rendezvous.

1 Introduction

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 [1, 12, 14].

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* (BH), a node that disposes of any incoming agent without leaving any observable trace of this destruction [2, 3, 5, 6, 7, 8, 9, 10, 13]. In this paper, we continue the investigation of the black hole search problem.

A *black hole* 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 computability and complexity of BHS depend on a variety of factors, first and foremost on whether the system is *asynchronous* [5, 6, 7, 8, 9] or *synchronous* [2, 4, 3, 13]. 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 [7]. 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. 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 black hole 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 [10, 6] . In particular, in [10] it was shown that a team of two or more colocated 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 (bi-connectivity is required for BHS in asynchronous systems [8]), 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 [7]. In the token model, in particular, it is known that in unoriented rings it is possible to locate a BH with $O(1)$ tokens per agent but performing $O(n^2)$ moves [11]. If the ring is oriented, $O(kn + nlogn)$ moves and O(1) tokens suffice, provided that the number k of agents a priori known [11].

In this paper, we study the impact of orientation and knowledge of team size on the cost of BHS using scattered agents with tokens. Since the protocols for unoriented rings do not require knowledge of the team size, we ask the natural question of whether, in oriented rings, knowledge of the team size is really necessary to reduce the cost from $O(n^2)$ while keeping the number of tokens per agents bounded.

In this paper we provide a definite positive answer. In fact, we prove that scattered agents can locate a BH in an oriented ring using only $\Theta(n \log n)$ moves in total and $O(1)$ tokens per agent, even if the number of agents is unknown. The proof is constructive: we present a protocol and prove that it achieves the claimed bounds.

This shows that, when moving from unoriented to oriented rings, the number of moves can be reduced from $O(n^2)$ to $\Theta(n \log n)$ without requiring a priori knowledge of the team size. Furthermore, for a team size larger than $O(\log n)$, the performance of the proposed algorithm is even better than that of the protocol for oriented rings with known team size.

Finally, let us point out that the proposed algorithm solves, with the same cost, also the problem of *electing a leader* among the scattered agents, in spite of the presence of a BH. It also solves with the same cost the *Rendezvous* problem despite the presence of a BH.

2 Model, Observations and Basic Tool

2.1 The Model and Basic Observations

Let $\mathcal{R}$ be a ring of n anonymous nodes (i.e. all the nodes look the same, they do not need to 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 need to have distinct identifiers), *mobile* (can move from a node to a neighbouring node) and *autonomous* (each has its own computing and limited memory capabilities). All agents have the same behaviour, 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 $\mathcal{H}$s are marked before the execution of an algorithm starts.

The agents can interact with their environment and with each other only through the means of *tokens*. A token is an atomic object that the agents can see, place in the middle of a node or/and on a port, or remove from there. Several tokens can be put on the same place. The agents can detect the multiplicity, but the tokens themselves are undistinguishable from each other. Initially, each $\mathcal{H}$ is marked by a token. Each agent starts with some fixed number of tokens (depending on the algorithm).

The basic computational step of an agent (executed either when the agent arrives to a node, or upon wake-up) is to *examine* the node (returns a triple of non-negative integers - multiplicity of tokens at the middle of the node, on the right port and on the left port, respectively), *modify* the tokens (by placing/removing some of the tokens at the current node) and either *fall asleep* or *leave* the node through either left or right port. The whole computational step is performed as an atomic action, i.e. as if it took no time to execute it.

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 travelling over the same link in the same direction.

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/communication. In fact, the agents might not be capable to see each other in the same node - they however see any tokens placed there.

One of the nodes of the ring $\mathcal{R}$ is highly harmful – it disposes of every agent that enters it, without leaving any trace of this destruction observable from the outside. Due to this behaviour, we will call this node *Black Hole* (or BH for brevity). 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: at the end there must be at least one agent that has not entered the BH and knows the location of the BH.

The complexity measures we are interested in are the number of tokens each agent starts with and the total number of moves executed by the agents (worst case over all possible timings).

Because of the asynchrony, the agents can not distinguish between a slow node and the BH; as a consequence, it is impossible to find the Black Hole if the size of the ring is not known [5]. Thus, we will assume that the agents have such a knowledge.

2.2 Cautious Walk with Token

At any time during the execution of the algorithm, each port will be classified either as *With Tokens* (i.e., one or more tokens have been placed on the node or its ports) or *Without Tokens* (i.e., no tokens on the node nor on its ports).

As we will see, in the protocol, having a predefined number of tokens on a port indicates that the link is currently being explored by an agent, and thus the link may be dangerous (it possibly leads to a Black Hole). To prevent unnecessary agent disappearances we will make sure that no two agents enter the BH over the same link. In order to achieve this, we have two basic rules for the agents. The first rule is the following:

- When an agent arrives at a node with a *With Tokens* port, it is not allowed to go across that port; that is, an agent can only leave through a *Without-Tokens* port.

In order to guarantee progress, the agents are also required to remove the token(s) whenever they learn that a "*With Tokens*" port in fact leads to a non-BH node. hence the second rule:

- When an agent marks "With Tokens" the port of node u leading to node v, after reaching node v, it immediately returns to u, it removes the tokens (to signal that v is not the BH and the link from u to v is *safe*), and then it resumes the algorithm by returning to v (unless otherwise instructed).

This process, which we call *Cautious Walk with Token* (*CWWT* for brevity) is an adaptation of the *cautious walk* technique used in systems with whiteboards. Notice that, during this process, an agent will not be interrupted except, possibly, from its returning to v (e.g., if it notice absence of token or extra tokens).

3 Algorithm *Pair Elimination*

3.1 General Description

The basic idea of the proposed algorithm, *Pair Elimination*, is to let all the agents try to form pairs as soon as they wake up. All the paired agents will eliminate all the single agents they meet. Each pair has a level. A pair increases its level each time it eliminates another agent. When two pairs meet, the higher level pair always eliminates lower level pair. Between pairs of the same level, the right most pair eliminates the left pair. As we will show, eventually only one pair will survive, and one of the two agents forming that pair will locate the black hole. Before we explain the algorithm in detail, we need to introduce some terminology.

Given an agent a_i and a node $v \neq$BH, we say that v has been *explored* by a_i if it has been visited at least once by a_i, *unexplored* otherwise; the *explored region* of a_i is the set of

non-BH nodes explored by a_i, and the *unexplored region* of a_i is the set of nodes unexplored by a_i.

The homebase $\mathcal{H}$ of an agent is identified by having one token in the middle of the node. An agent a_i starts exploring the ring from its $\mathcal{H}$, moving to the right; until it becomes part of a pair, its *explored region* is a segment with $\mathcal{H}$ as one of the two end nodes. We call the other end node of a_i's *explored region* a *LSP(Last Safe Place)* of a_i.

A node u, in which a pair of agents is formed, is called a *BP (Birth Place)* of that pair. The *birth place* of a pair is identified by having three or more tokens in the middle; A pair of agents explore the ring in opposite directions; when a pair is formed, there are two *LSP*s: the two end nodes of the union of *explored region* of the two agents.

Each Birth Place has a level; in particular, if it has $i+3$ tokens in the middle of the node, than that node is identified as a i*-BP (*i*-Level Birth Place)*. A BP is a 0-level-BP; as we will show, the highest level of a pair is at most $logn$. The level of BP is also the level of its pair.

A BP is called a *CP (Crown Place)* if at least one of the agent in the pair has finished exploring half of the ring size. The Crown Place *CP* is identified by having two tokens in the middle. If a BP is crowned, its priority becomes higher than that of any other level.

The priority relationship is shown in Figure 1.

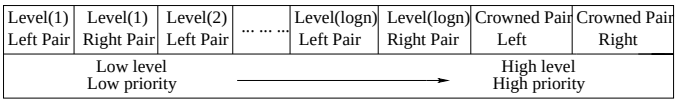

Figure 1. Pair Levels Priority table.

Let a meet a single agent b going to the right; note that, since agents do not see each other, this situation is detected by a finding tokens in predefined position. In this case, a leaves a message for b and becomes a left pair; when b sees the message from a, it becomes a right pair. Notice that since there are no whiteboard to write messages on, this communication is done only moving and placing tokens according to predefined rules. In particular, an agent leaves a *MN (Message Notice)* for its partner by stealing its token in the partner's LSP and placing it on the port of a node between the two LSPs. Once so placed, the token becomes a *Message sign* (*MS*) for its partner: the distance between the *MS* and the partner's LSP represents the information the agent wants to deliver.

After a pair is formed, the two paired agents start exploring the ring in opposite directions. They keep exploring a pre-defined number of nodes (initially, $\lfloor (n-1)/2 \rfloor$). If one finishes its share, it goes to seek its partner, and calculates the unexplored region according to the location of the partner. It leaves a message notice and a message sign to the partner then goes back to its LSP. We say this paired agent finished a *stage*.

In general, the following chain of actions by a paired agent a constitutes one *stage*: it explores a pre-calculated number of nodes, it checks the location of the partner, it leaves message for it, and then goes back to its LSP. The information in the message left for its partner is: the number of times a finished exploring its pre-calculated portion of the ring. This information is used by a paired agent to calculate the number of steps to take (i.e., the number of nodes to visit) in the next stage. When a paired agent b sees a message notice, it goes to check the information, and uses it to calculate the number of nodes it need to explore in the next stage.

If a paired agent goes into a BP of a lower level pair, it terminates this pair by removing all the tokens in their BP; it then goes to back to its LSP and continues exploring. When a left pair arrives at the BP of a same level pair, it terminates this pair by removing all the tokens in their BP; it then goes to its BP to increase its pair level, goes back to its LSP and continues exploring. When a left pair goes into a BP of a higher level pair, it becomes *Passive* immediately. When a right pair agent b arrives at the BP of a higher or same level pair, it goes back to its BP; if there is no token there, b becomes *Passive*; otherwise, it goes to the BP of the pair on the right to terminate that pair. At any time,if a paired agent a encounters a single agent c, a terminates c by stealing its token. We can now proceed with the presentation of the algorithm. We use the point of view of a right explorer (RE for brevity) to describe the algorithm. Most of the procedure for left explorers (LEs for brevity) can be achieved by changing the words "right" into "left" and changing "left" into "right" in procedure for the REs. The only procedure in which LE and RE behave differently is Right (Left) Exploring. We will explain it in subsection "Procedure Right/Left Exploring". In the algorithm, parameter $BPdist$ is used for an explorer to remember its BP. $EDist$ records the number of explored nodes between the two LSPs. $steps$ records the number of steps one explorer needs to explorer in each stage. $message$ is used for an explorer to remember the message the partner left to it.

3.2 Procedure "Form Pair"

This procedure is used to merge two single agents into a pair, so that they can cooperate to locate the BH. Initially, as soon as an agent wakes up, it moves right with *CWWT*. There are five situations a single agent can encounter when arriving at a node:

- A: it finds two or more tokens in the middle of a node. In this case, the node is either a pair's i-BP ($3+i$ tokens in the middle), or a crowned place (two tokens in the middle).

- B: it finds one token in the middle of a node for the second time. This means the agent knows there are at least two agents on its right side. So the agent can become passive immediately.
- C: it finds a token on the right port of a node (meets an agent walking with *CWWT* to the right).
- D: it finds out that there is no token in the port where it left its *CWWT* token, and there are no tokens in the center. It means that a paired agent terminated it.
- E: it finds out that there is no token in the port where it left its *CWWT* token, there are two or more tokens in the middle of the node. It means it now forms a pair with another agent.

Algorithm 1 Algorithm "Pair Elimination" — Initialize and procedure "Form Pair"

```
1: Initialize:Wake up; go to the right with CWWT, steps == 0, BPdist == 0, EDist == 0
2: procedure FORM PAIR
3:     keep walking right using CWWT until A, B, C, D or E happens
4:     if A, B or D happens then
5:         become passive
6:     else if C happens then
7:         become a Left Pair, move the token from port to the middle
8:         leave two extra tokens in the middle as a BP mark. Execute LEFT EXPLORING
9:     else if E happens then
10:        become a Right Pair, executes RIGHT EXPLORING
11:    end if
12: end procedure
```

3.3 Procedure "Right Exploring"

This procedure is used for a RE to explore the ring. A RE walks with *CWWT* to explore pre-calculated steps of the ring according to the position of the LE. The RE ignores any other agent except for its partner. There are five situations a RE can meet while exploring:

- A: it goes into a node with at least three but less number of tokens than its level in the middle. It is a BP that is in the lower level.
- B: it goes into a node with at least three but equal of bigger number of tokens than its level in the middle. It is a BP that is in the same or higher level.
- C: it finds no token on the port where it left its *CWWT* token.
- D: it finishes the pre-calculated $steps(\lfloor (n - EDist)/2 \rfloor)$ steps.
- E: it finds a token on the right port of a node.
- F: it determines $EDist == n - 2$. It means the explored region contains $n - 2$ nodes.

Algorithm 2 Algorithm "Pair Elimination" — Right Exploring

```
1: procedure RIGHT EXPLORING(steps, BPdist, EDist)
2:     keep walking to the right using CWWT and increasing BPdist, EDist and decrease steps, until A, B,C or D happens
3:     if A happens then
4:         pick the tokens in the node, then keep execute RIGHT EXPLORING
5:     else if B happens then
6:         go back to the BP
7:         if there are still tokens in its BP then
8:             check the current level, go back to its LSP, then execute RIGHT EXPLORING
9:         else become Passive
10:        end if
11:    else if C happens then
12:        execute CHECKING–RIGHT PAIR
13:    else if D happens then
14:        put a token on the right port, then execute SEEKING–RIGHT PAIR
15:    else if E happens then
16:        steal the token on the right port, then execute RIGHT EXPLORING
17:    else if F happens then
18:        become DONE
19:    end if
20: end procedure
```

3.4 Procedure "Left Exploring"

Procedure "Left Exploring" works almost the same as procedure "Right Exploring", except for the following situation: when a LE goes into the BP of the same level, it steals all the tokens, then goes back to its own BP to update its level. It becomes passive immediately if it goes into the BPs of a higher level pair. There are six situations a LE can meet while exploring:

- A: it goes into a node with at least three but smaller number of tokens than its level in the middle. It is a BP that is in the lower level.
- B: it goes into a node with equal number of tokens as its level in the middle. It is a BP which is in the same level.
- C: it goes into a node with bigger number of tokens than its level in the middle. It is a BP which is in higher level.
- D: it finds *CWWT* token was stolen in its LSP.
- E: it finishes the pre-calculated portion of ring. This pre-calculated part could be $(\lfloor (n - EDist)/2 \rfloor)$ after

seeking procedure, or ($\lfloor steps/2^{message} \rfloor$) after checking procedure. Here $UEDist$ is the number of nodes left unexplored in each stage.

- F: it finds a token on the left port of a node.
- G: it determines $EDist == n - 2$.

Algorithm 3 Algorithm "Pair Elimination" — Left Exploring

```
1: procedure LEFT EXPLORING(steps, EDist)
2:     keep walking to the left using CWWT and increasing BPdist and EDist, decrease steps, until A, B, C,D,E or F happens
3:     if A happens then
4:         pick the tokens, keep executing LEFT EXPLORING
5:     else if B happens then
6:         pick the tokens then go back to the BP
7:         if there are still tokens in its BP then
8:             increase the level, then return its LSP and execute LEFT EXPLORING
9:         elsebecome passive
10:        end if
11:    else if C happens then
12:        become Passive
13:    else if D happens then
14:        then execute CHECKING–LEFT PAIR
15:    else if E happens then
16:        put a token on the left port, then execute SEEKING–LEFT PAIR
17:    else if F happens then
18:        steal the token on the left port, then execute RIGHT EXPLORING
19:    else if G happens then
20:        become DONE
21:    end if
22: end procedure
```

3.5 Procedure "Checking"

Procedure "Checking" is called when the *CWWT* token of a paired agent is stolen. It can mean either it got eliminated by a higher level pair, or received a message notice from the partner. But it cannot decide which case it is, until later. In the first case, this agent will go back to its BP and realize all the tokens disappeared. In the second case, the agent will find a token on the port (it can also be in its BP), which is called message sign. It then will calculate the number of nodes to explore in the next stage, using the message it just collected. The number of steps between an explorer's LSP and the message sign is the number of stages the partner finished.

Algorithm 4 Algorithm "Pair Elimination" — checking

```
1: procedure CHECKING – RIGHT PAIR(BPdist,steps)
2:     keep going to the left and increase message until it either A or B happens.
3:     if A: there is no token in the middle of the BP then
4:         become Passive // its pair was eliminated by a higher level pair
5:         while not B, keep going to the left,keep increasing EDist
6:     end if
7:     if B: finds a token on the right port then
8:         steps=⌊steps/2^message⌋
9:         pick the tokens and walk back to its LSP, execute RIGHT EXPLORINGsteps, BPdist.
10:    end if
11: end procedure
```

3.6 Procedure "Seeking"

Procedure "Seeking" is used for an explorer to seek its partner. When a paired agent executes procedure "Seeking", it means it finished exploring pre-calculated number of nodes once. When a paired agent execute procedure Seeking for the first time, it means this pair reaches crowned level.

Algorithm 5 Algorithm "Pair Elimination" — Seeking

```
1: procedure SEEKING — RIGHT PAIR(EDist, BPdist)
2:     keep walking to the left and increasing EDist until goes back to the BP
3:     if A and there are at least two tokens in the middle of a BP then
4:         if there are more than two tokens in the middle then
5:             pick all but two tokens // crown the pair
6:         end if
7:     end if
8:     keep going to the left,keep increasing EDist
9:     if there is a token on the left port then
10:        if EDist == n − 2 then
11:            becomes DONE
12:        else
13:            pick the token, move to the right neighbor, put a token on the left port.
14:            keeps walking to the right until go back to its LSP
15:            execute RIGHT EXPLORING WITH ⌊(n − EDist)/2⌋ NEW STEPS.
16:        end if
17:    end if
18: end procedure
```

3.7 Analysis

3.7.1 Correctness

When there are only two scattered agents in the ring, algorithm *Pair Elimination* becomes algorithm *Divide with*

Tokens -[10].

First observe that pairs will indeed be formed:

Lemma 1 *At least one pair is formed.*

Next observe that the number of pairs that reach level i is at most half the number of pairs that reach level $i-1$:

Lemma 2 *Let x_i denote the number pairs that reach level i. Then $2x_i \leq x_{i-1}$.*

Proof: Consider three consecutive level $i-1$ pairs: A, B, and C; and assume that B reaches level i. This can happen only if (see Figure2): the right agent of pair B or the left agent of pair A have not reached the other pair's $(i-1)-BP$ before the left agent of pair B or the right agent of pair C reached the other $(i-1)-BP$, then pair C is eliminated by pair B. Otherwise, pair B will be eliminated by pair A. In general, taking any two neighbouring pairs, at most one of them will survive and reaches level i, the other one will be eliminated. Hence, the number of pairs that reached level i is at most half of the number of pairs that reached level $i-1$.

□

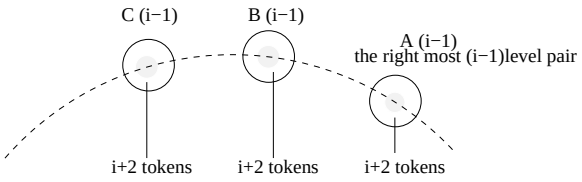

Figure 2. The BPs of three consecutive pairs A, B and C, which reached level $i-1$

Finally observe that at least one pair survives in each stage:

Lemma 3 *Let x_i denote the number pairs that reach level i. Then $x_i \geq 1$.*

Proof: Because of the priority rules, the right most pair that reaches level $i-1$ will eventually kill its "left neighbouring" level $i-1$ pair, and it will not be eliminated. So it will eventually reache level i.

□

By Lemmas 3.7.1, 2 and 3, the maximum possible level follows:

Corollary 4 *The maximum level is $logn$.*

Lemma 5 *Eventually at least one and at most two crowned pair(s) will be formed.*

Proof: If there is only one pair left in level i, then it will be crowned. For a pair to become crowned, one of its agents must have explored $\lfloor (n-1)/2 \rfloor + 1$ nodes and its BP must have not been reached by another pair. Clearly, if there are two segments with $\lfloor (n-1)/2 \rfloor + 1$ nodes in each, they must overlap ($(\lfloor (n-1)/2 \rfloor + 1) * 2 \geq n$). This means that there are at most two pairs with more than $\lfloor (n-1)/2 \rfloor + 1$ nodes between the two BPs. Hence, there can be at most two crowned pairs. See Figure 3.

□

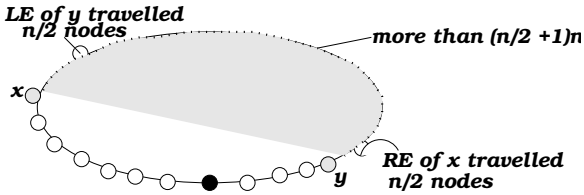

Figure 3. The BPs of three pairs: A, B and C, which reached level $i-1$

3.7.2 Complexity

Theorem 6 *Algorithm* Pair Elimination *correctly locates the BH within $O(n \log n)$ moves.*

Proof: In its life time an agent goes through three phases, if not eliminated during the executon. It starts as a single agent, then it becomes a paired agent, and finally it becomes a crowned agent. The number of moves performed by the agents during the execution of the algorithm can be separated according to these phases:

(1) By single agents: the worse case is that there is one agent per node except for the BH. They all start doing *CWWT* to the right. Because of the pair forming procedure, some of the agents' *CWWT* may be interrupted, so the number of moves before a single agent is formed, will be reduced from 3 to 2. But the worse case is still 3 moves per *CWWT*. So, there are $3(n-1)$ moves in total if there are $n-1$ single agents.

(2) By all paired agents of level i: each paired agent explore the ring with *CWWT*. It goes back to the BP to either increase the level/crow the pair or check the current level of the pair upon going into the BP of another pair. It then either becomes passive or goes back to the LSP. So the total number of moves with $n-1$ level i paired agents, is $(3+1+1)*(n-1)$. According to Corollary 4, there are maximum $logn$ levels. Hence, the total number of moves by all paired agents is $5(n-1)\log n$.

(3) By crowned agents: since there are at most two crowed pairs, and right crowned pair wins the left crowned pair. The

moves for right crowned pair to eliminate the left crowned pair is constant. Eventually there is one crowned pair left. Given the fact, the paired agents are co-located (they start locating the BH from the same node), according to algorithm *Divide with Tokens* -[10], the number of moves to locate the BH with two co-located agents is $n \log n$.

Hence, the total cost with scattered agents (at least two) is $O(n \log n)$ moves.

□

Theorem 7 *Algorithm* Pair Elimination *correctly locates the BH with four (4) tokens per agent.*

Proof: A token is needed to mark the $\mathcal{H}$ while a token is needed for *CWWT*. For marking a BP, an agent steals one token from the partner, then puts two more tokens in order to have three tokens in the middle of their BP. So two more tokens are needed. Each pair will not increase its level until it eliminated another pair. It also means the pair which won picks up the tokens in the other pair's BP. And one token is needed for a pair to increase the level shown in its BP. It shows, for increasing pair levels, there is no extra token needed. An explorer will leave a message to the partner only after it crowned their BP. Because to crown a BP only needs two agents in the middle of a node, from crown its i-BP, an explorer gains one more token. For the communication (leaving and checking message), an explorer steals a token from its partner first before it leaves a MS. Since one token is needed for a MS, no more token is needed for the MS. Hence, the total number of tokens each agent need is 4.

□

4 Concluding Remarks

In this paper, we show that a BH in an anonymous ring can be located by anonymous asynchronous agents scattered in an oriented ring, using $O(n \log n)$ moves in total and four (4) tokens per agent without any knowledge of the team size.

It is important to note that the proposed algorithm solves, with the same cost, also the problem of *electing a leader* among the scattered agents, in spite of the presence of a BH. It also solves with the same cost the *Rendezvous* problem despite the presence of a BH.

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] C. Cooper, R. Klasing, and T. Radzik. Searching for black-hole faults in a network using multiple agents. In *Proc. 10th Int. Conf. on Principles of Distributed Systems (OPODIS 2006)*, pages 320–332, 2006.

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

[4] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability and Computing*, to appear, 2007.

[5] S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, and N. Santoro. Optimal search for a black hole in common interconnection networks. *Networks*, 47:61–71, 2006.

[6] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Exploring a dangerous unknown graph using tokens. In *5th IFIP International Conference on Theoretical Computer Science (TCS 2006)*, pages 169–180, 2006.

[7] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, to appear, 2007.

[8] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agent protocols. *Distributed Computing*, to appear, 2007.

[9] S. Dobrev, P. Flocchini, and N. Santoro. Cycling through a dangerous network: a simple efficient strategy for black hole search. In , *Proc. of 26th International Conference on Distributed Computing Systems (ICDCS 2006)*, 2006.

[10] S. Dobrev, R. Kralovic, N. Santoro, and W. Shi. Black hole search in asynchronous rings using tokens. In 6^{th} *Conference on Algorithms and Complexity (CIAC '06)*, pages 139–150, 2006.

[11] S. Dobrev, N. Santoro, and W. Shi. Locating a black hole in a ring using tokens: The case of dispersed agents, 2007. `http://www.scs.carleton.ca/research/tech_reports/index.php?Year=2007`.

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

[13] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theoretical Computer Science*, to appear.

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