Searching for a Black Hole in Interconnected Networks using Mobile Agents and Tokens

Wei Shi

Faculty of Business and Information Technology
University of Ontario Institute of Technology
2000 Simcoe N. Street, Oshawa, Ontario, L1H 7K4, Canada

Joaquin Garcia-Alfaro

Institut Mines-Télécom, Télécom Bretagne
CS 17607, 35576 Cesson-Sévigné, France

Jean-Pierre Corriveau

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

Abstract

We study the impact of topological structure on the complexity of the *Black hole search* (*Bhs*) problem using mobile agents that communicate via tokens. First, we show that the token model can support the same cost as in the whiteboard model, despite the fact that communication between mobile agents is considerably more restricted (and complex) in a token model than in a whiteboard one. More precisely, in this paper, we focus on three specific topologies, namely: an asynchronous i) hypercube, ii) torus and iii) complete network. With knowledge of which of these topologies is being used, we present token-based solutions for *Bhs* where the number of moves executed by a team of two co-located anonymous agents can be reduced to $\Theta(n)$. These proposed solutions do not require the availability of a map and do not assume FIFO on either nodes or links.

Second, we consider the use of *scattered* agents for *Bhs* in an asynchronous i) torus and ii) complete network. We show that, using 3 *scattered* agents and 7 tokens in total, a black hole can be located with $\Theta(n)$ moves in an oriented asynchronous torus. Again, the solution does not assume FIFO on the links and nodes. If the number of *scattered* agents in a torus increases, cost is reduced but communication between these agents becomes significantly more complicated. We propose an algorithm that solves *Bhs* using k ($k > 3$) *scattered* agents, with only 1 token per agent, with $O(k^2n^2)$ moves.

Beyond theoretical proofs, we also discuss simulations of an actual system in order to evaluate our proposed solutions.

Key words: Black Hole, Mobile Agents, Tokens, Co-located, Scattered, Un-oriented, Simulation.

1. Introduction

1.1. Motivation

In the past decade, agent technology has shown great potential for solving problems in large scale distributed systems. A mobile agent is defined as abstract and autonomous software. Agents are versatile and robust in changing environments, and can be programmed to work in cooperative teams. Such team members may have different complementary specialties, or be duplicates of one another [1]. Mobile agent technology has been increasingly studied and several researchers (e.g., [2, 3]) discuss its strengths, such as the ability to: i) reduce network load, ii) overcome network latency, iii) support disconnected operations, iv) work in heterogeneous environments, v) allow asynchronous interaction, vi) enable remote searching and filtering, and vii) deploy new software components dynamically. For example, in recent years, a number of agent-based applications related to traffic control and management in different modes of transportation (including roadway [4, 5, 6, 7], railway [8, 9, 10], and air transportation [12, 13]) and healthcare [15, 16] have been reported.

Independently of such agent-based applications, recently there has been a significant increase in the use of IPads, smart phones, PDAs, and other wireless computer devices. Allowing such devices to communicate with each other and interact with other devices (such as vending machines, POS terminals, messaging systems, cars, GPSs (Global Positioning System), networked systems, central corporate data and applications, and the Internet) demands a highly efficient, dynamic wireless network that allows asynchronous interaction and supports disconnected operations. Mobile agent technology is a promising approach for addressing these challenges. In the same vein, Braz [14] states that web sites and other Internet services are not able to efficiently provide the full range of customization already desired by their clients (e.g., using the same information and organizing tools across many sites). In contrast, a mobile agent is not bound to the system/site where it begins execution. It has the unique ability to transport itself from one system of a network to another one. The ability to travel allows a mobile agent to move to a system that contains an object with which the agent wants to interact and then to take advantage of being in the same host or network as that object [7].

Security has also been identified as a key criterion for the acceptance of mobile agent technology. Computational and algorithmic research has started to consider security issues, mainly in regards to the presence of a *harmful host* (i.e., a network node damaging incoming agents) (see [1, 11, 17]). With respect to the computational issues related to the presence of a harmful host, the focus has been on a *black hole*, a node that disposes of any incoming agent without leaving any observable trace of this destruction [18, 19, 20, 21, 22, 23, 24, 25, 26, 27]. In this paper, we continue the investigation of the *Black hole search* (*Bhs*) problem.

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

The computability and complexity of *Bhs* depend on a variety of factors; first and foremost on whether the system is *asynchronous* [21, 22, 23, 28] or *synchronous* [18, 19, 26]. Indeed, the presence or absence of synchrony changes drastically the nature of the problem [24]. In this paper we pursue the investigation of the asynchronous case.

Most of the existing investigations on *Bhs* have assumed the presence of a powerful inter-agent communication mechanism, so-called *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. Each such whiteboard is accessible in fair mutual exclusion to all incoming agents (e.g. see [29]). In this research, we instead investigate *Bhs* in a *token model*. In that model, each agent has available a bounded number of tokens that can be carried, placed in a node and/or on a port of the node, or removed from them. Also, all tokens are identical (i.e., indistinguishable) and no other form of communication or coordination is available to the agents. We observe that the 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 *Bhs*. In this paper, we show that is not the case for the following three topologies: hypercube, torus and complete network. Within this specific context, we also answer the following question: under what conditions and at what cost is *Bhs* 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-based algorithm can be mapped to a specific configuration of tokens and thus it is possible to simulate a whiteboard environment. Using tokens, 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 for the ring topology in both cases of *co-located* agents (in which all the agents start from the same node) [21, 28] and of *scattered* agents (in which the agents start from different unknown nodes) [24]. In this paper, we first consider the use of a group of *co-located* agents to solve *Bhs* for a hypercube, a torus and a complete network. We then study *Bhs* in a torus and in a complete network for a group of *scattered* agents, which significantly complicates the solution.

Our decision to consider the hypercube network topology proceeds from the fact that such graphs are very versatile networks. For example, they have been used extensively to interconnect the processors of several parallel computers. Also, such an architecture allows for the emulation of a multitude of networks. Similarly, we remark that one of the networks that MasPar efficiently simulates is a torus, in part because of path diversity (i.e., there exists multiple minimum length paths between a source and a destination) and also because of better load balancing [30]. Finally, the complete network and/or mesh network topology is commonly used in Wireless Sensor Networks (WSNs), that is, in a wireless network consisting of spatially distributed autonomous entities that use sensors to monitor the condition of a particular environment or location. Such networks have gained a lot of practical relevance recently. In particular, Agilla [38, 37] is a new mobile agent middleware for wireless sensor networks that has received considerable researcher and industry attention in recent years. For example, the authors of [15, 16] present a healthcare application that allows the triage of victims in emergency scenarios to automatically update their medical condition. The proposed multiagent architecture combines Wireless Sensor Networks, an Electronic Triage Tag and a double multiagent system (Agilla-JADE) to achieve a low cost, infrastructure free, efficient system. The reliability of this system, which is crucial in the context of emergency care, rests entirely on the availability and correct working of all sensors and agents. In turn, this emphasizes

the need to be able to efficiently locate any compromised and/or malfunctioning sensor node that disposes of agents.

1.2. Main Results

In the context of *Bhs*, Flocchini et al. [31] proved that in networks of arbitrary but known topology, the pebble (or token) model of agent interaction is computationally as powerful as the whiteboard model; the complexity being exactly the same. More specifically, a team of two asynchronous agents, each endowed with a pebble and a map of the graph, can locate the *BH* with $O(n \log n)$ moves. In the same paper an open problem is pointed out: the topological structure impact on the complexity of *Bhs* with tokens. Intuitively, we "suspect" that the number of moves to locate the *BH* can be reduced in interconnected graphs in which multiple routes exist between any pair of nodes in the network. Conceptually, this topological characteristic offers 'shortcuts' between nodes. But, as much as shortcuts may reduce the number of moves, they also create a drawback: they complicate the communication and coordination between the agents. For example, in a complete network, each node is linked to every other node by an edge. A solution to leave a simple message (such as marking which links are explored by an agent) may require $O(\delta)$ tokens (where δ is the degree of the complete network). This clearly contradicts our goal of using $O(1)$ number of tokens (in contrast to using an unlimited number of tokens, which is equivalent to using a whiteboard, as previously mentioned) as the only means of communication between agents. The same observation holds for the torus and hypercube topologies: in contrast to a ring, in these topologies each node has more than two links adjacent to it, again leading to potentially more complex communication between agents via tokens. Consequently, it becomes an interesting goal to investigate whether there is a solution to *Bhs* when only a constant number of tokens are available in these topologies.

Except for our previous work presented in [24], all existing solutions to *Bhs* in asynchronous networks use *co-located* agents. In [32] and [33], Chalopin et al. did study *Bhs* using tokens with initially *scattered* mobile agents, but in a synchronous network. When a synchronous network is considered, the problem becomes much less complicated: each computation or movement of a mobile agent takes a known (instead of an unknown but finite) quantum of time. In this case, we can readily determine that an agent died in a *BH* if it does not resume its scheduled movement pattern within a predefined quantum of time. In asynchronous networks, it is not possible for us to distinguish the case of an agent stuck on a slow link/node (e.g., due to network traffic) from the case where it died in a *BH*. Thus, in contrast to asynchronous networks, in synchronous networks we can conclude that: a) we can detect whether there is a *BH* merely from an agent's failure to move within a given quantum of time; b) traversing the whole network in order to locate the *BH* is not necessary. Furthermore, when scattered agents are used to locate the *BH*, the initial locations and the wake up (i.e., being able to observe the environment, compute and move) time of these agents are not known. Thus, there is no communication between the *scattered* agents upon waking up. Conversely, when two or more agents start from the same node, namely agents are *co-located*, communication between these agents upon waking up leads to guaranteed coordination. Such coordination between the *co-located* agents can be achieved through whiteboard or tokens in asynchronous networks and through *time out* [32] or whiteboard [23] or tokens [32][33] in synchronous networks. But in fact, the ease with which agents can be synchronized makes the distribution of workload simpler and more efficient in synchronous networks than in asynchronous ones. And solving *Bhs* using *scattered* agents is a more general case of *Bhs* than using *co-located* agents. (Clearly, when the *scattered*

agents all happen to wake up from the same node, the problem becomes the *co-located* agents case.)

In the rest of this paper, we first prove that, in the context of *Bhs* in an asynchronous network with *co-located* agents, the token model can support the same cost as in the whiteboard model. More specifically, we offer solutions to *Bhs* for an asynchronous torus, hypercube and complete network using tokens. With specific knowledge of the network, the number of moves executed by a team of two co-located anonymous agents can be reduced to $\Theta(n)$. These solutions do not require the availability of a map and do not assume FIFO on either nodes or links.

We then consider the use of *scattered* agents for *Bhs* in an asynchronous torus and a complete network. We show that, using 3 *scattered* agents and 7 tokens in total, a *BH* can be located with $\Theta(n)$ moves in an oriented asynchronous torus. Again, the solution does not assume FIFO on the links and nodes. If the number of *scattered* agents in a torus increases, cost is reduced but communication between these agents becomes significantly more complicated. We propose an algorithm that solves *Bhs* using k ($k > 3$) *scattered* agents, with only 1 token per agent, with $O(k^2n^2)$ moves.

Beyond proofs, in order to verify our solutions and evaluate their performance, we developed simulations of an actual system using Java and Omnet 4++, which we discuss last.

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 identifiers. We say a graph is *oriented*, if there is a globally consistency of such labeling (or sense of direction) of all the edges (links), *un-oriented* otherwise [24, 25, 34].

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 only to a 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* [23]. 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). *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 or remove from the middle or a port of a node. 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 is 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. Second, an agent may *modify* tokens (by placing/removing some of the tokens present at the current node). Third, an agent may either become *Passive* (or equivalently, *fall asleep*) (i.e., temporarily stop participating to the *Bhs*) or *leave* the node through a port. Finally, an agent

may become *DONE*, in which case it stops executing the algorithm (and no longer participates in the search). Each computational step is performed as a single atomic (i.e., none interruptable) operation. We assume that there is 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*: the time an agent sleeps or is on transit is finite but unpredictable.

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 locate 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*.

3. Basic Tool and Technique

3.1. Cautious Walk with Token *(*CWWT*)*

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, that is, one of its links may possibly lead to the *BH*. Once a port is known to not lead to the *BH*, it is considered *safe*. To prevent unnecessary loss of agents, we require that no two agents ever enter the *BH* through the same link.

3.2. Bypass *Technique*

The *Bypass* technique is used in the algorithms that we propose to solve *Bhs* for hypercube and torus. For these two topologies, in contrast to a ring, each node has more than two links adjacent to it. This significantly complicates the communication between agents using tokens. But we notice the following key fact: both hypercube and torus topologies contain one or more non-intersecting ring subgraphs. Thus 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 that: a) two agents do not explore the same node at the same time; and b) all the nodes in the network get traversed using $O(n)$ moves, so that the total number of moves for locating the *BH* stays linear.

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 1). 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 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*.

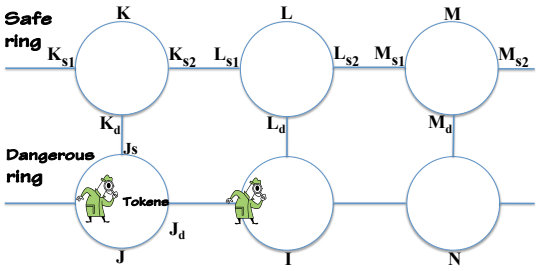

Figure 1: Two agents executing *Bypass* on a *dangerous ring* using the *safe ring*

4. *Bhs* with *co-located* agents

4.1. Bhs *in Hypercube — Algorithm Two Rings*

4.1.1. Basic Idea

The following well-known property of a hypercube is the key to our solution to *Bhs* 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 *Bhs*) to traverse the hypercube with tokens. The basic idea can be carried out using the following four steps:

1. Both agents start from a common $\mathcal{H}$. Each of them explore a Hamiltonian Cycle (i.e., a ring) of each $(d-1)$-hypercube according to a specific permutation (see below) with *CWWT*. More specifically, one agent stays in the ring (i.e., the Hamiltonian Cycle of one of the two $(d-1)$-hypercubes) in which the common $\mathcal{H}$ lies, and the other agent moves to the other ring through the connecting link using *CWWT*. After an agent has finished exploring its ring, we call this ring a *safe* ring, and call the other ring, which has not been fully explored, a *dangerous* ring.
2. 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 nodes but one in the entire hypercube. The only node the surviving agent has not visited is the *BH*.
3. When an agent notices that one node is marked by a *CWWT*, which means it is under exploration by the other agent, that first agent will *bypass* through a *safe* ring to the next node on the ring being currently explored.
4. The agent that explores $n-1$ nodes will survive and report the location of the *BH*.

The *bypass* technique will be used and only be used after one of the two rings in the hypercube is fully explored, that is, once a *safe* ring exists. The immediate 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:

1. when $d = 3$, $n = 2^2 = 4$, $\mathcal{P}_2$: $\{1, 2, 1, 2\}$;
2. when $d = 4$, $n = 2^3 = 8$, $\mathcal{P}_3$: $\{1, 2, 3, 2, 1, 2, 3, 2\}$;
3. when $d = 5$, $n = 2^4 = 16$, $\mathcal{P}_4$: $\{1, 2, 3, 2, 4, 2, 3, 2, 1, 2, 3, 2, 4, 2, 3, 2\}$;
4. when $d = 6$, $n = 2^5 = 32$, $\mathcal{P}_5$:
   $\{1, 2, 3, 2, 4, 2, 3, 2, 5, 2, 3, 2, 4, 2, 3, 2, 1, 2, 3, 2, 4, 2, 3, 2, 5, 2, 3, 2, 4, 2, 3, 2\}$;
   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:
5. when $d = i - 1$, $n = 2^{i-1}$, $\mathcal{P}_{i-1}$: $\{1, \mathcal{P}\prime_{i-2}, i-1, \mathcal{P}\prime_{i-2}, 1, \mathcal{P}\prime_{i-2}, i-1, \mathcal{P}\prime_{i-2}\}$
6. 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 to 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 $\mathcal{Q}_d$.*

Proof. There is a Hamiltonian cycle in a d dimension hypercube, when $d \geq 2$. Hence we assume we intend to construct a Hamiltonian cycle of a d ($d \geq 2$) dimensional hypercube. When $d = 2$, $\mathcal{P}_2$: $\{1, 2, 1, 2\}$, it is obvious a Hamiltonian cycle is constructed correctly. Now assume that when $d = i$, following the order of links indicated in $\mathcal{P}_i$: $\{1, \mathcal{P}\prime_{i-1}, i, \mathcal{P}\prime_{i-1}, 1, \mathcal{P}\prime_{i-1}, i, \mathcal{P}\prime_{i-1}\}$, a Hamiltonian cycle is constructed correctly.

When $d = i + 1$, we know there are two i dimensional hypercubes in the $i + 1$ dimensional hypercube due to the characteristics of the hypercube topology. We also know each i dimensional hypercube has a Hamiltonian cycle constructed according to $\mathcal{P}_i$ as per our assumption. As we can see, there are two links labeled 1 in the Hamiltonian cycle constructed according to $\mathcal{P}_i$. If we call the two rings (i.e., Hamiltonian cycles) that have 2^i nodes $\mathcal{R} - a$ and $\mathcal{R} - b$, we can merge $\mathcal{R} - a$ and $\mathcal{R} - b$ into one ring with 2^{i+1} nodes by the following three steps:

1. Remove one of the two links labeled 1 in both $\mathcal{R} - a$ and $\mathcal{R} - b$.
2. Link one of the two nodes currently adjacent to only $i - 1$ links in $\mathcal{R} - a$ to one of the two nodes currently adjacent to only $i - 1$ links in $\mathcal{R} - b$ with a link labeled $i + 1$ in the $i + 1$ dimensional hypercube.
3. Link the unique node currently adjacent to only $i - 1$ links in $\mathcal{R} - a$ to the unique node currently adjacent to only $i - 1$ links in $\mathcal{R} - b$ with another link labeled $i + 1$ in the $i + 1$ dimensional hypercube.

If we call this merged ring $\mathcal{R} - (i+1)$, we can observe that this $\mathcal{R} - (i+1)$ includes all the nodes in the $i + 1$ dimensional hypercube, because it includes all the nodes of $\mathcal{R} - a$ and $\mathcal{R} - b$ which, in turn, include all the nodes of two sub-hypercubes of the $i+1$ dimensional hypercube. Hence, when

$d = i + 1$, following the order of links indicated in $\mathcal{P}_{i+1}$: $\mathcal{P}_{i+1}$: $\{1, \mathcal{P}\prime_i, i+1, \mathcal{P}\prime_i, 1, \mathcal{P}\prime_i, i+1, \mathcal{P}\prime_i\}$, a Hamiltonian cycle can also be constructed correctly. This concludes the proof that Permutation $\mathcal{P}_d$ computed by an agent constructs a Hamiltonian cycle of $\mathcal{Q}_d$ ($n = 2^d$).

4.1.2. Algorithm Two Rings: Details

Two mobile agents a_1 and a_2 operate on $\mathcal{Q}_d$ starting from the same $\mathcal{H}$. The agent that first wakes up, called a_1, will explore the ring $\mathcal{R}_a$ contained in the $\mathcal{Q}_{d-1}$ according to the Hamiltonian cycle construction rule explained earlier. Before it starts exploring this sub-hypercube $\mathcal{Q}_a$, this agent leaves a token in the middle of its $\mathcal{H}$ in order to inform the partner (we call it a_2) to go to $\mathcal{R}_b$ and start exploring there. a_1 then explores its ring. When a_2 wakes up, it will explore $\mathcal{R}_b$ immediately. Recall that when an agent finishes exploring its ring, this ring becomes a *safe* ring. Once a_1 finishes exploring $\mathcal{R}_a$, if a_2 has not waken up yet, then a_1 will move the token that it left in the middle of their $\mathcal{H}$ to the port that leads to the link labeled n. To do so, a_1 informs a_2 to follow it to explore $\mathcal{R}_b$ together. Once a_2 wakes up, it will notice the token in its $\mathcal{H}$. Consequently it will go (with $CWWT$) to the ring ($\mathcal{R}_b$) in the second sub-hypercube $\mathcal{Q}_b$ through the link labeled n, then start exploring $\mathcal{R}_b$. Each agent explores a ring using $CWWT$ technique, until it notices its $CWWT$ token is moved by the partner.

Given the *BH* can only be in one of the Hamiltonian cycles, eventually one agent will finish exploring its ring. Let us assume that agent a_2 finishes exploring $\mathcal{R}_b$ first. Then a_2 will go to ring $\mathcal{R}_a$ to help a_1 to finish its job. Agent a_2 goes to look for a_1 in $\mathcal{R}_a$ if there is no token in their common $\mathcal{H}$. Since, the Hamiltonian cycle constructed according to what we proposed is unique given the same starting node, a_2 is able to follow a_1 on $\mathcal{R}_a$, instead of going in the opposite direction. Once a_2 catches up with (that is, sees the token of) a_1, the two of them will start using the *bypass* technique until they locate the *BH*.

4.1.3. Correctness and Complexity Analysis

Property 2. *Let $\mathcal{R}_1$ be one of the Hamiltonian cycle of a $\mathcal{Q}_{d-1}$ in a $\mathcal{Q}_d$ constructed according to $\mathcal{P}_{d-1}$, and $\mathcal{R}_2$ be the Hamiltonian cycle of the other sub-hypercube of $\mathcal{Q}_d$. There is an isomorphism between rings $\mathcal{R}_1$ and $\mathcal{R}_2$.*

Proof. The hypercube has node and edge symmetry [35]. For any pair of edges (u, v) and (u', v') in a d-hypercube $\mathcal{Q}_d$ there is an automorphism σ of $\mathcal{Q}_d$ such that $\sigma(u) = u'$ and $\sigma(v) = v'$. Such an automorphism can be found for any permutation π on $1, 2, 3, ..., n$ such that $\pi(k') = k$ where k and k' are the respective dimensions of (u, v) and (u', v') [35, 36]. We call v the *symmetric node* of v'. From Theorem 1, we know that the technique we use to construct a Hamiltonian cycle from a label permutation of a $(d-1)$-hypercube is unique. Let $\mathcal{Q} - A$ denote one of the two $(d-1)$-sub-hypercubes of $\mathcal{Q}_d$, and $\mathcal{Q} - B$ denote the other $(d-1)$-sub-hypercube of $\mathcal{Q}_d$. Given there is a automorphism between $\mathcal{Q} - A$ and $\mathcal{Q} - B$, there is also an automorphism between the two rings constructed out of the same permutation.

Since the two rings have no nodes in common, then:

Lemma 1. *Either $\mathcal{R}_1$ or $\mathcal{R}_2$ is a* safe *ring.*

Lemma 2. *Each one of the two rings will be explored by at least one agent.*

PROOF. Once the first agent wakes up, it will explore the ring of the $d-1$-sub-hypercube that contains the $\mathcal{H}$. Before it starts, it will leave a token in the middle of the $\mathcal{H}$ to inform the second agent to explore the other ring in order to prevent them from exploring the same ring. Only after an agent finds a *safe* ring, does it go to the other ring to help the partner agent to finish exploring the other ring. Hence each one of the two rings will be explored by at least one agent.

Lemma 3. *One and only one agent dies in the* BH.

PROOF. Given:

1. Both agents construct a Hamiltonian cycle of a $d-1$-sub-hypercube based on the same permutation;
2. The two $d-1$-sub-hypercubes that are connected by 2^{d-1} links labeled d, are automorphic.

We conclude that the two agents share the same sense of direction on both rings. Once an agent, say a_1 finds a *safe* ring, it will go to the other ring and explore the node of the ring in the same order as its partner, say a_2, did. Agent a_1 follows the route that a_2 traversed until it sees the $CWWT$ token a_2 left. Then the two agents will start exploring the *dangerous* ring using the *bypass* technique. According to this *bypass* technique, two agents never explore the same node in that ring. The algorithm terminates as soon as an agent has explored $n-1$ nodes. Hence, only one agent dies in the *BH*.

Theorem 2. *The* BH *is correctly located by the surviving agent.*

PROOF. According to Lemma 2, and Lemma 1, at least one agent will eventually finish exploring a *safe* ring. As we mentioned in Lemma 3, this agent will go to help the other agent exploring the second ring using the *bypass* technique. We know from Lemma 3, that there is one and only one agent that survives. The algorithm terminates as soon as an agent explored $n-1$ nodes. Hence the *BH* is correctly located.

Lemma 4. *Two tokens in total suffice to locate the* BH *in a labeled hypercube with co-located agents.*

PROOF. First, when the algorithm starts, one token is needed for the agent that wakes up first. Second, each agent needs one token to do $CWWT$ in both exploring and *bypass* stages. Last, the token used by the first agent in order to inform the second agent of its awakening can be reused by the second agent (which is the one that wakes up later than the first agent). Hence, 2 tokens in total suffice to locate the *BH* in a labeled hypercube with co-located agents.

Lemma 5. *$O(n)$ moves in total suffice using Algorithm Two Rings.*

PROOF. If we break down this algorithm into "procedures", then in procedure "Find a *safe* ring", each of the two agents requires a maximum of $3*(n/2)$ moves to explore a Hamiltonian cycle of a $d-1$-sub-hypercube of $\mathcal{Q}_d$. So, $O(n)$ moves is sufficient. In procedures "*Bypass*" and "Back to the *dangerous* ring", for every *bypass*, a linear number of moves is required. Therefore, even if there are $n/2$ such *bypass* steps, a linear number of moves is still sufficient. Hence, the total number of moves is linear.

According to the lemmas proved above, and following the lower bound from the whiteboard model presented in [20], we can conclude:

Theorem 3. *Using 2 co-located agents, 2 tokens in total and $\Theta(n)$ moves, the* BH *can be successfully located in a edge labeled hypercube with n nodes.*

4.2. Bhs *in Torus — Algorithm Cross Rings*

As in the ring topology, in a torus, the number of edges adjacent to each node is fixed regardless of the number of dimensions or nodes. Informally, the torus is a mesh with "wrap-around" links that transform it into a *regular* graph: every node has exactly four neighbors. The ports of each node in the torus are consistently labeled: *East*, *West*, *North*, *South*. Given these specific topology characteristics, we develop an algorithm *Cross Rings*, to locate the *BH* in a torus with *co-located* agents.

4.2.1. Algorithm "Cross Rings"

Let $\mathcal{R} - NS$ denote a ring with only the links labeled *South* and *North* in a labeled torus and, let $\mathcal{R} - EW$ denote a ring with only the links labeled *East* and *West* in a labeled torus. We also call $\mathcal{R} - NS$ a *north-south* ring, $\mathcal{R} - EW$ an *east-west* ring. Starting from a node, there are two obvious paths that allow an agent to traverse the torus and go back to the starting node. They are (see Figure 2):

1. the *east-west* ring that includes the starting node, plus every *north-south* ring that starts with a node in this *east-west* ring;

2. the *north-south* ring that includes the starting node, plus every *east-west* ring that starts with a node in this *north-south* ring

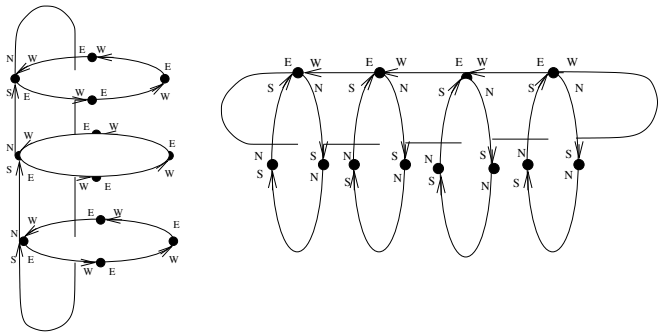

Figure 2: Two paths that allow an agent to traverse all the nodes in a labeled $3 * 4$ torus

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 1. *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.*

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 $\mathcal{R} - EW$, it puts 1 token in the middle of its *homebase* u. u is a node on both the *Base* ring and the *east-west* ring $\mathcal{R} - EW$. We call the *east-west* ring with 1 token in the middle of u an RUE (Ring Under Exploration), the 1 token in the middle of u, an UET (a token to indicate the ring is under exploration). This agent then explores this *east-west* ring $\mathcal{R} - EW$. When this agent finishes exploring $\mathcal{R} - EW$, it will move the UET (the single token it left in u) to the middle of the node next to the North of u on the *Base* ring and, explore this *east-west* ring. This agent continues exploring the *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 6. 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*" sketched out earlier. Eventually the algorithm terminates when there is only one node left unexplored in the last RUE. The only node left unexplored is the *BH*.

Lemma 6. *Eventually all but one* east-west *rings are explored.*

After a_1 finishes exploring all but one *east-west* rings, it will go and help a_2 to explore the ring a_2 currently explores. When one agent finishes exploring a ring (e.g. a *north-south* ring), it will know the number of nodes x of this ring. It can calculate the number of nodes y in an *east-west* ring, given n is known. If a *north-south* is the *Base* ring, we say that an agent finishes a *stage* once it finishes exploring a non-*Base* ring.

An agent a_1 will not visit an RUE until it has finished $y-1$ stages. Also, a_1 follows the path that a_2 took on the RUE until it sees the $CWWT$ token of a_2. Now a_1 and a_2 will execute procedure "*Bypass*". 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*.

The meaning of token(s) at different locations can be found in Table 1.

Token(s) position	Meaning
One token in the middle of a node	the *east-west* ring starts from this node is under exploration (the *north-south* ring is the *Base* ring)
Two tokens in the middle of the $\mathcal{H}$	the *Base* ring is found
One token on a port Two tokens on the *north* port of the $\mathcal{H}$	an agent is exploring the next node in this ring ($CWWT$ token) the first agent is exploring the *north-south* ring

Table 1: Token positions and their explanation in Algorithm *Cross Rings*

4.2.2. Correctness and Complexity Analysis

According to Observation 1 and Lemma 6, we can obtain the following Lemma:

Lemma 7. *At least one agent will find a* Base *ring in the torus.*

If we assume there are y nodes on an *east-west* ring, and $x = n/y$ nodes on a *north-south* ring, then:

Lemma 8. $x - 1$ east-west *rings will be explored eventually.*

Proof. According to Lemma 6, all but one *east-west* rings will be explored eventually. Given there are x nodes on each *north-south* ring, $x - 1$ *east-west* rings will be explored eventually.

Lemma 9. *The* UET *advances on the* north-south *ring correctly.*

Lemma 10. *None of the* east-west *rings will be explored more than once.*

Lemma 11. *At most* 1 *agent dies in the* BH. *And within finite time at least* 1 *agent will determine the location of the* BH.

Proof. From Lemmas 7 and 8 we know that two agents will sooner or later find a *Base* ring. Then both agents keep exploring the *east-west* rings along the *north-south* ring (the *Base* ring) until eventually one agent explores $x - 1$ *east-west* ring. According to Lemma 10, there are at most $x - 1$ such explorations before the two agents start to *bypass* each other on the last *dangerous east-west* ring left using a *safe east-west* ring. We also observe that from Lemma 11 at most one agent dies in the *BH*. Eventually the surviving agent will stop the algorithm when it explored and *bypassed* $y - 1$ nodes on the last *dangerous east-west* ring.

Lemma 12. *Two co-located agents with* 5 *tokens in total suffice to locate the* BH *in a labeled torus.*

Proof. According to Lemma 11, 2 agents suffice to locate the *BH*. We now prove that a total of 5 tokens is sufficient for both agents to locate the *BH*.

- When the algorithm starts, two tokens are needed for the agent that wakes up first in order to determine what is the *Base* ring. But once the second agent sees the *Base* ring is known, it can pick up the 2 tokens and eventually reuse them.

- Each agent needs one token to do *CWWT* for exploring and for *bypass*ing (including the "Back to the *Dangerous* Ring" procedure). As we just mentioned in the previous bullet, the second agent can reuse the 2 tokens in the middle of their $\mathcal{H}$ once the *Base* ring is decided.

- As we explained in Lemma 9, before one agent picks up the *UET* (one token) in the *Base* ring from node u, it goes to the next node v to the *north* to put a second *UET*. Only after putting the second *UET*, does the agent go back to node u to pick up the first *UET*. Then it starts exploring the *east-west* ring from node v. We observe that: a) an agent can use the *CWWT* token to put this second *UET* in node v and b) an agent can reuse the picked up *UET* in u to continue the exploration with *CWWT*. Hence, only one extra token is used as a *UET*.

Lemma 13. *$O(n)$ moves is sufficient using algorithm Cross Rings.*

Proof. In procedure "Find a *Base* ring", a *Base* ring is established after agent a_1 explored the *north-south* ring that includes the $\mathcal{H}$ or a_2 explored the *east-west* ring that includes the $\mathcal{H}$. If the number of nodes on a *north-south* ring is x, the number of nodes on a *east-west* ring is y, then $x * y = n$. So, $(x + y) \leq n$ moves will be executed. Hence, $O(n)$ moves suffice.

In procedure "*Bypass*" in a torus, an agent a_1 walks from the dangerous ring, through 1 link connecting to the *north* port, to the *safe* ring. a_1 is going to take y_1 steps before it executes procedure "Back to the *Dangerous* Ring". Then, a_1 executes y_2 steps before it walks back to the *dangerous* ring through 1 link. $y_1 + y_2 \leq y$ such links are going to be traversed given there is a maximum of y nodes on each *east-west* ring. So, it takes an agent $O(n)$ moves, even when using *CWWT*. Hence, on both a *safe east-west* ring and the *dangerous east-west* ring on its *north*, n links in total are going to be traversed in order to finish traversing the whole *east-west* ring. So, $O(n)$ moves are required for an agent during procedures "*Bypass*" and "Back to the *Dangerous* Ring". Hence, $O(n)$ moves in total are sufficient for two agents to locate the *BH*.

According to the lemmas listed above, and following the lower bound from the whiteboard model presented in [20], we can conclude:

Theorem 4. *Using* 2 co-located *agents and* 5 *tokens in total, the* BH *can be successfully located within* $\Theta(n)$ *moves in a labeled torus with* n *nodes.*

4.3. Bhs *in a Complete Network — algorithm Take Turn*

In this subsection, we present the solution to *Bhs* 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 port of the next node 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 searches 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, a 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*.

4.4. Correctness and Complexity

It is trivial to prove that any complete network has a subgraph that allows one node connection to all the other nodes in a complete network. Let $\mathcal{S}_n$ denote such a subgraph of complete graph $\mathcal{K}_n$. We will get:

Lemma 14. *Each link in $\mathcal{S}_n$ will be traversed only once.*

Lemma 14 and the fact that there is only one link in this subgraph that leads to a *BH*, clearly leads to Lemma 15.

Lemma 15. *There is at most one agent that dies in the* BH *using Algorithm Take Turn. The surviving agent will locate the* BH *correctly.*

Lemma 16. *Two agents can locate the* BH *within $\Theta(n)$ moves.*

According to the above Lemmas, and following the lower bound from the whiteboard model presented in [20], we conclude:

Theorem 5. *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. One token per agent for n agents suffice to correctly locate the *BH*. Hence we get the following theorem:

Theorem 6. *Using* n scattered *agents,* 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 — Algorithm Modified 'Cross Rings'

Again in this subsection, 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[1], and thus a maximum of two agents die after the first move. Hence, we conclude:

Lemma 17. *At least* 3 *scattered agents are needed to locate the* BH *in an* oriented *torus.*

The basic idea for solving *BH*s with *scattered* agents is to let two of the three agents form a pair that executes algorithm *Cross Rings* starting from the node where they formed this pair (i.e., their $\mathcal{H}$). We will now 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* agent. Consequently, we work out a way for the third agent to become *DONE* (i.e., stop working) 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.

Procedures "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" (see below) 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 finishes exploring the *north-south* ring, it then executes procedure "Single Agent Explores an *east-west* Ring".

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

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:

– 1 token on the *north* port of the pair's $\mathcal{H}$, then a_2 makes the *east-west* ring a *Base* ring by picking up the token on the *north* port of the pair's $\mathcal{H}$. a_2 then executes algorithm *Cross Rings* to the *east*.

– 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*; 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*.

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.

5.2.1. Correctness and Complexity Analysis

Lemma 18. *One pair will be formed within finite time.*

PROOF. According to the algorithm, as long as one single agent sees the tokens of another single agent, it will be able to modify the tokens immediately and become a paired agent consequently. If the other agent has already died in the *BH* and thus never comes back, we still say a pair is formed. Otherwise, the other agent will come back to pick up its *CWWT* token sooner or later. Eventually it will see the modified token configuration and, in turn, become a paired agent consequently. Now we only need to prove that at least one single agent will see the tokens of another single agent.

Assume there is no such single agent that will see the tokens of another single agent before the algorithm terminates. According to procedure "Paired Agent Explores a *north-south* Ring", once an agent wakes up, it is a single agent, and it will try to explore the *north-south* ring starting from the node in which it wakes up. If this single agent finishes exploring the *north-south* ring without dying in the *BH*; or seeing the token(s) of another agent (if it sees two tokens on a port then it forms a pair with that agent, if it sees one token on a port or one token in the middle of a node, then it becomes *passive*); or being eliminated by a paired agent (having its *CWWT* tokens stolen), it is going to explore all the *east-west* rings until it:

- either dies in the *BH*; or
- forms a pair with another single agent upon seeing the tokens of it; or
- becomes *Passive* upon seeing one token on a port or one token in the middle of a node or, noticing its *CWWT* tokens were stolen; or
- terminates the algorithm upon finishing exploring $n-1$ nodes ($n-2$ links).

We know that all three (minimum team size) agents execute the same algorithm and, all single agents walk with *CWWT*. According to the assumption: if no single agent sees the token of another single agent before the algorithm terminates, these agents must have died in the *BH*. We also know that if a single agent only leaves through the *north* and/or *east* ports of a node, it can only go into a *BH* through a link connecting to the *south* and/or *west* ports of the *BH*.

Consequently, one single agent will see the *CWWT* token of another single agent that died in the *BH* either in the node to the *west* of the *BH* or to the *south* of the *BH*. This contradicts the assumption we made at the beginning of this proof: "there is no such single agent that will see the tokens of another single agent before the algorithm terminates". So, the assumption is wrong. We therefore conclude that sooner or later at least one single agent will see the tokens of another single agent before the algorithm terminates. We also already proved that as long as one single agent sees the tokens of another single agent, they can form a pair correctly. Hence, eventually there will be a pair formed within finite time.

Lemma 19. *At least one agent will find a* Base *ring in the torus.*

Lemma 20. *A single agent will not interfere with the progress of any paired agent.*

PROOF. In procedures "Single Agent Explores a *north-south* ring" and "Single Agent Explores an *east-west* ring", as soon as a single agent sees one of the following token configurations, it will immediately become *Passive*:

- Case 1: there is only one token on a port.
- Case 2: as long as there is one token in the middle of a node.
- Case 3: its *CWWT* tokens were stolen. Namely, it no longer has 2 tokens on a port.

The above token configurations cover all the token configurations relevant to a pair agent.

Then, in all the procedures that a paired agent executes, we have added "eliminate single agent" steps. Such steps ensure that when either a paired agent encounters a single agent, or a single agent encounters a paired agent, the single agent will become *Passive* eventually. Hence, a single agent will not interfere with the progress of any paired agent.

Lemma 21. Bypass *can be correctly executed by a pair of agents.*

Proof. The only modification we do to procedure "*Bypass* in Torus" in algorithm *Cross Rings* is that we add one token in the middle of a node when one agent notifies the other agent to *bypass*. This extra token is used to eliminate the single agent (once the single agent sees a token in the middle, it will become *Passive* immediately), and it is used every time one agent *bypasses* another agent.

As we mentioned above, a paired agent uses one token on the port to continue its *CWWT*. When a single agent sees a token on a port, it immediately becomes *Passive*. When a paired agent sees there is only one token in a node and it is in the middle of this node, it will pick up the token before it continues exploring the next node. Hence, there is no possibility of having a token in the middle and a token on the port of a node, except for a paired agent trying to *bypass* another paired agent.

Lemma 22. *At most* 2 *agents die in the* BH*.*

Proof. Assume a single agent died in the *BH* first, and had its *CWWT* token left in the neighbor node of the *BH*. Normally no agent can go through a port with *CWWT* token(s), according to the *CWWT* rules defined in section 3. But according to algorithm *Modified 'Cross Rings'*, when a paired agent encounters the 2 *CWWT* tokens that a single agent left, it will pick them up and continue executing the algorithm. If this happens, this paired agent will leave its *CWWT* token on the port where the *CWWT* tokens of that single agent were picked up, and this paired agent will die in the *BH*. According to Lemma 11 in algorithm *Cross Rings* at most 1 agent dies in the *BH* when there are two co-located agents. Hence, at most one of the two paired agent will die in the *BH*, and thus at most 2 agents die in the *BH* during Algorithm *Modified 'Cross Rings'*.

Lemma 23. *Within finite time* 1 *agent will determine the* BH *location.*

Proof. As shown in Lemma 19, a pair will be formed within finite time. After a pair is formed, its agents execute the procedures in algorithm *Cross Rings* with a little modification, namely: eliminate the single agents. It is obvious that this step takes $O(1)$ time. And recall that we proved in Lemma 11 that within finite time one agent will determine the location of the *BH* using algorithm *Cross Rings*. Hence, within finite time, an agent will determine the location of the *BH* using algorithm *Modified 'Cross Rings'*.

Lemma 24. *Three agents with a maximum of seven (*7*) tokens in total are sufficient to locate the* BH *in a labeled torus with scattered agents.*

PROOF. According to Lemmas 22 and 23, 3 agents are sufficient to locate the *BH*. We now prove that a maximum of 7 tokens are sufficient in order for three agents to locate the *BH*. Before all, we must make a difference between the *CWWT* tokens of a single agent and those of a paired agent, because of the following two facts:

1. We have to have at least 3 agents in the torus when these are scattered in order to have one agent survive and eventually locate the *BH*.
2. Algorithm *Cross Rings* locates the *BH* correctly with 2 co-located agents.

We can either use 2 tokens on a port as the *CWWT* tokens of a single agent and 1 token on a port as the *CWWT* token of a paired agent, or vice versa. We arbitrarily decided to choose the first of these two alternatives for our algorithm. The tokens are used in the following situations:

- When the algorithm starts, two tokens are needed for a single agent to explore the *north-south* ring then all the *east-west* rings. Hence, $3 * 2 = 6$ tokens are required in total for three single agents.
- Two tokens are used to form a pair. The agent a_1 that initiates forming a pair will use the 2 *CWWT* tokens of the other single agent. The fact is that as soon as an agent becomes a paired agent, it only uses 1 token as its *CWWT* token. So, beyond the 1 token a_1 is going to use for *CWWT* as a paired agent, there will be one extra token that can be reused in other situations. When the other single agent a_2 comes back from its *CWWT*, it becomes a paired agent upon seeing the modified token configuration. One token (the *CWWT* token) is needed for a_2 to continue as a paired agent. Given the 2 *CWWT* tokens of a_2 are used for finding a *Base* ring, 1 additional token is required by a_2. It is also possible that by the time a_1 formed a pair with a_2, a_2 has already died in the *BH*. In this case, the extra token is not required.
- One token is used as a *UET*. This is because an agent can always use temporarily the *CWWT* token as the second *UET* (see Lemma 12). Given a_1 has an extra token and leaves it in the middle of the new $\mathcal{H}$, this extra token can be used as a *UET*.
- One token is used for the "*Bypass*" (including step "Back to the *Dangerous* Ring") procedure. This *bypass*ing is only going to happen once all but one *north-south*/*east-west* ring has been explored. So, there is no need for keeping the *UET*. The *UET* will be reused for *bypass*ing.

Hence we conclude: seven (7) tokens in total are sufficient to locate the *BH* in a labeled torus with 3 scattered agents.

Lemma 25. *$O(n)$ move suffice using algorithm Modified 'Cross Rings'.*

PROOF. We know from Lemma 18 that within finite time 1 pair will be formed before the algorithm terminates. In the worst case, each single agent traverses the whole torus before it either dies in the *BH* or forms a pair with another single agent. So, it takes at most $3n$ moves for an agent to traverse the torus using *CWWT*. For three single agents, it costs $3 * 3n$ moves in total. It is important to observe that none of the modifications we introduced to algorithm *Modified 'Cross Rings'* affect the number of moves. And we know that once an agent becomes a paired agent, $O(n)$ move suffice to locate the *BH*, according to Theorem 4 in algorithm *Cross Rings*. Hence, $O(n)$ moves in total is sufficient for the three agents to locate the *BH*.

According to the lemmas above, and following the lower bound from the whiteboard model presented in [28], we can conclude:

Theorem 7. *Using* 3 scattered *agents and* 7 *tokens in total, the* BH *can be successfully located using* $\Theta(n)$ *moves in a labeled torus with* n *nodes.*

5.3. In a Torus with k Scattered *Agents — Algorithm Single Forward*

In this section, we study *BH*s in a labeled torus with k ($k > 3$) *scattered* mobile agents. Here, k is not known to any of the agents. The number of nodes n in this torus and the dimension of the torus $x \times y$ are known to all the *scattered* agents. In this section, we do require that all the links and nodes obey the FIFO rule.

5.3.1. General Description

The agents are in three basic states: *single*, *forward* and *checking*. Each agent tries to explore the whole torus on its own. An (either *single* or *forward* or *checking*) agent always goes for an unexplored node reachable from its current location using only *north* and *east* links. An agent will never go through a *west* or *south* port unless it knows that port is *safe*. This (together with $CWWT$) ensures that at most two agents enter the *BH*. An agent is able to remember the number of nodes that it explored.

Every agent a_s wakes up as a *single* agent. It becomes a *forward* agent, only when it finds a token on the port that a_s intent to go through. In other words, when an agent arrives at a node, if further progress is blocked (i.e., at least one of the *unsafe north*/*east* ports is *blocked* (*with token*)), that port becomes a *Check Point* for agent a_s. If there is no other *unsafe north* or *east* port available (*without token*), *single* agent a_s remains as is and waits in the node after putting one token in the middle. a_s continues as a *single* agent if the port it wants to use becomes *without token*.

A *forward* agent a_f continues exploring the torus until it goes into a node u, with at least one token in the middle. We say this node is the second *Check Point* of this *forward* agent. a_f immediately becomes a *checking* agent a_c that checks the availability (at least one port is *without token*) of these two *Check Point*s. If both are unavailable then a_c chooses one *Check Point* to wait. When either of the two *Check Point* becomes without token, a_c continues as either a *forward* agent or a *single* agent. Eventually an agent that explored $n - 1$ nodes will terminate the algorithm and locate the *BH*. The whole algorithm can be summarized as follows:

1. *single* agent: has no *Check Point*, explores and becomes forward if blocked;
2. *forward* agent: has one *Check Point*, explores and becomes checker if blocked again;
3. *checking* agent: has two *Check Point*s, sits at one *Check Point* and upon any change checks the other *Check Point*. Becomes a *forward* agent when one of the *Check Point*s unblocks *without token*, a *single* agent if both of them become unblocked.

5.3.2. Correctness and Complexity Analysis

Lemma 26. *Within finite time at least one agent will survive and determine the location of the* BH.

Lemma 27. *One token per agent suffices using algorithm* Single Forward *to locate the* BH.

Lemma 28. *k (k > 3)* scattered *agents can locate the* BH *after executing* $O(k^2n^2)$ *moves using Algorithm Single Forward.*

PROOF. During the entire lifespan of an agent a_1, a_1 can explore no more than $n-1$ nodes. a_1 can also be blocked in a node by another agent (that may have died in the *BH*). Once a_1 is blocked, it will either continue its exploration or go back to the previous *Check Point* to check the availability of that node (i.e., whether it still has tokens in it). Each such check takes at most n moves. Only a change in the number of tokens can possibly trigger such a check. And only the entry or exit of a *single* or *forward* agent will trigger a change in the number of tokens, because no token is used for a checking agent to execute a check. In each node, there is a constant number of entry and exists preceeding the visit of a *single* or *forward* agent. So there are at most k such checks, because there are k agents in total. Hence, $O(k^2n^2)$ moves in total are executed by k agents.

Theorem 8. *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.*

6. Simulation Results

We present in this section the experimental results obtained from a series of Java simulations of algorithms Cross Rings, Modified Cross-Rings, and Single Forward. Each simulation is inspired by middleware platforms such as Agilla [37, 38] and implements a given population of mobile agents (from a population size of 2 up to 7). Mobile agents execute the algorithms within a network of interconnected nodes that follows a torus topology. The implementation consists of a simple discrete event, time-step based simulation engine, in which every agent executes the aforementioned algorithms at every step of simulated time. The simulation engine implements a discrete event scheduler, a graphical view, a data-collection system, and the simulated objects themselves, that is, network nodes and mobile agents. The simulation system consists of more than 4,600 lines of Java code. A sample simulation video is available online at `http://goo.gl/bJ3Ky/`.

For the sake of comparison, we simulated as well a random case, hereinafter denoted as Random Walk algorithm. An agent running the Random Walk case simply chooses the next port in a torus at random. Instead of using tokens, this algorithm is based on a whiteboard model: the agent keeps track of the map using the local memory allocated in each node. Every agent running the random walk algorithm also knows the total number of nodes in the system. Contrarily to algorithms Cross Rings, Modified Cross-Rings, and Single Forward, the Random Walk algorithm considers failures, i.e., the algorithm does not guarantee successful termination of a simulation with at least one agent alive and reporting the BH. Therefore, a simulation based on the random walk algorithm a) successfully terminates when at least an agent realizes that there is just one node that has not been visited (this node being the BH) or b) ends with a failure if all the agents die at the BH. Finally, given the random nature of the process, agents can visit the same node more than once.

Our simulation sets consist of 30 to 100-node networks. Agents start as co-located agents for the simulations associated with the Cross Rings algorithm and Random Walk with two agents, and as scattered agents for the remaining simulations. We consider that the execution of a simulation is successful if the BH is discovered by at least one surviving agent. Otherwise, the simulation is counted as a failure. For each successful simulation, we compute the percentage of moves that are necessary to discover the BH. All data is calculated from 100 independent successful runs of each setting with random initial agent and BH placement. Algorithms Cross Rings, Modified Cross Rings, and Single Forward guarantee that only 100 executions are necessary in order to obtain

100 independent successful runs. However, the random walk simulations require a higher number of runs to obtain the required data, given the possibility of failures. Table 2 summarizes the number of independent runs that were required during our experiments to obtain 100 successful tests in the random walk case. Figure 3 illustrates the average move results and the 95% confidence intervals obtained during the initial experimental series, in which populations of two co-located agents and populations of three scattered agents executed, respectively, algorithms Cross Ring, Random Walk and Modified Cross Ring. Results confirm that *O(n)* moves suffice to locate the BH in all simulations. Notice that those simulations executing algorithm Cross Rings with populations of two co-located agents and 5 tokens in total obtained the lowest number of moves; while simulations executing algorithm Modified Cross Rings with populations of three scattered agents and 7 tokens in total reported the highest number of moves. Simulations executing the Random Walk algorithm, with zero tokens but requiring the allocated memory of each node to map the system, provide intermediate results in terms of moves. However, they present a very high rate of failures (cf. Table 2) in comparison with algorithms Cross Rings and Modified Cross Rings — which guarantee the termination process with zero failures. Figure 4 compares the results obtained with the execution of the Modified Cross Ring algorithm with the average move results and the 95% confidence intervals obtained during the execution of algorithm Random Walk with populations ranging from 4 to 7 agents over the 30 to 100-node networks. Results confirm that *O(n)* moves suffice to locate the BH in all Random Walk simulations, but at the cost of *O(n)* memory on each node (i.e. Whiteboard model) and failures, as reported in Table 2. Finally, Figure 5 shows the same comparison with the last experimental series, where a population from 4 to 7 agents with a single token per agent execute algorithm Single Forward over the 30 to 100-node networks. Results confirm that $O(k^2n^2)$ moves suffice to locate the BH in all those simulations based on the execution of the algorithm Single Forward.

Discussion

Both theoretical analysis and simulation results show that, a *BH* can be located with $\Theta(n)$ moves in an oriented asynchronous torus using only 3 *scattered* agents. And such solution does not assume FIFO on the links and nodes. Contrary to our intuition that the moving cost maybe reduced by using more agents, we observe from the theoretical analysis that, when the number of *scattered* agents in a torus increases, communication between these agents becomes significantly more complicated. It consequently leads to a more expensive move cost, that is to $O(k^2n^2)$. Therefore, we did experiments to compare the move costs of algorithm Modified Cross Ring (using 3 scattered agents) and algorithm Single-Forward that uses 4, 5, 6 and 7 agents respectfully. The result is presented in Figure 5. This result confirms our observation that with more (than 3) agents the number of moves increases from liner to quadratic.

	# of nodes							
# of agents	*30*	*40*	*50*	*60*	*70*	*80*	*90*	*100*
2 (co-located)	861	1281	1267	1789	1697	2004	1846	2265
3 (scattered)	372	418	597	589	628	681	690	582
4 (scattered)	206	237	243	286	288	360	283	321
5 (scattered)	142	172	194	181	185	204	208	212
6 (scattered)	121	123	139	142	153	149	162	165
7 (scattered)	111	116	120	123	111	143	142	129

Table 2: Number of independent runs required to obtain 100 successful tests with algorithm Random Walk.

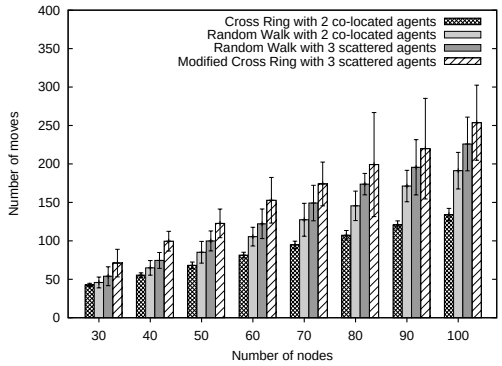

Figure 3: Cross Ring and Modified Cross Ring vs. Random Walk Algorithm with 2 and 3 agents

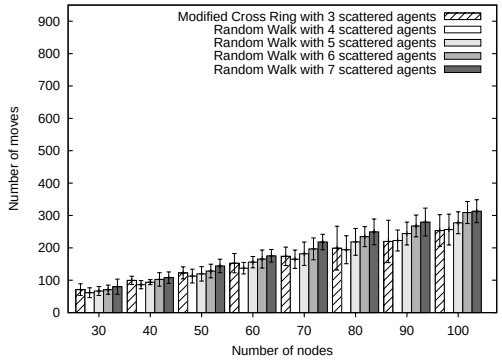

Figure 4: Modified Cross Ring with exactly three agents vs. Random Walk with more than 3 agents

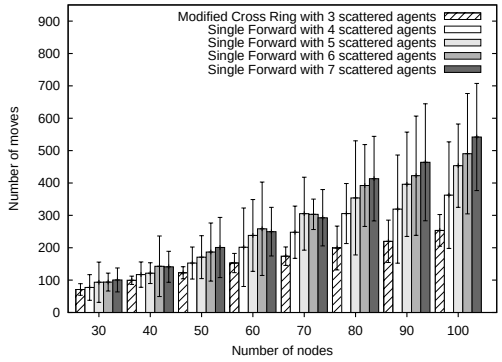

Figure 5: Modified Cross Ring with exactly three agents vs. Single Forward with more than 3 agents

7. Conclusion

In this paper, we have developed a set of token-based algorithms for locating a *BH* in three interconnected network topologies: hypercube, torus and complete network. We presented solutions with both *co-located* agents and *scattered* agents. In Table 3, we list the six algorithms discussed in this paper. We have shown that the complexity of the general algorithm for an arbitrary network can be considerably improved using instead an algorithm designed for a specific network topology. We compare the similarities and differences between the results obtained for these specific topologies and analyze the impact of topology and other performance factors on *Bhs*.

	Complete Network		**Hypercube**	**Torus**		
	Co-located Agents	*Scattered Agents*	*Co-located Agents*	*Co-located Agents*	*Scattered Agents*	
Algorithm Name	*Take Turn*	*Modified 'Take Turn'*	*Two Rings*	*Cross Rings*	*Modified 'Cross Rings'*	*Single Forward*
Orientation	No	No	No	No	Yes	Yes
FIFO	No	No	No	No	No	Yes
Team Size	2	n	2	2	3	3 or more
Token Cost	1 in total	1 per agent	1 per agent	5 in total	7 in total	1 per agent
Move Cost	$\theta(n)$	$O(n^2)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$O(k^2n^2)$

Table 3: Comparative Evaluation Table — Complete Network, Hypercube and Torus.

From this table and the conclusions drawn in [24, 28] we can make the following observations:

1. When we use co-located agents to solve *Bhs*:
   - minimum team size (2 agents) is achieved on all three topologies.
   - one more token is sufficient to eliminate all the extra agents when there are more than 2 (i.e., minimum team size) agents in the network.
   - the token cost is proportional to the connectivity for the three studied interconnected topologies, namely: Torus, Hypercube and Complete Network.
   - being the sparsest bi-connected graph and the one for which the cost (in terms of number of moves) for *Bhs* using whiteboards is the worst, the ring topology still has the highest cost (both for number of tokens and number of moves).
2. Within one topology, using scattered agents always requires more agents than using co-located agents.
3. For co-located agents, the cost of solving *Bhs* in the Ring topology is the worst.
4. Unlike when using co-located agents, for scattered agents, the cost of solving *Bhs* worsens as connectivity increases.

Clearly, our algorithms need to be further investigated in order to be improved towards the lower bound associated with each of their parameters. This should be feasible in the short term.

Moreover, in section 5, we demonstrate that *Bhs* with a team of *scattered* agents (a problem not addressed in whiteboard models) is rather complex but solvable in some dense graphs. We are now exploring an optimal solution for *Bhs* in a hypercube using *scattered* agents.

Finally, in this paper we have considered *Bhs* with only one *BH*. We would like to study the problem with multiple black holes. But, clearly, in such a case, agents may be isolated into segments

of the topology unreachable from the majority of the nodes. In other words, the solvability of *Bhs* in the case of multiple black holes appears to depend directly on connectivity in a specific network. We believe this complex problem must be tackled once, and only once we have carried out the previously suggested future work.

References

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

[2] D. B. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents", *Communication. ACM.*(1999) **42(3)**: 88-89.

[3] R. S. Gray and G. Cybenko and D. Kotz and R. A. Peterson and D. Rus, "D'Agents: Applications and performance of a mobile-agent system", *Software - Practice and Experience.* (2002) **32(6)**: 543-573.

[4] J.Z., Hernandez and S., Ossowski and A., Garcia-Serrano, "Multiagent architectures for intelligent traffic management systems", *Transportation Research Part C.*(2002) **10(56)**: 473506.

[5] F. Logi and S. G. Ritchie, "A multi-agent architecture for cooperative inter-jurisdictional traffic congestion management", *Transportation Research Part C* (2002) **10(56)**: 507527.

[6] J. L. Adler and G. Satapathy and V. Manikonda and B. Bowles and V. J. Blue, "A multi-agent approach to cooperative traffic management and route guidance", *Transportation Research Part B* (2005) **39(4)**: 297318.

[7] B. Chena and H. H. Chengb and J. Palen, "Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems", *Transportation Research Part C: Emerging Technologies.* (2009) **17(1)**:110.

[8] C. Bel and W. van Stokkum, "A model for distributed multi-agent traffic control", *Multiple Approaches to Intelligent Systems*, **Proceedings 1611**, 1999, pp: 480489.

[9] J. Blum and A. Eskandarian, "Enhancing intelligent agent collaboration for flow optimization of railroad traffic", *Transportation Research Part A* (2002) **36(10)**: 919930.

[10] H. Proenca and E. Oliveira, "MARCS: Multi-agent railway control system", *Advances in Artificial Intelligence* (2004) **3315**: 1221.

[11] D.M. Chess, "Security issues in mobile code systems", *Proc. of* 1998 *Conf. on Mobile Agent Security (MAS'98)*, (1998) **LNCS 1419**: 1-14.

[12] W. Glover and J. Lygeros, "A stochastic hybrid model for air traffic control simulation", *Hybrid Systems, Computation and Control*, **Proceedings 2993**: 372386.

[13] W. G. Li and M.V. Pinheiro, "Method to balance the communication among multi-agents in real time traffic synchronization", *Fuzzy Systems and Knowledge Discovery, Part 1*, **Proceedings 3613**: 10531062.

[14] C. Braz, "Mobile Agents for Wireless E-Commerce Applications", *Master Thesis.* (February 2003), Universit de Montral.

[15] J. Herbert and J. O'Donoghue and G. Ling and K. Fei and C.L. Fok, "Mobile agent architecture integration for a wireless sensor medical application", *Proc. of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*, 2006, pp. 235–238.

[16] E. Mercadal and S. Robles and R. Martí and C. Sreenan and J. Borrell, "Heterogeneous Multiagent Architecture for Dynamic Triage of Victims in Emergency Scenarios", *Advances on Practical Applications of Agents and Multiagent Systems*, 2011, pp. 237–246.

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

[18] C. Cooper and R. Klasing and T. Radzik, "Searching for black-hole faults in a network using multiple agents", in LNCS 4305 (M. Momenzadeh and A. Shvartsman eds.), *Proc. of* 10^{th} *Int. Conf. on Principles of Distributed Systems (OPODIS'06)*, 2006, pp. 320–332.

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

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

[21] S. Dobrev and P. Flocchini, R. Kralovic and N. Santoro, "Exploring an unknown dangerous graph using tokens", *Theoretical Computer Science*, Volume 472, February 2013, pp. 28-45.

[22] S. Dobrev and P. Flocchini and G. Prencipe and N. Santoro, "Searching for a black hole in arbitrary networks: Optimal mobile agent protocols," *Distributed Computing.* (2006) **19(1)**: 1–9.

[23] S. Dobrev and P. Flocchini and G. Prencipe and N. Santoro, "Mobile search for a black hole in an anonymous ring," *Algorithmica.*(2007) **48(1)**: 67–90.

[24] S. Dobrev, N. Santoro and W. Shi, "Scattered Mobile Agents Searching for a Black Hole in an Unoriented Ring Using Tokens," *International Journal of Foundations of Computer Science(IJFCS).* (2008) **19(6)**: 1355-1372.

[25] S. Dobrev and N. Santoro and W. Shi, "Scattered black hole search in an oriented ring using tokens" in *Proc. of* 9^{th} *Workshop on Advances in Parallel and Distributed Computational Models (APDCM'07)*, 2007, IEEE International Volume , Issue: 26-30 pp. 1–8.

[26] R. Klasing and E. Markou and T. Radzik and F. Sarracco, "Hardness and approximation results for black hole search in arbitrary networks", *Theoretical Computer Science.* (2007) **384(2-3)**: 201–221.

[27] W. Shi, "Black Hole Search with Tokens in Interconnected Networks", *Proc. of* 11^{th} *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'09).* 2009, pp. to 670-682.

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

[29] P. Fraigniaud and D. Ilcinkas, "Digraph exploration with little memory", in LNCS 2996 (V. Diekert and M. Habib eds.), *Proc. of* 21^{st} *Symp. on Theoretical Aspects of Computer Science (STACS'04)*, 2004, pp. 246–257.

[30] T.A. El-Ghazawi, "Characteristics of the MasPar parallel I/O system", *Proc. of the* 5^th *Symposium on the Frontiers of Massively Parallel Computation (Frontiers'95)*. 1995, pp. 265–.

[31] P. Flocchini and D. Ilcinkas and N. Santoro,"Ping Pong in Dangerous Graphs: Optimal Black Hole Search with Pebbles", *Algorithmica.* (11 February 2011), pp. 1-28

[32] J. Chalopin and S. Das and A. Labourel and E. Markou, "Black Hole Search with Finite Automata Scattered in a Synchronous Torus" in *Proc. of* 25^th *International Symposium on Distributed Computing (DISC'11)*, 2011, pp. 432-446.

[33] J. Chalopin and S. Das and A. Labourel and E. Markou, "Tight Bounds for Scattered Black Hole Search in a Ring" in *Proc. of* 18^th *International Colloquium on Structural Information and Communication Complexity (SIROCCO'11)*, 2011, pp. 186-197.

[34] L. Barrire, P. Flocchini, P. Fraigniaud, and N. Santoro, "Rendezvous and Election of Mobile Agents: Impact of Sense of Direction", *Theory Computing Systems* (2007) **40(2)**: 143-162.

[35] P. Flocchini and B. Mans, "Optimal election in labeled hypercubes", *Journal of Parallel and Distributed Computing.* (1996) **33(1)**: 76-83.

[36] T. Leighton, "Introduction to parallel algorithms and architectures: arrays, trees, hypercubes", *M.I.T. Press* (1992).

[37] C.L. Fok, G.C. Roman and C. Lu. (2009). Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 4(3): 26 pages.

[38] Agilla: A Mobile Agent Middleware for Wireless Sensor Networks. http://mobilab.cse.wustl.edu/projects/agilla/