Black Hole Search with Tokens in Interconnected Networks

Wei Shi

University of Ontario Institute of Technology, Canada
wei.shi@uoit.ca

Abstract. We study the Black Hole search problem using mobile agents in three interconnected network topologies: hypercube, torus and complete network. We do so without relying on local storage. Instead we use a less-demanding and less-expensive *token* mechanism. We demonstrate that the Black Hole can be located with a minimum of two (2) *co-located* agents performing $\Theta(n)$ moves with $O(1)$ tokens, in each of these three topologies. Then we study the Black Hole search problem with *scattered* agents. We show that the optimal number of moves can be achieved with the optimal number of mobile agents using $O(1)$ tokens.

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

1 Introduction

Computational and algorithmic research has just recently started to consider security issues, 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), see [1,2]. With respect to the computational issues related to the presence of a harmful host, the focus has been on a *black hole* (*BH*), a node that disposes of any incoming agent without leaving any observable trace of this destruction [3, 4, 5, 6, 7, 8]. In this paper, we continue the investigation of the *black hole search* (*Bhs*) problem. Our research concern is to determine under what conditions and at what cost, within finite time, at least one of a team of mobile agents can survive and know all the links leading to a *BH*.

Most of the existing investigations on *Bhs* have assumed the presence of a powerful inter-agent communication mechanism, *whiteboards*, at all nodes. In the *whiteboard model*, each node has a local storage area where information can be written and read by the agents (e.g. see [9]). In this research, we investigate the *Bhs* in a *token model*. Communication between mobile agents is considerably more restricted (and complex) in a *token model* than in a *whiteboard* one: information-rich messages written to and read from a whiteboard must instead be represented using a limited number of tokens. The question then is whether this additional constraint complicates significantly token-based solutions to the *Bhs*. In this paper, we show that is not the case for the following three interconnection networks: hypercube, torus and complete network. We also answer the following question: under what conditions and at what cost is the *Bhs* problem solvable. Notice that the use of tokens introduces another complexity measure: the number of tokens. Indeed, if the number of tokens is unlimited, each information-rich message of a whiteboard environment can be mapped to a specific configuration of tokens and thus it is possible to simulate a whiteboard environment. The question then is how few agents are truly required by a solution to *Bhs*.

The problem of locating the *BH* using tokens has been examined in the ring topology in both cases of *co-located* agents (i.e., all the agents start from the same node in the network) [4, 10] and of *scattered* agents (i.e., the agents start from different unknown nodes in the network) [7, 8]. In [4] it is demonstrated that in order to locate the *BH* without *whiteboards*, $O(\Delta^2 M^2 n^7)$ moves suffice with $\Delta+1$ mobile agents and one token per agent.[1] Also, a recent solution proposed in [11] solves the *Bhs* problem in an arbitrary network with a team of two asynchronous agents with a map, using $\Theta(n \log n)$ moves, where n is the number of nodes. All existing solutions except for [7, 8], solve the *Bhs* problem using *co-located* agents. Here we propose to solve the *Bhs* problem for some specific network topologies, hoping to achieve better complexity than for the *Bhs* problem on an arbitrary network. We first consider the *Bhs* problem in hypercube, torus and complete network using *co-located* agents. We then study the *Bhs* problem in torus and complete network with a group of *scattered* agents. The *scattered* initial locations of the team of agents significantly complicate the solution of the problem. Yet, we show that for *Bhs* in these network topologies, 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. Also, with specific knowledge of the network, the number of moves executed by a team of two asynchronous agents can be reduced to $\Theta(n)$. The results hold even without using a map for a team of two agents in a complete network. In the *scattered* agents case, we show that the *Bhs* problem can be solved in a complete network with $O(n^2)$ moves, where n is both the number of *scattered* agents and the number of nodes in the network. We then show that, with 3 *scattered* agents and 7 tokens per agent, a black hole can be located with $\Theta(n)$ moves in a torus. We also observe that, when the number of *scattered* agents in a torus increases, the problem becomes significantly more complicated. A simple algorithm we develop solves *Bhs* with k $(k > 3)$ *scattered* agents, with $O(k^2 n^2)$ moves using only 1 token per agent.

2 Model, Assumptions and Terminology

Let $\mathcal{G} = (V, E)$ denote a simple connected undirected graph, where V is the set of vertices or nodes and E is the set of edges or links in $\mathcal{G}$. At each node $x \in V$, the incident edges are labeled by an injective mapping λ_x. Hence, each edge (x, y) has two labels, $\lambda_x(x, y)$ at x, and $\lambda_y(x, y)$ at y. $\lambda_x(x, y)$ and $\lambda_y(x, y)$ will be called the port numbers. We say a graph is *oriented*, if there is a globally

[1] Here, M is the number of edges in the graph, n is the number of nodes in the graph, and Δ is the maximum degree of the graph.

consistency of such labeling (or sense of direction) of all the edges (links), *un-oriented* otherwise [7,8].

Operating on $\mathcal{G}$ is a set of k agents $a_1, a_2, ..., a_k$. The agents have limited computing capabilities and bounded storage. They all obey an identical set of behavioral rules (referred to as the "protocol"), and can move from node to neighboring node. We make no assumptions on the amount of time required by an agent's actions (e.g., computation, movement, etc.) except that it is finite. Thus, the agents are *asynchronous* [5]. Also, these agents are *anonymous* (i.e., do not have distinct identifiers) and *autonomous* (i.e., each has its own computing and bounded memory capabilities). If *co-located*, agents start at the same node, called *homebase* ($\mathcal{H}$ for brevity). *Scattered* agents start at different $\mathcal{H}$s.

We postulate that, while executing a *Bhs*, 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, carry, place in the middle or on a port of a node, or remove. Several tokens can be placed at the same location. The agents can detect such multiplicity, but the tokens themselves are undistinguishable from each other. Initially, there are no tokens in the network, and each agent starts with $O(1)$ number of tokens.

The basic computational behavior of an agent (executed either when an agent arrives at a node, or upon wake-up) consists of three actions called *steps*. First an agent need to *examine* its current node and evaluate (as a non-negative integer) the multiplicity of tokens at the middle of the node and/or on its ports. (An agent therefore may have to evaluate several multiplicities for its current node.) Second, an agent may *modify* tokens (by placing/removing some of the tokens at the current node). Third, an agent may either become *Passive*(i.e., temporarily stop participating to the *Bhs*) or *leave* the node through a port. Finally, an agent may become *DONE*, namely terminate the whole algorithm. A step is performed as a single atomic (i.e., none interruptable) operation. We assume that there is a fair scheduling of the steps of the operation at the nodes, so that, at any node at any time, at most one computational step will take place, and every intended step is performed within finite time. This computation is *asynchronous* in the sense that the time an agent sleeps or travels is finite but unpredictable. It is known that in an *asynchronous* system, it is *undecidable* to determine if there is a *BH* or not [6]. The consequences of this fact are numerous and render the asynchronous case considerably difficult. Hence, in this research, we assume that there is one and exactly one *BH* in the network. All the agents are aware of the presence of the *BH*, but, at the beginning of the search, the location of the *BH* is unknown. The goal of this search is to identify all the links leading to the *BH*. At the end of the search, there must be at least one agent that has survived (i.e., not entered the *BH*) and knows the location of the *BH*.

We will consider three complexity measures for the *Bhs* problem. The first one is *size*: the number of agents needed to locate the *BH*. The other two complexity measures of interest are the *token size* (i.e., the number of tokens each agent needs to start with) and the *cost* (i.e., the total number of moves executed by the agents in the worst case over all possible timings). We study the following three topologies in such model and under such assumptions: hypercube, torus and complete network.

3 Basic Tool and Technique

3.1 *CWWT*

Cautious Walk with Token (henceforth *CWWT* for brevity) is an adaptation of the *cautious walk* technique used in systems with whiteboards [5]. It is a basic step in all our algorithms and is explained below.

At any time during the execution of this algorithm, a port will be classified either as *With Tokens* (i.e., one or more tokens have been placed on this port) or *Without Tokens* (i.e., no tokens on this port). The details of how to establish that a port is with or without tokens will differ across the algorithms that we introduce throughout this paper. During a *CWWT*, having a certain number of tokens on a port indicates that the link of this port is currently being *explored* by an agent. The exact number and location of tokens required to determine that a port is being explored may vary between the algorithms that use *CWWT*. Clearly, a port under exploration may be *dangerous* (i.e., possibly leading to the *BH*). To prevent unnecessary loss of agents, we require that no two agents enter the *BH* through the same link. In order to achieve this, we establish two basic rules for the agents that use *CWWT*. The first rule is:

When an agent a arrives at a node u with a port p under exploration, that agent is not allowed to move through port p. In fact, agent a can only leave through port p once p becomes *safe*.

In order to explain how a port becomes *safe*, consider an agent a that leaves token(s) on a port p of node u in order to explore the node v through the link of p. Our second rule captures how p becomes *safe*:

Upon reaching node v through port q, if v is not a *BH*, then a immediately returns to u and removes tokens on p. Thus, p necessarily becomes *Without-Tokens* and both its link and itself are thereafter considered *safe*. Port q is also considered *safe* once visited by a.

3.2 *Bypass* Technique

The *Bypass* technique is used in the algorithms to solve *Bhs* in both hypercube and torus with *co-located* agents. For those topologies, in contrast to a ring, each node has more than two links and ports adjacent to it. This significantly complicates the communication between agents using tokens, but also offers multiple paths between any of the two nodes in the graph. In fact, we get the following observation:

Observation 1. *Both hypercube and torus topologies contain one or more ring subgraphs, as shown in Figure 1.*

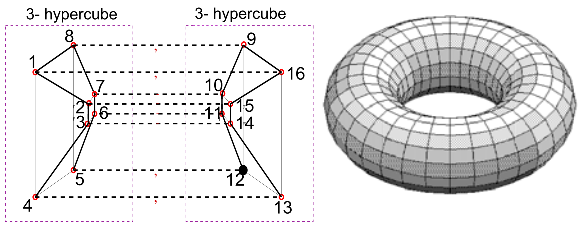

Fig. 1. Hypercube and Torus

According to our assumption that "there is one and only one *BH* in the network", we remark:

Observation 2. *It is impossible that the* BH *is in both* ring *a and* ring *b.*

We then call the ring without the *BH* a *safe ring*; a *dangerous ring* otherwise.

The basic idea of the *Bypass* technique is to use the links and nodes on a *safe ring* to create a bridge over an unknown node (possibly *BH*) on the *dangerous ring* that is under exploration by an agent. This bridge will allow a second agent to continue exploring the rest of the *dangerous ring*. This technique ensures a) that two agents do not explore the same node at the same time; and b) that all the nodes in the network get traversed using a linear number of moves, so that the total number of moves for locating the *BH* stays linear. Details follow:

Once in the "*Bypass*" procedure, an agent acts differently whether advancing in a *safe ring* or in a *dangerous ring*. Let A_d denote the agent that is exploring a node I in the *dangerous ring*, and A_s denote the agent that is going to *bypass* node I through path J, K, L, M, N (see Figure 2). When A_s arrives at node J , it moves the token(s) from port J_d to J_s if J_s is *without token*. Otherwise, A_s picks up the token(s) from port J_d, then A_s walks through J_s to node K. A_s then walks to node M through node L. If port M_s is *with token*, then A_s moves the tokens from port M_{s1} to port M_{s2}, then walks to the next node on the *safe*

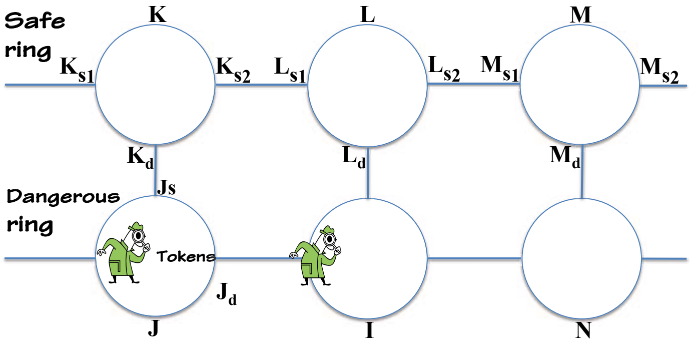

Fig. 2. Two agents executing "*Bypass* on a *dangerous ring and a* safe ring

ring. Otherwise, A_s leaves a token at port M_d, then it becomes ready to go back to the *dangerous* ring. From this point on, A_s becomes an agent exploring the *dangerous ring* M'_d in the next stage. If the old A_d does not die in node I, then it becomes an agent trying to *bypass* node N that is under exploration by the other agent. Namely, in the new stage, agent A_d will become a new A'_s. These two agents keep changing roles to *bypass* a node in the *dangerous ring* that is under exploration, until one dies in the *BH*.

4 *Bhs* with *Co-located* Agents

4.1 *Bhs* in Hypercube — Algorithm *Two Rings*

The following well-known property of a hypercube is the key to our solution to the *Bhs* problem in this topology:

Property 1. $\mathcal{Q}_d$ consists of two $d-1$-hypercubes connected by 2^{d-1} links labeled as d.

Given this property, we find a way for two mobile agents (given 2 is the minimum team size for the *Bhs* problem) to traverse the hypercube with tokens. The basic idea can be carried out using the following three steps[2]:

– let one agent stay in the common $\mathcal{H}$, and the other agent move to the other ring through the connecting link using *CWWT*.
– have both agents explore a Hamiltonian Cycle (i.e., a ring) of each $(d-1)$-hypercube according to a specific permutation (see below) with *CWWT*. After an agent has finished exploring its ring, we call this ring a *safe* ring, and call the other ring, which has not finished being exploring, a *dangerous* ring.
– let the agent that finished exploring the *safe* ring go to the other ring through a connecting link. This agent will help the other agent exploring the *dangerous ring*. It keeps walking on the *dangerous ring* until it sees the marker of the other agent. The two agents then repeat multiple stages of the *bypass* technique until one agent dies in the *BH* and the surviving agent finishes exploring all but one nodes in the entire hypercube. The only node the surviving agent has not visited is the *BH*.

The detail we need to address is how do we make the agents only walk on an appropriate Hamiltonian cycle and 2^{d-1} links labeled as d, in a labeled $\mathcal{Q}_d$. The following technique makes it possible:

We define a permutation that can construct a unique Hamiltonian cycle when a starting node is given. Let $\mathcal{P}_d$ be a permutation of length n: $\{p_1, p_2, ..., p_{n/2}, p_1, p_2, ..., p_{n/2}\}$. The sequence is constructed as follows:

[2] Due to the page limit, most Lemmas and Theorems and their proofs are omitted. Details can be found in: http://www.scs.carleton.ca/∼swei4/FinalThesis(VF2007May23).pdf

- when $d = 2$, $n = 2^2 = 4$, $\mathcal{P}_2$: $\{1, 2, 1, 2\}$;
- when $d = 3$, $n = 2^3 = 8$, $\mathcal{P}_3$: $\{1, 2, 3, 2, 1, 2, 3, 2\}$;
- when $d = 4$, $n = 2^4 = 16$, $\mathcal{P}_4$:
  $\{1, 2, 3, 2, 4, 2, 3, 2, 1, 2, 3, 2, 4, 2, 3, 2\}$;
- when $d = 5$, $n = 2^5 = 32$, $\mathcal{P}_5$:
  $\{1, 2, 3, 2, 4, 2, 3, 2, 5, 2, 3, 2, 4, 2, 3,$
  $2, 1, 2, 3, 2, 4, 2, 3, 2, 5, 2, 3, 2, 4, 2, 3, 2\}$;

  If we let $\mathcal{P}\prime_d$ denote the sequence from the second digit to the $2^{d-1^{th}}$ digit of $\mathcal{P}_d$, then:
- when $d = i - 1$, $n = 2^{i-1}$, $\mathcal{P}_{i-1}$: $\{1, \mathcal{P}\prime_{i-2}, i-1, \mathcal{P}\prime_{i-2}, 1, \mathcal{P}\prime_{i-2}, i-1, \mathcal{P}\prime_{i-2}\}$
- when $d = i$, $n = 2^i$, $\mathcal{P}_i$: $\{1, \mathcal{P}\prime_{i-1}, i, \mathcal{P}\prime_{i-1}, 1, \mathcal{P}\prime_{i-1}, i, \mathcal{P}\prime_{i-1}\}$

While d increases, each permutation $\mathcal{P}_d$ can be constructed by executing the following two steps on permutation $\mathcal{P}_{d-1}$:

a) replace the second occurrence of '1' found in the sequence by '*d*';

b) duplicate this modified sequence and append it to its own end (effectively creating a sequence that consists of the modified sequence followed by itself).

Given all the agents know the size of the hypercube $n = 2^d$, they can all come up with such a permutation individually. All their permutations will be the same, because they construct it according the same rules. Each element in the permutation represents a label of a link. Every such number indicates which link an agent is going to explore next.

Theorem 1. *Permutation $\mathcal{P}_d$ computed by an agent constructs a Hamiltonian cycle of $\mathcal{Q}_d$.*

4.2 *Bhs* in Torus — Algorithm *Cross Rings*

Informally, a torus is a mesh with "wrap-around" links that transform it into a *regular* graph: every node has exactly four neighbors. We develop an algorithm *Cross Rings*, to locate the *BH* in a torus with *co-located* agents when the torus is *oriented*, that is, when the ports of each node in the torus are consistently labeled as: *East*, *West*, *North*, and *South*.

Let $\mathcal{R} - NS$ denote a ring with only links labeled *South* and *North* in a labeled torus and, let $\mathcal{R} - EW$ denote a ring with only links labeled *East* and *West* in a labeled torus. Starting from a node, there are two obvious paths that allow an agent to traverse the torus and go back to the starting node. See Figure 3:

It is clear that a *north-south* ring $\mathcal{A}$ and an *east-west* ring $\mathcal{B}$ share exactly one node, say v. If node v is not the *BH*, we know the *BH* cannot be on both $\mathcal{A}$ and $\mathcal{B}$. We then get the following observation:

Observation 3. *Let 2 agents start from v. If we let one agent traverse the* north-south *ring $\mathcal{A}$, and another agent traverse the* east-west *ring $\mathcal{B}$, then there is at least one agent that survives its traversal.*

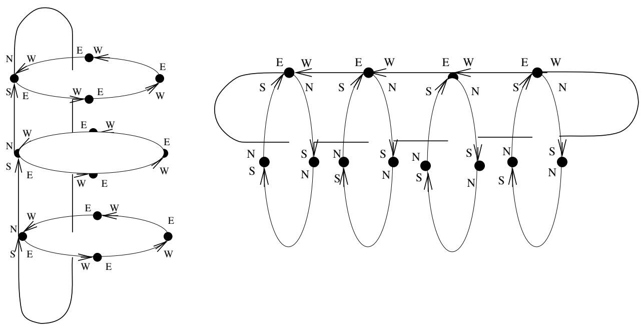

Fig. 3. Two paths that allow an agent to traverse all the nodes in a labeled $3 * 4$ torus: 1. an *east-west* ring, plus every *north-south* ring that starts with a node in this *east-west* ring, and 2. a *north-south* ring, plus every *east-west* ring that starts with a node in this *north-south* ring

If only one agent finishes traversing a ring (i.e., the other agent died in the *BH*), then we call this ring a *Base* ring. If both agents finish traversing their rings, then we call the ring that is traversed the earliest, a *Base* ring, which is also a *safe* ring. Hereafter, we assume that the *Base* ring is a *north-south* ring. (The algorithm would be essentially the same if the *Base* ring were an *east-west* ring). Now, we let the surviving agent(s) (either one or two) explore all the *east-west* rings, each of which starts from a node on the *Base* ring. In order to prevent the two agents from both dying in the *BH*, we let both agents explore a *dangerous* node using $CWWT$ with 1 token on a port.

Before one agent starts exploring an *east-west* ring, it puts 1 token in the middle of the *homebase* u. This agent then explores the first *east-west* ring. When this agent finishes exploring an *east-west* ring, it will move the single token it left in u to the next node to the North of u on the *Base* ring. We call the *east-west* ring marked by this token, a RUE (*R*ing *U*nder *E*xploration). An agent continues exploring *east-west* rings one by one, until it sees a token in the next node. It then puts a second token in the next node to the *north*, comes back to pick up the token it left in the previous node, goes to the next node to the *north* again and starts exploring a new *east-west* ring. Given there is only one *BH*, and there is no common node(s) shared by any two *east-west* rings, we obtain Lemma 1. Given one agent a_1 will finish exploring all but one *east-west* ring. The other agent a_2 is either exploring the RUE or died in the *BH* in the last RUE. Then a_1 will go and help a_2 to explore the last *east-west* ring. Because we assumed that one of the *north-south* rings is the *Base* ring, we say that an agent finishes a *stage* as soon as it finishes exploring an *east-west* ring. An agent a_1 will not visit a RUE (by a_2) until this is the only *east-west* ring left. Also, a_1 follows the path that a_2 took on this last *east-west* ring, until it sees the $CWWT$ token of a_2. Now a_1 and a_2 will execute the procedure "*Bypass* on Torus", sketched out earlier. Eventually the algorithm terminates when there is only one node that is not explored in the last *RUE*. The only node left unexplored is the *BH*.

Lemma 1. *Eventually all but one* east-west *rings are explored.*

4.3 *Bhs* in a Complete Network — Algorithm *Take Turn*

In this subsection, we present the solution to the *Bhs* problem in a complete network without using *sense of direction*, that is, no ports of any node is labeled. However, it is important to note that, even without a common labeling, the *co-located* agents share a common reference (e.g., indexing) mechanism for the $n-1$ links of their $\mathcal{H}$ and thus can share a common order of traversal of these links.

For simplicity, we will say that the links are traversed 'clockwise' when going from the lowest to the highest index, 'counterclockwise' otherwise (This is merely a convention and the actual order of traversal could be defined differently, as long as it is shared by the *co-located* agents.). A team of two *co-located* agents is used to solve the problem. We can imagine the complete network as a star-shape network with a node (which we will take to be the $\mathcal{H}$ of this pair of *co-located* agents) in the middle.

The idea is very simple: once an agent a_1 wakes up, it puts one token on a port of its node, which it views as its $\mathcal{H}$. a_1 then explores the node reachable from this port. When a_1 comes back to its $\mathcal{H}$ after exploring a node, if the token of a_1 is still at the port where it was left, then a_1 will move this token to the next port clockwise, and repeat this exploration step. Once the second agent a_2 wakes up, it moves the token of a_1 to the next clockwise, and explores the node accessible through this port. When an agent comes back from the exploration of a node, if it sees the token it left is missing, then this agent continues clockwise until it finds the port with one token. It moves this token to the next port clockwise and starts exploring another node through this port. During this process, an agent keeps counting the number of ports it visited (i.e., ports it used to access nodes to explore) or passed (i.e., ports that are between the port this agent just visited and the port that currently has a token). As soon as one agent notices that this total (of ports being counted) reaches $n-1$, it terminates the algorithm immediately. It is important to know that we use a variable *bhlocation* to record the location of the *BH*. Each time an agent a_i moves the token used by partner a_j to the next port, a_i resets the variable *bhlocation* to 0, then keeps increasing it by one each time it explores a new node. Also, variable *nCount* is incremented as ports are used. a_i terminates the algorithm as soon as it realizes *nCount* reaches $n-1$, at which point *bhlocation* indicates the location of the *BH*: the $bhlocation^{th}$ port counter clockwise leads to the *BH*.

Theorem 2. *Using two (2)* co-located *agents and one (1) token in total, the* BH *can be successfully located in a complete network of n nodes, with $\Theta(n)$ moves in total.*

5 *Bhs* with *Scattered* Agents

5.1 In a Complete Network

The algorithm for locating the *BH* with *scattered* agents follows: upon one agent waking up, it leaves a token in the middle of its $\mathcal{H}$ and waits. This agent starts executing algorithm *Take Turn* as soon as its token is moved to a port of its $\mathcal{H}$. If an agent wakes up in a node that has a token in the middle, then this agent starts executing algorithm *Take Turn* immediately. Once an agent wakes up in a node that has a token on a port of its $\mathcal{H}$, it becomes *Passive* immediately. Eventually, a maximum of $n/2$ pairs of agents will execute algorithm *Take Turn* and finally locate the *BH*. Given algorithm *Take Turn* requires n moves, $n/2 * n = n^2$ moves in total suffice with n *scattered* agents. 1 token per agent for n agents suffice to correctly locate the *BH*. Hence we get the following theorem:

Theorem 3. *Using n* scattered *agents, one (1) token per agent and $O(n^2)$ moves, the* BH *can be successfully located in an un-oriented complete network $\mathcal{K}_n$.*

5.2 In a Torus with Minimum Number of Agents

Again in this section, we assume the torus under investigation is *oriented*. We also assume no agent wakes up in the *BH*. It is possible that 4 agents could die immediately after the first move: one enters the *BH* through the *North* port, one through the *South* port, one through the *East* port, and one through the *West* port. In order to minimize team size, we program each mobile agent to enter each node through only the *South* or *West* ports[3], and thus a maximum of two agents die after the first move. Hence, we conclude:

Lemma 2. *At least* 3 *scattered agents are needed to locate the* BH *in an* oriented *torus .*

The basic idea for solving the *BH*s problem with *scattered* agents is to let two of the three agents form a pair that execute algorithm *Cross Rings* starting from the node (their $\mathcal{H}$) where they formed this pair. In the following paragraphs we will explain how the agents form a pair and how a pair of agents finds a *Base* ring. Then, the rest of the algorithm is almost the same as algorithm *Cross Rings*. In algorithm *Cross Rings*, there are only two agents working on the *Bhs*. But in the *scattered* agents case, we need to find out a way to eliminate the third *scattered* agents. Consequently, we work out a way for the third agent to become *DONE*, in order to simplify the communication between the working pair: as soon as an agent goes into a node with 2 tokens on any of a port (the indication of a single agent), it will pick up all the tokens and then continue.

[3] In order for an agent to traverse an oriented torus, each agent must visit at least two ports of each node.

Procedure "Initialization" and "Single Agent Explores a *north-south* Ring: upon waking up, an agent becomes a single agent and it immediately executes procedure "Single Agent Explores a *north-south* Ring" to the *north*. In procedure "Single Agent Explores a *north-south* Ring", an agent a_1 explores the *north-south* ring starting from node u ($\mathcal{H}$), with $CWWT$ (two tokens on the port). a_1 keeps counting the number of nodes in this *north-south* ring.

Case 1: When a_1 goes into a node with one token in the middle of a node, a_1 becomes $DONE$ immediately.
Case 2: When a_1 goes into a node with two tokens on the *east* port, it executes "Paired agent finds a *Base* ring" to the *north*.
Case 3: a_1 goes into a node with two tokens on the *north* port, it leaves one extra token in the middle of the node. It then executes "Paired agent finds a *Base* ring" to the *east*.
Case 4: When a_1 comes back to the node where it left its $CWWT$ tokens, if two tokens are in the middle and at least one token on the *east* port of the node, it then executes "Paired agent finds a *Base* ring" to the *north*.
Case 5: When a_1 goes into a node, if any of the following three situations happens, a_1 will become *Passive* immediately. All three situations indicate that a pair was formed. The situations are: either there is at least one token in the middle of the node (there may be also token(s) on a port of that node), or there is a token on the *north* port, or there is a token on the *east* port.
Case 6: When a_1 finished exploring the *north-south* ring, it then executes procedure "Single Agent Explores an *east-west* Ring".
Case 7: When a_1 comes back to the node where it left its $CWWT$ tokens, if all the $CWWT$ tokens are no longer there, it becomes $DONE$.
Case 8: When a_1 finishes exploring one *east-west* ring, it immediately explores the next *east-west* ring that starts from the next node to the *north* on the *north-south* ring. a_1 then executes procedure "Single Agent Explores an *east-west* Ring" again.

Procedure "Paired Agent Finds a *Base* Ring": As a single agent, as soon as a_1 sees two tokens on a port of a node (the *CWWT) of another single agent* a_2, it modifies the token configuration in this node and becomes a paired agent immediately. After a_1 becomes a paired agent, it executes procedure "Paired Agent Finds a *Base* Ring". Once an agent a_2 becomes a paired agent (after seeing the modified token configuration a_1 left to it) it also executes procedure "Paired Agent Finds a *Base* Ring". We call this node with the modified token configuration the *homebase* ($\mathcal{H}$ for brevity as used earlier) of these two paired agents. It is worth repeating that if a_1 executes "Paired Agent Finds a *Base* Ring" to the *north*, then a_2 will execute "Paired Agent Finds a *Base* Ring" to the *east*, or vice versa.

Upon starting "Paired Agent Finds a *Base* Ring" to the *north*. A paired agent a_1 keeps walking to the *north* with $CWWT$, until it goes back to the $\mathcal{H}$ of this pair. It is possible to have the following token configurations in this node:

1. there is 1 token on the *north* port and two tokens in the middle of their $\mathcal{H}$ (and maybe another token on the *east* port if the other paired agent a_2 is exploring the node to the *east* after being a paired agent). In this case, the *north-south* ring becomes the *Base* ring. a_1 informs a_2 of this result by picking up the token on the *north* port.

2. there are 2 or 3 tokens in the middle of the node. In this case, 2 tokens in the middle of the $\mathcal{H}$ shows that the second agent a_2 finished exploring the *east-west* ring before a_1 finished exploring the *north-south* ring. So, the *east-west* ring becomes the *Base* ring.

In either case, a_1 then keeps walking to the *east* until it sees 1 token in the middle of a node. It then executes algorithm *Cross Rings* to the *east* port. If there are 3 tokens in the middle (a_2 is exploring the first *east-west* ring as a paired agent), a_1 executes algorithm *Cross Rings* to the *east* port immediately. When agent a_2 walks back to the $\mathcal{H}$ of this paired agent after exploring an *east-west* ring, there are either

a) 2 tokens in the middle of the $\mathcal{H}$ (a_1 informed a_2 that the *north-south* ring is the *Base* ring). So a_2 keeps walking to the *north* until it arrives in the node with a token in the middle. It then executes algorithm *Cross Rings* to the *north.*

b) or 3 tokens in the middle of the $\mathcal{H}$ or 1 token on the *north* port and 2 tokens in the middle of their $\mathcal{H}$ (this means that not only a_1 informed a_2 that the *north-south* ring becomes the *Base* ring, but also that a_1 is exploring the *east-west* ring that a_2 just finished). Then a_2 will execute algorithm *Cross Rings* to the *north.*

c) or, in the third case, a_2 decides that the *east-west* ring is the *Base* ring and picks up the token on the *north* port of the pair's $\mathcal{H}$. a_2 then executes algorithm *Cross Rings* to the *east.*

During the execution of procedure "Paired Agent Finds a *Base* Ring", there are two other possible scenarios: 1) as soon as a_1 or a_2 goes into a node with 2 tokens on any of a port, it will pick up all the tokens then continue. 2) as soon as a_1 or a_2 notices its *CWWT* token is moved, it will continue using the *Bypass* technique as a paired agent.

5.3 In a Torus with *k Scattered* Agents

We also study the *Bhs* problem in a labeled torus with k ($k > 3$) *scattered* mobile agents. Here, k is not known to any of the agents. From the result shown in Theorem 4 we conclude that: not only an increase of team size does not help to reduce the total number of moves, but also drastically complicates the communication mechanism and increases the total number of moves performed during the *Bhs.*

Theorem 4. *Using k ($k > 3$)* scattered *agents and one token per agent, the* BH *can be successfully located using $O(k^2n^2)$ moves in a labeled torus with n nodes.*

6 Conclusion

In this paper, we developed a set of token-based algorithms for locating a *BH* in three interconnected network topologies. We sketched out solutions with both *co-located* agents and *scattered* agents. This set of algorithms suggests that the token model is computationally and complexity-wise as powerful as the whiteboard model, regardless of the topology of the network, and with the knowledge of a specific network topology, the cost of *Bhs* is improved from $\Theta(n \log n)$ to $\Theta(n)$. Moreover, in section 5, we show that *Bhs* with a team of *scattered* agents is rather complex but solvable in some dense graphs. We are now exploring a solution for *Bhs* on Hypercube with optimal complexity using *scattered* agents.

References

1. Greenberg, M., Byington, J., Harper, D.G.: Mobile agents and security. IEEE Commun. Mag. 36(7), 76–85 (1998)
2. Oppliger, R.: Security issues related to mobile code and agent-based systems. Computer Communications 22(12), 1165–1170 (1999)
3. Dobrev, S., Flocchini, P., Kralovic, R., Prencipe, G., Ruzicka, P., Santoro, N.: Optimal search for a black hole in common interconnection networks. Networks 47, 61–71 (2006)
4. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Exploring a dangerous unknown graph using tokens. In: Navarro, G., Bertossi, L., Kohayakwa, Y. (eds.) TCS 2006. IFIP, vol. 209, pp. 169–180. Springer, Heidelberg (2006)
5. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. Algorithmica 48(1), 67–90 (2007)
6. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Searching for a black hole in arbitrary networks: Optimal mobile agent protocols. Distributed Computing 19(1), 1–9 (2006)
7. Dobrev, S., Santoro, N., Shi, W.: Scattered Mobile Agents Searching for a Black Hole in an Unoriented Ring Using Tokens. International Journal of Foundations of Computer Science(IJFCS) 19(6), 1355–1372 (2008)
8. Dobrev, S., Santoro, N., Shi, W.: Scattered black hole search in an oriented ring using tokens. In: Proc. of 9th Workshop on Advances in Parallel and Distributed Computational Models (APDCM 2007), IEEE International, vol. (26-30), pp. 1–8 (2007)
9. Fraigniaud, P., Ilcinkas, D.: Digraph exploration with little memory. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 246–257. Springer, Heidelberg (2004)
10. Dobrev, S., Kralovic, R., Santoro, N., Shi, W.: Black hole search in asynchronous rings using tokens. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 139–150. Springer, Heidelberg (2006)
11. Flocchini, P., Ilcinkas, D., Santoro, N.: Ping Pong in Dangerous Graphs: Optimal Black Hole Search with Pure Tokens. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 227–241. Springer, Heidelberg (2008)