

USING MOBILE AGENTS FOR BLACK HOLE SEARCH
WITH TOKENS IN MULTI NETWORKS

by
Wei Shi

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario

May, 2007

© Copyright by Wei Shi, 2007

Abstract

A Black Hole is a highly harmful host that disposes of visiting agents upon their arrival without leaving any observable trace of such destruction. In this dissertation, we study the Black Hole search problem using mobile agents in four topologies: ring, hypercube, torus and complete network. We do so without relying on local storage. Instead we use a less-demanding and less-expensive *token* mechanism.

In the first part of this dissertation, we study in depth the black hole search problem in an anonymous ring network. We prove that with co-located agents, the problem can be solved with a minimal of two co-located agents, three tokens in total performing in $\Theta(n \log n)$ moves. With scattered agents, we prove that, in oriented rings, the number of moves can be reduced from $O(n^2)$ to the optimal $\Theta(n \log n)$ using only $O(1)$ tokens per agent, without any knowledge of the team size. Interestingly, the proposed algorithm also solves, with the same cost, the *Leader Election* problem and the *Rendezvous* problem for the scattered agents despite the presence of a Black Hole. Then we prove that, even if the ring is un-oriented, locating the Black Hole is feasible with a minimum of three (3) scattered agents. With a team of four (4) or more scattered agents, $O(1)$ tokens per agent, a Black Hole can be located with $\Theta(n \log n)$ moves.

We study the Black Hole Search problem also for three other topologies. For the co-located agents, we show that the Black Hole can be located with minimum of 2 agents performing $\Theta(n)$ moves with $O(1)$ tokens in each of these three topologies. We present solutions for the Torus and Complete Network with scattered agents, knowing that the Black Hole Search problem was never studied with scattered agents in any of these three network topologies, neither using the whiteboard model, nor with the token model.

For these four topologies, in both the co-located and scattered agents cases, we obtain solutions with minimum number of agents and demonstrate that the number of tokens can be reduced to a constant number even if both the mobile agents and the network are *anonymous*.

Acknowledgements

I would like to give my sincere thanks to my two co-supervisors Dr. Nicola Santoro and Dr. Stefan Dobrev for their time, patience and constant help throughout the development of the research. I also want to thank them for their generous financial support, without which, as an international student, my studies at Carleton would not have been possible.

I also want to thank all my friends for their friendship and support.

Most importantly, this work would not have been possible without the love and constant encouragement from my parents Yongming Shi and Huimin Liu.

Finally, no words can express the gratitude and ultimately the love I feel for the one who weathered the storms and supported me so much.

Table of Contents

Abstract	ii
Acknowledgements	iii
Table of Contents	vii
List of Figures	x
List of Tables	xi
List of Algorithms	xiv
Chapter 1 Introduction	1
1.1 Introduction and Thesis Statement	1
1.2 Major Contributions	5
1.3 Thesis Organization	9
Chapter 2 The Framework, Literature Review and Research Method	11
2.1 Model and Framework	11
2.1.1 Overall Framework	11
2.1.2 Four Network Topologies	14
2.2 Overall Literature Review	15
2.2.1 Existing Research on the Hypercube	17
2.2.2 Existing Research on the Torus	18
2.2.3 Existing Research on the Complete Network	19
2.3 Research Method	20

Chapter 3	Black Hole Search in Rings	24
3.1	Basic Tool and Observations	25
3.1.1	Basic Tool — Cautious Walk with Tokens (<i>CWWT</i>)	25
3.1.2	Observations	27
3.2	BH Search by Co-located Agents	28
3.2.1	Elimination Technique	28
3.2.2	General Description and Basic Ideas	28
3.2.3	Algorithm <i>Divide with Token</i>	32
3.2.4	Algorithm <i>Divide with Token +</i>	33
3.2.5	Algorithm <i>Divide with Token -</i>	43
3.3	BH Search in an Oriented Ring by Scattered Agents	52
3.3.1	Basic Observation	52
3.3.2	Algorithm <i>Gather Divide</i>	52
3.3.3	Algorithm <i>Pair Elimination</i>	55
3.4	BH Search in an Unoriented Ring by Scattered Agents	75
3.4.1	Introduction	75
3.4.2	Algorithm <i>Shadow Check without CWWT</i>	76
3.4.3	Algorithm <i>Modified Shadow Check without CWWT</i>	90
3.4.4	Algorithm <i>Shadow Check</i>	94
3.5	BHS in an Anonymous Ring: Summary	99
Chapter 4	Black Hole Search in Hypercubes	103
4.1	Topological Characteristics	103
4.1.1	The Hypercube and Its Labeling	103
4.1.2	Gray Code	106
4.1.3	Hamiltonian Cycle	107
4.2	Basic Technique and Observations	108

4.3	Model and Assumptions	108
4.4	Basic Ideas	111
4.5	Co-located Agents Case — Algorithm <i>Two Rings</i>	116
4.5.1	General Description	116
4.5.2	<i>Bypass</i> Technique	117
4.5.3	Procedure “Initialization” and “Find a <i>safe</i> ring”	120
4.5.4	Procedure “ <i>Bypass</i> ”	123
4.5.5	Correctness and Complexity Analysis	124
Chapter 5	Black Hole Search in Tori	129
5.1	Topological Characteristics	129
5.2	Algorithm “Cross Rings” — The Case of Co-located Agents	130
5.2.1	Assumption, Basic Ideas and General Description	130
5.2.2	Procedure “Initialization” and “Find a <i>Base</i> Ring”	135
5.2.3	Procedure “Explore the <i>east-west/north-south</i> Rings”	140
5.2.4	Procedure <i>Bypass</i> on Torus	141
5.2.5	Correctness and Complexity Analysis	146
5.3	Algorithm <i>Modified ‘Cross Rings’</i> — The Case of 3 Scattered Agents	157
5.3.1	Assumptions, Basic Ideas, Observation and General Description	157
5.3.2	Procedures “Initialization” and “Single Agent Explores a <i>north-south</i> Ring”	160
5.3.3	Procedure “Single Agent Explores an <i>east-west</i> Ring”	163
5.3.4	Procedure “Paired Agent Finds a <i>Base</i> Ring”	166
5.3.5	The Rest of the Algorithm	171
5.3.6	Correctness and Complexity Analysis	174
5.4	Algorithm <i>Single Forward</i> — The Case of k Scattered Agents	181
5.4.1	Assumptions	181

5.4.2	General Description	182
5.4.3	Procedures “Initialization” and “Single Agent”	183
5.4.4	Procedures “Forward Agent” and “Checking Agent”	185
5.4.5	Correctness and Complexity Analysis	186
Chapter 6	Black Hole Search in Complete Networks	190
6.1	Topological Characteristics	190
6.2	Assumption, Basic Technique and Observations	191
6.3	BHS in a Complete Network with Co-located Agents	192
6.3.1	Algorithm <i>Take Turn</i>	194
6.3.2	Correctness and Complexity	194
6.4	BHS in a Complete Network with Scattered Agents	196
6.4.1	Some Basic Observations, Assumptions and Conclusions	197
Chapter 7	Conclusion	199
7.1	Recapitulation	199
7.2	Comparative Evaluation	203
7.3	Future Work	206
	Bibliography	210

List of Figures

Figure 1.1	Simplified summary table of the algorithms in this dissertation.	6
Figure 2.1	A Torus	22
Figure 2.2	A 5-Hypercube	22
Figure 2.3	A complete graph with n ($n \geq 3$) nodes is a full connected Chordal Ring	22
Figure 3.1	Pair levels table	57
Figure 3.2	The BPs of three consecutive pairs that reached level $i - 1$. . .	68
Figure 3.3	Two <i>crowned pairs</i>	70
Figure 3.4	Flow Chart for Algorithm <i>Shadow Check</i>	94
Figure 4.1	Hypercube Interconnection Strategy.	103
Figure 4.2	Hypercube Processor Numbering Scheme	104
Figure 4.3	Sub-tree in a Hypercube of 8 Nodes	105
Figure 4.4	The first several steps in constructing a k -bit Gray code	106
Figure 4.5	Hamiltonian cycles and connecting links in a 2-hypercube and a 3-hypercube	110
Figure 4.6	A 4-hypercube with Hamiltonian cycle highlighted in bold and eight ($2^{4-1} = 8$) dimension 4 connecting links marked with red cross.	110
Figure 4.7	A 4-hypercube consisting of two 3-hypercubes (in purple dashed lines) with eight ($8 = 2^{4-1}$) 4-dimension connecting links marked with red cross.	111

Figure 4.8	A Hamiltonian cycle constructed according to the permutation we proposed in a 2 dimensional hypercube	114
Figure 4.9	Merging Hamiltonian cycles of two i -hypercubes into a Hamiltonian cycle of $(i + 1)$ -hypercube. Here $k = 2^i$	115
Figure 4.10	Bypass Technique — Steps 1, 2, 3 and 4	119
Figure 4.11	Bypass Technique — Steps 5, 6, 7 and 8	119
Figure 4.12	Bypass Technique — Steps 9, 10 and 11	120
Figure 5.1	A simple Torus	129
Figure 5.2	All the nodes of a $3 * 4$ Torus connected by minimum number of rings.	131
Figure 5.3	Two paths that allows an agent to travers all the nodes in a labeled $3 * 4$ torus	132
Figure 5.4	Bypass Technique on Torus — Steps 1, 2, 3 and 4	144
Figure 5.5	<i>Bypass</i> Technique on Torus — Steps 5, 6, 7 and 8	144
Figure 5.6	<i>Bypass</i> Technique on Torus — Steps 9, 10 and 11	145
Figure 5.7	The case of one agent involved in moving the <i>UET</i>	148
Figure 5.8	a_f puts the second <i>UET</i> before a_s does.	149
Figure 5.9	The first two scenarios when two agents are both ready to move the <i>UET</i>	151
Figure 5.10	The last two scenarios when two agents are both ready to move the <i>UET</i>	151
Figure 5.11	One example of how two agents can handle the <i>UET</i> correctly.	152
Figure 5.12	The two ways for an agent to go from node u to node v	156
Figure 5.13	Token positions and their meanings.	159

Figure 5.14 Token configurations and their resulting actions in procedure
“Single Agent Explores a *north-south* Ring”. 161

Figure 5.15 Token configurations and their resulting actions in procedure
“Single Agent Explores an *east-west* Ring”. 165

Figure 5.16 The comparison table of procedures “Find a *Base* Ring” in
algorithm *Cross Rings* and procedures “Paired Agent Finds a
Base Ring” in algorithm *Modified ‘Cross Rings’*. 169

Figure 6.1 Left: The simplest way for an agent to traverse a unlabeled
complete network. Right: $n - 1$ agents wander into the BH
once they wake up. 190

Figure 6.2 The Star Shape Complete Network of 24 nodes with the \mathcal{HB} of
a pair of co-located agents in the middle 193

Figure 6.3 The Star Shape Complete Network with the \mathcal{HB} of a pair of
co-located agents in the middle 195

Figure 7.1 Comparative evaluation table — Ring. 204

Figure 7.2 Comparative evaluation table — Complete Network, Hyper-
cube and Torus. 205

List of Tables

Table 3.1	Summary of BHS in an Anonymous Ring with Co-located Agents	100
Table 3.2	Summary of BHS in an Anonymous Ring with Scattered Agents	101
Table 4.1	Token positions and their explanation — 1	117
Table 4.2	Token positions and their explanation — 2	117
Table 5.1	Keywords equivalence table	133
Table 5.2	Token positions and their explanation in Algorithm <i>Cross Rings</i>	135

List of Algorithms

1	Algorithm <i>Divide with Token +</i> — Procedure “Initialization” and “Exploring”	35
2	Algorithm <i>Divide with Token +</i> — Procedure “Seeking” and “Check & Split”	36
3	Algorithm <i>Divide with Token +</i> — Procedure “Checking”	37
4	Algorithm <i>Divide with Token -</i> — Procedure “Initialization” and “Explore”	47
5	Algorithm <i>Divide with Token -</i> — Procedure “Checking”, “Seeking” and “Check & Split”	48
6	Algorithm <i>Gather Divide</i>	54
7	Algorithm <i>Pair Elimination</i> — Procedure “Initialization” and “Form Pair”	61
8	Algorithm <i>Pair Elimination</i> — Procedure “Right Exploring”	62
9	Algorithm <i>Pair Elimination</i> — Procedure “Left Exploring”	64
10	Algorithm <i>Pair Elimination</i> — Procedure “Checking”	65
11	Algorithm <i>Pair Elimination</i> — Procedure “Seeking”	66
12	Algorithm <i>Shadow Check without CWWT</i> — Procedure “Initialization” and “Junior Explorer”	82
13	Algorithm <i>Shadow Check without CWWT</i> — Procedure “Junior Explorer” — Cases 1 and 2	83
14	Algorithm <i>Shadow Check without CWWT</i> — Procedure “Junior Explorer” — Cases 3 and 4	83

15	Algorithm <i>Shadow Check without CWWT</i> — Procedure “Junior Explorer” — Cases 5 and 6	84
16	Algorithm <i>Shadow Check without CWWT</i> — Procedure “Checker”	84
17	Algorithm <i>Shadow Check without CWWT</i> — Procedure “Senior Explorer” — right agents	85
18	Algorithm <i>Modified Shadow Check without CWWT</i> — Procedure “Checker”	92
19	Algorithm <i>Shadow Check</i> — Procedure “Initialization”	94
20	Algorithm <i>Shadow Check</i> — Procedure “Junior Explorer”	95
21	Algorithm <i>Shadow Check</i> — Procedure “Senior Explorer”	96
22	Algorithm <i>Shadow Check</i> — Procedure “Check”	96
23	Algorithm <i>Two Rings</i> — Procedure “Initialization”	121
24	Algorithm <i>Two Rings</i> — Procedure “Explore on a Ring”	122
25	Algorithm <i>Two Rings</i> — Procedure “ <i>Bypass</i> ”	125
26	Algorithm <i>Cross Rings</i> — Procedure “Initialization”	137
27	Algorithm <i>Cross Rings</i> — Procedure “Find a <i>Base</i> Ring”	138
28	Algorithm <i>Cross Rings</i> — Procedure “Advance in the <i>Base</i> Ring”	139
29	Algorithm <i>Cross Rings</i> — Procedure “Explore the <i>east-west</i> Rings”	140
30	Algorithm <i>Cross Rings</i> — Procedure “Explore the <i>north-south</i> Rings”	140
31	Algorithm <i>Cross Rings</i> — Procedure “ <i>Bypass</i> on Torus”	146
32	Algorithm <i>Cross Rings</i> — Procedure “Back to the <i>Dangerous</i> Ring — Torus”	147
33	Algorithm <i>Modified ‘Cross Rings’</i> — Procedure “Initialization” and “Single Agent Explores a <i>north-south</i> Ring”	164
34	Algorithm <i>Modified ‘Cross Rings’</i> — Procedure “Single Agent Explores an <i>east-west</i> Ring”	167

35	Algorithm <i>Modified ‘Cross Rings’</i> — Procedure “Paired Agent Finds a <i>Base Ring</i> ”	172
36	Algorithm <i>Modified ‘Cross Rings’</i> — Extra Lines	173
37	Algorithm <i>Modified ‘Cross Rings’</i> — Another Extra Line	173
38	Algorithm <i>Single Forward</i> — Procedure “Initialization” and “Single Agent”	184
39	Algorithm <i>Single Forward</i> — Procedure “Forward Agent”	186
40	Algorithm <i>Single Forward</i> — Procedure “Checking Agent”	187
41	Algorithm <i>Take Turn</i> — Procedure “Initialization” and “Take Turn”	194

Chapter 1

Introduction

1.1 Introduction and Thesis Statement

Whereas exploration problems by mobile agents have been extensively studied in the context of safe networks, the reality of networked systems supporting mobile agents is that these systems are highly unsafe. Indeed, the most pressing concerns all pertain to 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)[24, 68, 74, 75, 76, 93, 99, 109].

Computational and algorithmic research has just recently started to consider these issues. The computational issues related to the presence of a harmful agent have been explored in the context of “intruder capture” and “network decontamination”. In the case of a harmful host the focus has been on the notion of a Black Hole (BH), that is, of a node that disposes of any incoming agent without leaving any observable trace of this destruction [27, 30, 42, 44, 45, 46, 48, 49, 79]. The goal of all these investigations is to determine the fundamental properties, the computational limits, and the inherent complexity of the Black Hole Search (BHS) problem. In this dissertation, we continue the investigation of the BHS problem.

As mentioned earlier, a BH is a network site that disposes of any incoming agent without leaving any observable trace of this destruction. It corresponds to, for example, a node where a resident process (e.g., an unknowingly installed virus) deletes visiting agents or incoming data. Furthermore, any undetectable crash failure of a site

in an asynchronous network transforms that site into a BH. In presence of a BH, the first important goal is to determine its location. To this end, a team of mobile system agents is deployed. Their task is completed if, within finite time, at least one agent survives and knows the links leading to the BH. The research goal is to determine under what conditions and at what cost mobile agents can successfully accomplish this task, called the Black Hole Search (BHS) problem. The main complexity parameter is the size of the team; i.e., the number of agents used in the search. Another important complexity measure is the amount of moves performed by the agents in their search.

Both solvability and complexity of BHS depend on a variety of factors; first and foremost on whether the system is asynchronous [42, 44, 45, 46, 48] or synchronous [27, 30, 31, 79]. Indeed the nature of the problem changes drastically and dramatically in each case. For example, in both synchronous and asynchronous systems, with enough agents it is possible to locate the BH if we are aware of its existence. However, if there is doubt on whether or not there is a BH in the system, in absence of synchrony this doubt cannot be removed. In fact, in an asynchronous system, it is undecidable to determine if there is a BH [45]. The consequences of this fact are numerous and render the asynchronous case considerably difficult. In this dissertation we continue the investigation of the asynchronous case.

Other important factors influencing solvability and complexity are the amount of a priori knowledge held by the agents (e.g., number of nodes, map of network, etc.) and the means offered by the system for agent communication and coordination (e.g., whiteboard, blackboard, reliable message passing, etc). In particular, with the exception of [44], all existing investigations on BHS in asynchronous systems have assumed the presence of a powerful inter-agent communication mechanism, whiteboard, at all nodes. In the whiteboard model, each node has available a local storage area (the

whiteboard) accessible in fair mutual exclusion to all incoming agents. Upon gaining access, the agent can write messages on the whiteboard and can read all previously written messages. This mechanism can be used by the agents to communicate and mark nodes or/and edges, and has been commonly employed in several mobile agents computing investigations (e.g. see [10, 24, 55, 58, 63, 64, 90, 92, 99]). Although many research questions are still open, the existing investigations have provided a strong characterization of the asynchronous BHS problem using whiteboards. The availability of whiteboards at all nodes is a requirement that is expensive to guarantee in practice and theoretically (perhaps) not necessary. This leads to the theoretically intriguing and practically important question of whether there are simpler and less expensive inter-communication and synchronization mechanisms that would still empower a team of agents to locate the BH.

In this dissertation, we consider the less powerful token model, often employed in the exploration of safe graphs. In this model, each agent has available a bounded number of tokens that can be carried, placed on or removed from the middle or a port of a node [44, 49, 52, 53]. All tokens are identical (i.e., indistinguishable) and no other form of communication or coordination is available. Some obvious questions immediately arise: is the BHS problem still solvable using this weaker mechanism, and if so under what conditions and at what cost? Notice that the use of tokens introduces another complexity measure: the number of tokens. Indeed, if the number of tokens is unbounded, clearly it is possible to simulate a whiteboard environment; hence the question immediately arises of how many tokens are really needed. These questions are clearly theoretically relevant, as the answers would shed light on the nature and limits of the problem and on the impact of communication mechanisms on computability and complexity. They also have a practical relevance. In fact, although existing mobile agent platforms provide whiteboards or even stronger means

for communication and coordination of mobile agents, such platforms are not available on most networked environments. Hence, a less intrusive mechanism such as tokens might be more feasible.

We ask all these questions and intend to provide definite answers for the ring, hypercube, torus and complete network. We start our research by investigating the ring topology, which is the sparsest bi-connected graph¹ and the one for which the cost (in terms of number of moves) for BH search with whiteboards is the worst. Then we study the three other topologies that have much stronger connectivity.

In all four network topologies studied in this dissertation, the problem of locating the BH using tokens is examined both in the case of *co-located* agents, that is when all the agents start from the same node and (with the exception of the hypercube) in the case of *scattered* agents, that is when the agents start from different nodes.

The problem becomes considerably more difficult if the agents are *scattered*, that is, when they start from many different nodes. In particular, with scattered agents, the presence (or lack) of orientation in the network topology and the team size are important factors. Here, an “oriented” network topology is taken to be one in which all the agents are able to agree on a common sense of direction. Conversely, an “unoriented” network topology means the agents may not be able to agree on a common sense of direction.

Finally, letting Δ denote the degree of a network topology, we observe the following fact: when a team of scattered agents are in an unoriented topology, even if the team size is Δ , it is possible that all the agents disappear in the BH after their first steps upon waking up. Hence, we observe:

Observation 1 *Δ agents are not enough to locate the BH in an unoriented network of degree Δ if the agents are scattered.*

¹Edge bi-connectivity is required for BHS in asynchronous systems [46]

1.2 Major Contributions

In this dissertation, we have considered the token model, which is less powerful and less expensive than the whiteboard model. Some obvious questions were raised: is the BHS problem still solvable using this weaker mechanism, and if so, under what conditions and at what cost? Also, notice that the use of tokens introduces another complexity measure: the number of tokens. Indeed, if the number of tokens is unbounded, clearly it is possible to simulate a whiteboard environment. Hence another question was immediately raised: how many tokens are really needed?

We asked all these questions and provided answers for the ring, hypercube, torus and complete network. We started our research by investigating the ring topology, which is the sparsest bi-connected graph and the one for which the cost (in terms of number of moves) for BH search with whiteboards is the worst. Then we studied the three other topologies, which have stronger connectivity.

In all four network topologies, which were studied in this dissertation, the problem of locating the BH using tokens has been examined in the case of *co-located* agents, that is when all the agents start from the same node and *scattered* agents, that is the agents do not start from the same node.

The results of this dissertation are summarized as follows (See Figure 1.1):

		Name	Agents k	Tokens	Orien- tation	k Known	Moves	
Ring	Co-located	DWT -	2+	2/agent	No	No	$\Theta(n \log n)$	*
	Scattered	Gather Divide	2+	1/agent	Yes	Yes	$O(kn + n \log n)$	
		Pair Elimination	2+	4/agent	Yes	No	$\Theta(n \log n)$	*
		MSCWC	4+	5/agent	No	No	$\Theta(n \log n)$	*
Torus	Co-located	Cross Rings	2+	5 total	No	No	$\Theta(n)$	
	Scattered	M 'Cross Rings'	3	7 total	Yes	No	$\Theta(n)$	
		Single Forward	3+	1/agent	Yes	No	$O(k^2 n^2)$	
Hyper- cube	Co-located	Two Rings	2+	1/agent	No	No	$\Theta(n)$	
Complete Network	Co-located	Take Turn	2+	1 total	No	No	$\Theta(n)$	
	Scattered	M'Take Turn'	n+	1/agent	No	No	$O(n^2)$	

Figure 1.1: Simplified summary table of the algorithms in this dissertation.

In the co-located agents case, BHS is indeed solvable [44, 49]. In particular, in [49] we show that a team of two or more co-located agents can solve BHS with $\Theta(n \log n)$ moves and two (2) tokens per agent in a ring network. Later, we develop simple algorithms to solve BHS problem for the three other network topologies. Without requiring the FIFO rule², we prove that:

- using two (2) co-located agents, $O(1)$ tokens in total and $\Theta(n)$ moves, the BH can be successfully located in a labeled hypercube.
- using two (2) co-located agents and $O(1)$ tokens in total, the BH can be successfully located with $\Theta(n)$ moves in a labeled torus.
- using two (2) co-located agents and $O(1)$ token in total, the BH can be successfully located in a complete network without sense of direction [60, 61, 62], with

²Throughout this dissertation, the FIFO rule refers to the fact that a node must process agents in their serialized FIFO order of arrival in this node. Consequently, because of this rule, an agent cannot overtake another on the same link.

$\Theta(n)$ moves in total.

In the case of scattered agents, the following results are obtained in the ring topology: in [52], we show that a team of two or more scattered agents can solve BHS with $\Theta(n \log n)$ moves and five (5) tokens per agent when the orientation is known. Furthermore, in [53] we show that a team of three (3) or more scattered agents can solve BHS with $O(n^2)$ moves and four (4) tokens per agent when the orientation is unknown. But given one more agent (4 scattered agents in total), BHS can be solved with $\Theta(n \log n)$ moves and four (4) tokens per agent when the orientation is still unknown. For the Torus and Complete Network topologies, we prove that:

- using three (3) scattered agents and seven (7) tokens in total, the BH can be successfully located using $\Theta(n)$ moves in a labeled torus. Also, k scattered agents can locate the BH using one token per agent after executing $O(k^2 n^2)$ moves in a labeled torus.
- using n scattered agents and one (1) token per agent, the BH can be successfully located in a unoriented complete network with $O(n^2)$ moves in total.

The results of this thesis constitute four major contributions:

First, we observe that it is rather unrealistic to have at least $O(\log n)$ bits of local storage available all the time to agents to access through fair mutual exclusion (as is the case using whiteboards). Also, face-to-face recognition³ (which is commonly hypothesized) is not easily realizable in practice. Consequently, obtaining a solution to the BHS problem that requires neither local storage nor face-to-face recognition constitutes, in our opinion, a significant improvement to the state of the art for this problem. We present such a solution, which is based on the token model. It is important to note that this token model imposes more constraints on the BH search

³Agents are able to see/identify each other

problem than the whiteboard model does, since it requires that both local storage and face-to-face recognition be avoided.

Our second contribution is two-fold. On the one hand, we claim that, costwise (with respect to memory usage and number of moves), the algorithms we develop show the token model to be as efficient as the whiteboard model [45, 46, 49]. This is important given the fact that the token model is more constrained than the whiteboard one. On the other hand, we contend that investigating the token model separately for each of the four abovementioned topologies results in algorithms that are more efficient than a general, topology-independent, one. Thus, our second contribution is to suggest that, for BHS, topology-specific token-based algorithms are as powerful (i.e., able to solve the problem) and as efficient as those based on a whiteboard model.

Third, we want to generalize our results by considering two flavors of the BHS problem, namely: using either co-located or scattered agents. The latter are agents that start at different locations in the network, whereas the former all start at the same node. More precisely, upon waking up, scattered agents start from different nodes, which are called *homebases*. Since such agents start from different *homebases*, they may not agree on a same sense of direction when the network is unoriented. Also, contrary to co-located agents, the scattered agents must initially execute the exploration of the topology at hand individually (that is without cooperating with other agents). This fact leads to a cost increase in terms of the total number of moves. In order to reduce this move cost, we let all the agents form pairs (or ‘gather’) first. Then we try to eliminate all the extra agents once at least two agents become co-located. The lack of a shared sense of orientation and the absence of an initial common strategy greatly increase the complexity of the BHS problem when using scattered agents. Indeed, even when using a whiteboard model, BHS with scattered agents has only been studied in the ring topology [44]. And thus solving the BH

search problem in three topologies using scattered agents constitutes, in our opinion, a significant contribution.

Last, we study several performance impact factors (namely: team size, knowledge of team size, token cost, sense of direction, FIFO and connectivity of the network topology) and investigate the trade-offs between them. Such an investigation of these performance impact factors should considerably help the researcher to choose the best strategy to solve the BHS problem under different environment constraints.

The premise for these contributions is the postulate that the following are open problems:

1. solving the BHS problem while avoiding whiteboards.
2. establishing whether known bounds for the BHS problem can be improved by considering specific network topologies (as opposed to not making assumptions about topology).
3. solving the BHS problem with scattered mobile agents.

In summary, we believe our proposed contributions directly address these three open problems.

1.3 Thesis Organization

In Chapter 2, we will introduce the framework, summarize the literature and explain the research method. We will introduce 15 BHS algorithms over Chapter 3, 4, 5 and 6. In Chapter 3 we are going to study the BHS problem in the ring topology in both the co-located agents and the scattered agents cases. In Chapter 4, the BHS problem is investigated in the hypercube topology with co-located agents. The BHS algorithms for tori are presented in Chapter 5. In Chapter 6, BHS in complete networks is

studied. Finally, in Chapter 7, we conclude this dissertation by recapitulating our whole research in Section 7.1, before Section 7.2 that offers a comparative evaluation. The chapter finishes by pointing to some future work in Section 7.3.

Chapter 2

The Framework, Literature Review and Research Method

2.1 Model and Framework

We are going to study the BHS problem in four types of network topologies, namely: ring, hypercube, torus and complete network. We first briefly introduce these four topologies in 2.1.2, then, in 2.1.1, we discuss the aspects of our research framework shared across these topologies.

2.1.1 Overall Framework

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} . A vertex cut of \mathcal{G} is a set of vertices whose removal renders \mathcal{G} disconnected. The *vertex connectivity* is the size of a smallest vertex cut. A graph is called k -connected or k -vertex-connected if its vertex connectivity is k or greater; a complete graph with n vertices has no cuts at all, but by convention its connectivity is $n - 1$. We assume that the graph is *anonymous*, that is, the nodes of the graph do not have any unique identifier. 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 consistent of such labeling (or sense of direction) of all the edges (links), *unoriented* otherwise [60].

Operating on \mathcal{G} is a set of k agents a_1, a_2, \dots, a_k . A Mobile Agent is a software

entity, with social ability (communicate with each other), computing, and most important, mobility (i.e., can move from a node to a neighboring node). The agents have limited computing capabilities and bounded storage. They obey 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* [45]. Also, these agents are *anonymous* (i.e., do not have distinct identifiers), *autonomous* (i.e., each has its own computing and bounded memory capabilities). They may start at the same node, called *homebase* (\mathcal{HB} for brevity). But the agents may also start at different nodes, each of which is also called a \mathcal{HB} . Recall that when all the agents start from the same \mathcal{HB} , they are said to be *co-located* agents. When the agents do not start from the same \mathcal{HB} , they are said to be *scattered* agents. Different agents may start at different and unpredictable times regardless of being co-located or scattered. Agents do not know how many other agents woke up (being active) before them.

We postulate that, while executing a BH search, 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 of a node or on a port of the 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 some fixed number of tokens (which depends on the specific algorithm, as we will explain later).

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. (An agent therefore may have to evaluate several multiplicities for its current node.) Second, an agent may *modify* the tokens (by placing/removing some of the tokens at the current node). Third, an agent may either become *Passive/fall asleep* (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. This computational 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.

Note that the tokens are the only means for inter-agent communication. There is no read/write memory (e.g., whiteboards) for the agents to access in the nodes, nor is there face-to-face recognition. In fact, the agents do not even see each other — they can only see the tokens. This computation is *asynchronous*: in the sense that, the time an agent sleeps or is on transit is finite but unpredictable.

Most importantly, it is postulated that one, and exactly one, of the nodes of the network is highly harmful — it disposes of every agent that enters it, without leaving any trace of this destruction observable from the outside. Due to this behavior, we will call this node the *Black Hole* (or BH for brevity). 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 (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 we are interested in 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).

2.1.2 Four Network Topologies

The choice of interconnection network [1] topology remains a critical subject in the design of efficient distributed networks. Most of the performance limitations are due to the performance of the communication system. Between extreme cases such as weakly connected circular rings or strongly connected complete networks, a solution will result from a compromise in order to satisfy, as it was mentioned by Hillis [72], a set of sometimes incompatible requirements: small degree and small diameter, bounded degree and expandability, fault tolerant connectivity and efficient layout, and so forth. Moreover it should be of obvious interest for the routing system that the topology may provide symmetrical schemes for global communications [35]. A symmetrical scheme means that all nodes can behave in a similar way and from this fact will arise a maximum simplicity in the design and processing of the system's communication kernel. More precisely, symmetry in the topology means that the representative graph is provided with an algebraic group structure as it is the case for the hypercube and some other families of Cayley graphs [1]. The hypercube therein appeared as a promising topology in the past decade in addition to its property of minimum broadcast graph [98]. Unfortunately, degree increasing with size brings on one troublesome hardware drawback with respect to the expandability requirement. That is not the case of torus, its vertex-transitive extension. This feature gives a renewal of interest to this topology which had been the first one to be proposed for parallel computers [107] and, in spite of a rather large diameter, returns up-to-date [96].

In this dissertation, we consider the BHS problem in four topologies. They are:

two extreme cases such as weakly connected circular rings and strongly connected complete networks, and two cases in the middle: hypercubes and tori.

2.2 Overall Literature Review

The research on safe *exploration* of unknown graphs was started by Shannon in 1951 [101]. Most of the work since then has focused on single mobile agent exploration (e.g., [2, 9, 13, 34, 37, 39, 54, 65, 91, 94]).

For exploring arbitrary anonymous graphs, various methods of marking nodes have been used by different authors. Bender *et al.* [13] proposed the method of dropping a pebble [73] (i.e., a token) on a node to mark it and showed that any strongly connected directed graph can be explored using just one pebble, if the size of the graph is known, and using $O(\log \log n)$ pebbles otherwise. Dudek *et al.* [54] used a set of distinct markers to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [14] was to employ two cooperating agents, one of which would stand on a node, thus marking it, while the other explored new edges. In Fraigniaud and Ilcinkas [63, 64] marking was achieved by accessing whiteboards located at nodes, and their strategy explored directed graphs and trees. In [39, 64] the authors focused on minimizing the amount of memory used by the agents for exploration. However they did not require the agents to construct a map of the graph.

Safe explorations by multiple agents were initially studied for a team of finite automata [18, 19, 21, 22, 23, 28, 82, 95, 97]. More recently such investigations have focused on collaborative exploration by *Turing machines*. As mentioned above, a two agent exploration algorithm for directed graphs was introduced in [14], whereas algorithms for exploration by more agents were given by Frederickson *et al.* for arbitrary graphs [66], by Averbakh and Berman for weighted trees [8], and more

recently by Fraigniaud et al. for trees [63]. Finally, an algorithm has been recently presented by Das et al. for constructing the map of an arbitrary unknown anonymous graph by scattered (or dispersed) agents making use of whiteboards [32].

The *Black Hole Search* problem has been recently investigated in different contexts. The problem posed by the presence of a harmful host has been intensively studied from a programming point of view (e.g., see [75, 99, 109]), and recently also from an algorithmic perspective [45, 46]. In asynchronous environments, it has been studied when the network is an anonymous ring, characterizing the limits and presenting optimal solutions [45]. In [46], the problem has been considered when the network is arbitrary and optimal solutions have been given under different assumptions about the level of topological knowledge available to the agents. The results of [42] achieve an optimal linear cost for many important interconnection networks. Finally, an efficient protocol for networks of arbitrary but known topology has been presented in [47]. In a synchronous environment, the tree has been investigated in [29] and optimal solutions given, with general results presented in [79].

In all these investigations, the nodes of the network have available to them a *whiteboard*, that is, a local storage area that the agents can use to communicate information. Access to the whiteboard is gained by mutual exclusion and the capacity of the whiteboard is always assumed to be at least of $O(\log n)$ bits.

Among the many problems of interest in mobile agent computing, only *rendez-vous* has been researched in both the whiteboard [11] and the token models [85]. In the former model, this problem, which requires the agents to gather in the same node, has also been investigated in the presence of a BH [43].

The BH location problem has been studied with a concern on identifying conditions for its solvability and determining the smallest number of agents suffice for its solution [45, 46]. The BHS problem with whiteboard in a ring topology was studied by [45].

To conclude, recall that the premise for this dissertation is that the whiteboard model rests on the rather unrealistic requirement that each node of the network to have a whiteboard.

In [48], BHS by mobile agents using a whiteboard model is studied in some inter-connection networks including the hypercube and torus topology. n moves is proved to be sufficient for BHS, when there is a labeled map [2, 13, 34, 37, 39, 63, 94] available for each agent.

The most recent research on the BHS problem with tokens is presented in [44]. A general solution on an unknown graph with $\Delta + 1$ mobile agents, $O(\Delta^2 M^2 n^7)$ moves (here, M is the number of edges in the graph, n is the number of nodes in the graph, Δ is the maximum number of degree of the graph) is presented.

We continue the literature research pertaining to each individual topology in the following subsections.

2.2.1 Existing Research on the Hypercube

Many graph-theoretic results about hypercubes are known. In 1984, L. N. Bhuyan and D. P. Agrawal [16] studied generalized hypercube structures for a computer network. Wu [111], Bhatt and Ipsen [15] both studied the embedding trees in hypercube and published their result in 1985.

In 1988, Saad and Schultz [98] proposed a theoretical characterization of the n-cube as a graph and showed how to map various other topologies into a hypercube.

In 1991, Kaklamani proved that the bounds for Oblivious Routing in the Hypercube are tight [77].

In [106], an election algorithm for the oriented hypercube, where each edge is assumed to be labeled with its dimension in the hypercube was proposed. The algorithm exchanges $O(n)$ messages (where n is the size of the cube). A randomized version

of the algorithm achieves the same (expected) message bounds, but uses messages of only $O(\log \log n)$ bits and can be used in anonymous hypercubes. In 1997, Kranakis and Krizanc studied distributed computing on anonymous hypercube networks [84]. Variations of the hypercube networks have been proposed by several researchers. Hamiltonian properties of hypercube variants are explored in [5, 7, 25, 81, 98].

Dobrev and Ruzicka studied Linear Broadcasting and $n \log \log n$ election in an unoriented Hypercube in 1997 [50]. Optimal Leader Election in a labeled Hypercube was presented by Flocchini and Mans [59].

In [42], BHS by mobile agents in Hypercubes and related networks were studied using *whiteboards* with co-located agents. Then, in 2005, Flocchini addressed contiguous search in the Hypercube for capturing an intruder [56].

In [48], BHS by mobile agents using a whiteboard model is studied in some interconnection networks including the hypercube topology. n moves is proved to be sufficient for BHS, when there is a labeled map [2, 13, 34, 37, 39, 63, 94] available for each agent.

2.2.2 Existing Research on the Torus

Traditionally, a torus is oriented if its node have assigned their communication links North-East-South-West labels in a globally consistent manner [78]. In [57], a new way of labeling tori is presented. In 1988, Bodlaender [20] studied several transitive networks such as: ring, complete network, a 2-dimensional grid network with boundary connections, i.e. the torus.

Broadcasting a message to all the other nodes of an asynchronous totally unlabeled torus was studied in [40] in 1998. The same year, under the same assumption (anonymous unoriented Tori), Dobrev and Ruzicka improved the lower and upper bounds achieved in [51]. The proposed algorithm works also on non-square tori, does

not require the knowledge of sizes n and m , and uses only messages of size $O(1)$ bits. This is the first known broadcasting algorithm on an unoriented torus that does not use all edges.

In [87], the message complexity of distributed algorithms in Tori and Chordal Rings when the communication links are unlabeled was studied. A preprocessing algorithm was introduced to introduce the notion of a handrail: a partial structural information that allows messages to travel with a globally consistent direction.

Flocchini, Huang and Luccio studied the decontamination problem in chordal rings and tori in [57]. In [70], the authors study the map construction problem using mobile agents in an anonymous, unoriented torus of unknown size. An optimal result was achieved with one token per agent, and $\Theta(n)$ moves in total.

2.2.3 Existing Research on the Complete Network

In 1989, Korach, Moran and Zaks presented algorithms for leader election and spanning tree construction in an asynchronous complete network in [80]. An optimal complexity of $O(n \log n)$ was achieved. Singh [104] presented a Leader Election algorithm that requires $O(n)$ messages and $O(\log n)$ time with sense of direction. In the same paper the lower bound $\Omega(n \log n)$ was achieved for an asynchronous complete network without sense of direction. In 1998, Santoro and Mans presented an optimal Leader Election protocol in a faulty loop network, which requires only $O(n \log n)$ messages in the worst-case, where n is the number of processors [89]. Recently, [108] analyzed algorithms for leader election in complete networks using asynchronous communication channels. The paper presented a novel algorithm that reduces the information necessary to select a leader when compared with other leader election algorithms for complete networks. In that paper, the algorithm works without sense of direction. It achieves $O(n)$ messages and $O(n)$ time without requiring knowing the number of

nodes in the system.

Cockayne studied multi-message broadcasting in complete graphs in 1980. An optimal solution was presented in [26]. Dobrev [41] studied the broadcasting problem in complete networks with dynamic faults.

The BHS problem has not been studied specifically on complete networks. The closest research consists in general algorithms on arbitrary graphs in the whiteboard model [46, 79], as well as for exploring a dangerous unknown graph in token model [44].

2.3 Research Method

As previously mentioned, we want to consider four specific topologies namely: ring, hypercube, torus and complete network. For both co-located and scattered agents, we plan to solve the BHS problem in those topologies using tokens.

Some basic problems, such as *Leader Election* [38, 100, 103, 104, 108], *Broadcasting* [26, 40, 41, 50, 51], have been solved on the topologies that interest us in this dissertation. We observe that the broadcasting problem appears to be relevant for the BH search problem, since during this search, the mobile agents must traverse the whole graph in order to locate the BH in the network. Similarly, we believe that results obtained in Leader Election algorithms with respect to reducing the number of active agents in order to minimize the total number of moves could help with finding the minimal team size required to solve the BH search problem. Ultimately, we remark that existing research on hypercube, torus and complete network, has been helpful in devising algorithms for solving the BHS problem on the proposed topologies.

As we mentioned earlier that the ring is the sparsest bi-connected graph. It is the most challenging topology and thus it is the first we tackle, in the next chapter. Having obtained a solution for BH search in a ring using tokens, we then consider

the torus and hypercube topologies. Once we have solutions for these two topologies, we then tackle the complete network next since it has the highest connectivity. We observe a very interesting fact: contrary to the case of co-located agents, in the case of scattered agents, the degree of complexity of a BHS algorithm is directly proportional to ‘the connectivity of a topology.

We initially selected these topologies because they are commonly used in practice and frequently referenced in literature. In turn, we will show for each topology, that this allows us to determine the minimum size for the team of agents used for a search.

But there is more to say on this selection of topologies. Indeed, we remark that in earlier separate work [102, 103], we had suggested that it is possible to identify common aspects in a family of algorithms for a single topology. In this dissertation, we have reused some of those ideas to approach the task at hand. More specifically, during our research, we came to believe that some of the insights gained in developing a solution to the BH problem with tokens in a ring were reusable across the other topologies we consider. This belief rests on the observation that there are common characteristics between all the topologies we consider. Indeed, the hypercube, the torus and the complete network can all be thought of as multi-connected rings. For example, any torus topology with a dimension of more than $3 * 3$ can be presented as a donut shaped graph (see Figure 2.1); a hypercube with more than 2 dimensions can be presented as a structure that consists of rings with common vertices (see Figure 2.2); and a complete graph is known to be a fully connected chordal ring [88] (see Figure 2.3). This viewpoint suggests that some aspects of our solutions to the BHS with tokens in a ring can be reused in these other topologies. In particular, the “cautious walk with token” technique presented in chapter 3, indeed appears to be highly reusable in these other topologies.) Consequently, the possibility of such reuse reinforces, in our opinion, the coherence of the set of topologies under investigation

in this dissertation.

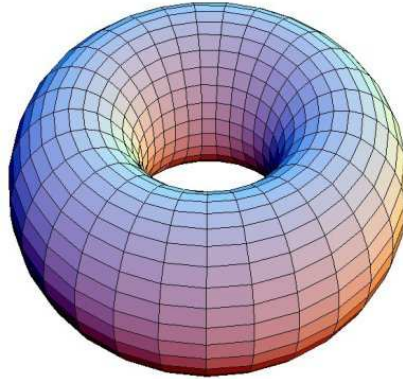


Figure 2.1: A Torus

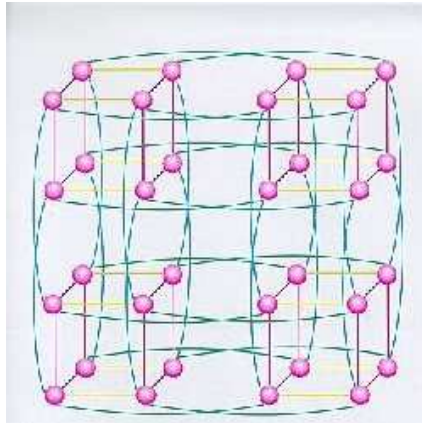


Figure 2.2: A 5-Hypercube

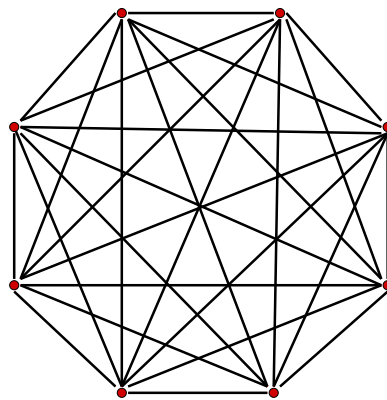


Figure 2.3: A complete graph with n ($n \geq 3$) nodes is a full connected Chordal Ring

Finally, from a methodological viewpoint, we add that once we obtained solutions for the torus topology, we compared them to the solutions we already had for the ring (for both co-located and with scattered agents). Most interestingly, the algorithmic and complexity similarities and differences between these two sets of solutions provided a clear indication of how to proceed with the hypercube and the complete network. In particular, in our opinion, considerable algorithmic resemblance suggests looking for a “core” algorithm applicable to a family of “ring-relevant” topologies. We also noticed that the complexity measures we had determined directly stemmed from the specifics of each of the selected topologies. Ultimately, this leads us to prove (in Section 7.2, Chapter 7) whether topology-specific solutions offer better performance than topology-independent ones for the problem at hand.

Chapter 3

Black Hole Search in Rings

In this chapter, the network under consideration is an asynchronous ring of n nodes with a single BH. In such a network, it is known that when using whiteboards, the BHS problem can be solved with a team of just two agents, and performing only $\Theta(n \log n)$ moves[45]. Here, we consider the same topology and examine the BHS problem using tokens. We consider two sub-problems: using co-located agents and using scattered (or equivalently dispersed) agents.

For the co-located agents case, we first suggest that a team of two agents is sufficient to locate the BH in finite time (even though the token model is a weaker coordination model than whiteboards). Furthermore, we claim that this can be accomplished using only $O(n \log n)$ moves in total, which is optimal, as when using whiteboards. Finally, we show that the agents need to use only $O(1)$ tokens. These results are obtained iteratively from improving an initial simplistic solution. The first improvement we present uses a total of 10 tokens. In our second improvement, that number is reduced to 3.

The point to be ultimately grasped from these algorithms is that, although tokens are a weaker means of communication and coordination than whiteboards, their use does not negatively affect solvability and it does not even lead to a degradation of performance. On the contrary, whereas the protocols using whiteboards assume at least $O(\log n)$ dedicated bits of storage at each node, the token-based solutions proposed here use only three tokens in total. In other words, we contend that, with

respect to memory requirements, our token-based solutions are less demanding than whiteboard-based ones.

For the scattered agents case, when the orientation of the ring is available, we show that a minimum of 2 agents can locate the BH within $O(kn + n \log n)$ moves with only 1 token per agent, here k is the number of agents. Then we prove that a BH can be located in an anonymous ring by anonymous asynchronous and scattered agents, using $O(n \log n)$ moves in total and four (4) tokens per agent without any knowledge of the team size.

Also, we show that in an unoriented ring, a team of k scattered agents, where $k \geq 3$, can locate the BH within n^2 moves, each agent using $O(1)$ tokens without knowing k (i.e., the total number of the agents). Interestingly, we can reduce the number of moves to $O(n \log n)$, which is optimal, with a minimum of 4 scattered agents and $O(1)$ tokens per agent. These results hold even if both agents and nodes are *anonymous*.

3.1 Basic Tool and Observations

3.1.1 Basic Tool — Cautious Walk with Tokens (*CWWT*)

Cautious Walk with Token (henceforth *CWWT* for brevity) is an adaptation of the *cautious walk* technique used in systems with whiteboards [45]. This algorithm is the 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). Whereas the details of how to establish that a port is with or without tokens will differ across the algorithms we will introduce throughout the dissertation, this classification holds: for *CWWT*, a port is always

determined to be with or without tokens.

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. For brevity, we will simply say that the port is being explored. 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, its link 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 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 being visited by a .

What agent a_2 does next depends on the specific algorithm that uses *CWWT*. Typically, a_2 will return to v and continue exploring the topology at hand. However, the presence or absence of tokens in u and/or on p may lead to different behavior for a_2 , as will be explained later.

To conclude, we must emphasize that *CWWT* works if, and only if, agent a_2 cannot modify v in anyway before a_2 has first returned to u to indicate p and the link of p are *safe* and then gone back to v . In turn, this going-back-to u followed by returning-back-to v is a single atomic instantaneous action.

3.1.2 Observations

It is essential to understand that, regardless of topology, because of the asynchrony, agents cannot distinguish a slow node from a BH. From this we get:

Lemma 1 [45] *It is impossible to find the BH if the size of the topology is not known.*

In the co-located agents case, all the agents can agree on a same sense of direction. But the first move of an agent can end up in the BH, regardless of the topology, we immediately get:

Lemma 2 [45] *At least two agents are needed to locate the BH in the co-located agents case in any topology.*

In the case of scattered agents in ring topology, we observe that:

- since all the agents may start from different nodes upon wake-up, there may be no common agreement on the orientation of the ring;
- there are two links leading to the BH;
- the first move of an agent can end up in the BH,

From Observation 1, we know that 2 scattered mobile agents are not enough to locate the BH in an unoriented ring.

3.2 BH Search by Co-located Agents

3.2.1 Elimination Technique

In order to minimize the *cost*, that is the total number of moves executed by all agents, we use an elimination technique to reduce the number of active agents to two in the co-located versions of our algorithms — namely, the first two agents to wake-up. The subsequent co-located BHS algorithms of this chapter are therefore described using two agents, and assume this elimination technique has first been applied. Let us elaborate on this technique.

Upon initial wake-up, an agent examines the \mathcal{HB} . There are three possibilities:

- *There is no token on any port.* This means the agent a_1 is the first to wake-up. In this case, place a token on one port, and this determines the orientation. We call this port a right port. Then a_1 will execute the BHS algorithm by starting exploration from the right port of the \mathcal{HB} . a_1 will be referred to as the right agent.
- *There is/are token(s) only on the right port.* This is the second agent to wake-up. Place a token on the left port and execute the BHS algorithm by starting exploration from the left port of the \mathcal{HB} . This agent will be called agent a_2 and be referred to as the left agent.
- *There are tokens on both ports.* This means two agents already woke-up before. The current agent becomes *Passive*, namely, ceases to participate in the computation.

3.2.2 General Description and Basic Ideas

A node is *explored* if it has been visited by an agent, *unexplored* otherwise. The set of the explored nodes is called the *explored region*; the part of the ring consisting of

unexplored nodes, the *unexplored region*. As all agents start from the same \mathcal{HB} , the explored region and unexplored region partition the ring into two connected parts. In this ring, the two explored nodes that have an unexplored neighbor are called the Last-Safe-Places (LSPs). At the very beginning of the search, the \mathcal{HB} is the sole LSP. Clearly, the LSPs keep changing while the explored region is getting larger and larger. Exploration is performed using the *CWWT*.

At any moment in the execution, a port can be classified as

- *unexplored* — no agent has exited or arrived via this port yet;
- *dangerous* — an agent has exited via this port, but no agent has arrived yet via it, or
- *safe* — an agent has arrived via this port.

As previously explained, the *CWWT* is used to make a port *safe*.

Reduction Technique

The main technique for locating the BH in a ring with co-located agents is borrowed from [45]. We sketch it out here and then provide additional details in the next subsection. Each agent holds a numeric counter that can be increased and decreased by 1 according to each specific algorithm. The two first agents to wake up logically partition the unexplored region into two connected parts: $\lceil (n-1)/2 \rceil$ for the right agent a_1 and to $\lfloor (n-1)/2 \rfloor$ for the left agent a_2 , and then each agent goes to explore its part using *CWWT*. Since there is only one BH, one of the two agents (say agent a_1) will finish exploring its part. Agent a_1 then traverses the explored part until it reaches the LSP of agent a_2 . The counter of a_1 is used so that, at that moment, a_1 will know the distance between the LSPs, which corresponds to the size of the *explored region*. Since n is known, a_1 can compute the current size of the *unexplored region*. If

this unexplored part consists of a single node v , a_1 determines that the BH is located at v and terminates the algorithm (since a_2 has already been terminated by entering the BH). Otherwise, a_1 divides the remaining *unexplored region* into two connected, almost equally sized work assignments W_{a_1} and W_{a_2} for a_1 and a_2 , such that W_{a_2} contains the node to which a_2 is currently heading. Then, a_1 leaves a message for a_2 (informing it about W_{a_2} , as explained shortly) and goes on to explore W_{a_1} . The process is repeated until the *unexplored region* contains a single node — the BH.

Communication between agents therefore involves an agent leaving or receiving a message. We investigate this idea next.

Communication Technique

The previous description omits details of how the agents communicate, how to identify an LSP, and how to implement the *CWWT*. In [45] whiteboards were used to store a message, as well as the status of the ports. In our research, we aim to implement a similar approach but using only tokens. In other words, communication between agents must be achieved using tokens. Furthermore, we aim to use an overall constant number of tokens.

The first step towards this goal is to use tokens only for *dangerous* ports (possibly leading to the BH). Thus, no tokens are used to identify *safe* or *unexplored* ports. This is feasible because all ports between the *dangerous* ports leaving from LSPs are implicitly *safe*.

The second step, namely using tokens to encode a message, is trickier because we want to use as few tokens as possible. In our algorithms for BH location using co-located agents, in the context of the previous description, the message left by a_1 for a_2 is not the size of W_{a_2} , but the number of times a_1 has finished its part before a_2 has made any progress. We will show later that this information is sufficient for

a_2 to continue its work assignment. For now, let M denote the number encoded in the message from a_1 to a_2 . When agent a_2 gets this message, it can re-compute the size of its work assignment by halving the size of its last work assignment M times (and being careful to use the same rounding as a_1 used when the unexplored area has odd size). The key observation is that M is conveyed from a_1 to a_2 using a single token and can be at most $\log n$. The basic technique we use to encode the message is to have a_1 put a token for a_2 at a location that allows a_2 to obtain M by counting from its node (its LSP) to the node where the token has been dropped in the *explored regions*.

If and when a_1 completes its current work assignment, it must increment M . This merely consists in picking the token corresponding to M and dropping it on the next node in the correct direction.

Finally, agent a_1 has to somehow signal to a_2 that there is a message waiting for it. How this is achieved will vary from algorithm to algorithm. The basic technical difficulty lies in the fact that the tokens are undistinguishable and seeing a token might mean very different things, depending on context. In addition, we should avoid situations where one agent needs more tokens than it has, while the other collects tokens that it does not need.

Plan for the Rest of the Section

In the rest of this section we introduce three algorithms: *Divide with Token*, *Divide with Token +*, and *Divide with Token -*. The first one uses $O(n \log n)$ tokens and simulates a whiteboard in each node. Given we want to reduce the token cost, this is not acceptable. However, it is crucial to understand that as we reduce the number of tokens used, the interpretation of tokens becomes more complex. More precisely, sometimes the same number of tokens placed in different locations may mean different

things, or the same number of tokens in the same location may have different meanings to an agent depending on the state of this agent. In the second algorithm, we use a constant number of tokens, and we try to avoid ambiguity by using different numbers of tokens to mean different things. This results in a fairly simple algorithm that, however, does not use the optimal number of tokens. Finally, the algorithm *Divide with Token* – aims to rectify this problem by reusing the tokens as much as possible and applying detailed context analysis when tokens are encountered.

All three algorithms presented in this section use the *CWWT* technique and depend on the FIFO requirement explained earlier.

3.2.3 Algorithm *Divide with Token*

If the number of tokens each mobile agent can carry is unlimited, all the messages used for communication in order to locate the BH in the whiteboard model, can be easily simulated by numbers of tokens in each node. The idea is simple: A certain number of tokens in each node can represent a certain meaning, that is, a certain message. Hence, it is quite straightforward to solve the BHS problem with an unlimited number of tokens after conducting some modifications on the algorithm presented in [45]. Such modifications merely consist in mapping each message that can be written/read from/to the whiteboard into a certain number of tokens. We call this simple algorithm *Divide with Token*. It follows that:

Observation 2 *The BHS problem can be solved in a Ring topology with tokens.*

Algorithm *Divide with Token* is based on the assumption that an unlimited number of tokens is available for all the agents. Obviously, this is an overly simplistic assumption and the problem becomes trivial to solve. Therefore, it is not useful to expand on this algorithm. Instead, we repeat, our objective is to have a constant

number of tokens available for each agent, so that the token model is really less expensive than the whiteboard model, as we hope to demonstrate.

3.2.4 Algorithm *Divide with Token +*

The basic idea of algorithm *Divide with Token +* is to use the following specific number of tokens to convey specific meanings:

- 1 token at a port — used for agent elimination at the \mathcal{HB}
- 2 tokens at a port — *dangerous* port at the LSP of an agent. We call these two tokens on the port the *CWWT* tokens.
- 3 tokens at a port — *dangerous* port with a message for the agent
- 1 token in the middle of a node — so-called *end-point marker* used to capture M (i.e., the number of times a_1 has finished its part before a_2 has made any progress, as explained above). The length of the segment of the *explored region* that starts from the LSP of a_2 and ends at the node with this *end-point marker* is M .
- 2 tokens in the middle of a node — the message from one agent to the other has been read. These two tokens can be picked up and reused for the next message

Beyond these conventions, we require that each agent maintain the following variables:

- *Steps*: the size of the remaining work assignment, initialized to $\lceil (n-1)/2 \rceil$ for the right agent a_1 and to $\lfloor (n-1)/2 \rfloor$ for the left agent a_2
- *MsgILeft*: used to store the value of M that the agent at hand is to leave for the other agent
- *DistC* — used to calculate the distance between LSPs

- $Msg4Me$ — used to store the value of the M that the other agent left for the agent at hand

The values of both $MsgLeft$ and $Msg4Me$ are computed while traversing the *explored region*. This is explained in the description that follows.

Algorithm Description

The algorithm is described for the right agent a_1 (which will be referred to using the first person singular in such descriptions throughout the dissertation). The algorithm for the left agent a_2 is almost identical; the only differences being the use of opposite directions, and the use of the floor function instead of the ceiling one when calculating the work assignment (see Subsection 3.2.2).

In the initialization step, the previously-mentioned elimination technique is used to limit the number of active agents to two. This technique also determines the right agent a_1 and the left agent a_2 . Note that unlike [45], the initial work assignment of the first agent is the whole *unexplored region* so that we do not need to deal with the case where the first agent has explored its part, while the second agent has not started the algorithm yet. When the second agent wakes up, it will seek the first agent to divide the workload based on what remains unexplored at that moment.

In procedure “Explore” an agent explores its work assignment using *CWWT*, checking for messages from the other agent while exploring. Here, checking for a message means detecting an *end-point marker* in the middle of a node.

Algorithm 1 Algorithm *Divide with Token +* — Procedure “Initialization” and “Exploring”

```

1: INITIALIZATION:(upon initial wake-up in the  $\mathcal{HB}$ )
2:  $DistC = 0, Mes4Me = 0, MesLeft = 0$ 
3: if the right port has no token on it then
4:   mark as dangerous: put two tokens on the right port
5:   execute procedure EXPLORE( $n - 1$ ) as the right agent  $a_1$ 
6: else if the left port has no token on it then
7:   mark as dangerous: put two tokens on the left port
8:   execute procedure SEEKING() as the left agent  $a_2$ 
9: else
10:  become Passive immediately
11: end if
12: procedure EXPLORE(Steps)
13:  while true do
14:    go from the current node  $u$  to its right neighbor  $v$ 
15:    return back to node  $u$  // as per Cautious Walk
16:    if there are still two tokens on the right port of  $u$  then // no message for me
17:      if  $u$  is the  $\mathcal{HB}$  then
18:        remove one token from the right port of  $u$ 
19:      else
20:        remove two tokens from the right port of  $u$ , which is safe
21:      end if
22:      move to  $v$ , put two tokens on the right port of  $v$  and decrement Steps.
23:      if Steps = 0 then // finished exploring my assignment
24:        exit the loop and execute SEEKING() // now find the other agent
25:      end if
26:    else // there must be 3 tokens on the right port — a message waiting for me
27:      exit the loop and execute procedure CHECKING() // check the message
28:    end if
29:  end while
30: end procedure

```

In procedure “Seeking” the agent determines the distance between the LSPs and then either locates the BH (if there is a single unexplored node remaining) or leaves/updates the message for the other agent. Several cases need to be handled:

- this is the first message for the other agent,
- a message for the other agent has already been left before, but that agent did not read it yet and now the message must be incremented,
- a message for the agent has already been left and it has been read, a new message should be left now.

Furthermore, an agent executing procedure “Seeking” must not get confused by the tokens remaining in the \mathcal{HB} after the elimination technique.

The details of procedure “Seeking” follow (for a right agent, as usual):

Algorithm 2 Algorithm *Divide with Token +* — Procedure “Seeking” and “Check & Split”

```

1: procedure SEEKING
2:   go to the left until a token is encountered in a node  $u$ . Use variable  $DistC$  to count
   the distance traveled
3:   if found two tokens on the left port then //  $u$  is the LSP of the other agent
4:     put a third token on the left port and also put a token in the middle of  $u$ 
     // indicate that there is a message, and the message's value is “0”
5:     execute CHECK&SPLIT( $DistC$ )
6:   else if found one token in the middle of a node  $u$  then // this is the end-point
   marker of agent  $a_2$ 's last message and has not been yet
7:     remove the token, go to the node on the right, put the token there and increment
    $MsgLeft$  // update/increment the message
8:     execute CHECK&SPLIT( $DistC + MsgLeft - 1$ )
9:   else if found two tokens in the middle of a node  $u$  then // the other agent read
   my previous message
10:    remove the two tokens from the middle and continue from the beginning of
   SEEKING, not resetting the  $DistC$  counter
11:   else if found two tokens — one each on each port then // this is the HB, ignore
12:     ignore them and continue from the beginning of SEEKING, not resetting the  $DistC$ 
   counter
13:   end if
14: end procedure

15: procedure CHECK & SPLIT( $Dist$ )
16:   if  $Dist = n - 2$  then // single unexplored node remaining
17:     become DONE, the BH is in the remaining unexplored node
18:   else
19:     go right until two tokens are found on the right port // return to your LSP
20:     execute Explore( $\lceil (n - Dist)/2 \rceil$ ) // new work assignment, the left agent would
   use floor here
21:   end if
22: end procedure

```

We emphasize that procedure “Checking” is executed by the agent that notices a message has been left for it. The indication of such a message is simply one extra token, for a total of 3 tokens, on the port on which the agent left its *CWWT* tokens. Then, this agent “reads” the message by traversing leftward until an *end-point marker* is found. Note that, as captured in the algorithm above, for the first message this

means zero leftward moves.

The agent “reading” the message must then indicate to the other agent that the message has been “read”. This is achieved by placing another token “on top of” the *end-point marker*. In other words, tokens are undistinguishable but, conceptually, we suggest this extra token be thought of as a “message-read” token. The key point is that indicating a message has been read allows the other agent to reuse these tokens for a next message. Here are the details:

Algorithm 3 Algorithm *Divide with Token +* — Procedure “Checking”

```

1: procedure CHECKING
2:   remove the third token from the right port
3:   go left until a token is found in the middle of a node  $u$ , counting in  $Msg4Me$  the
   number of nodes traversed
4:   put a second token in the middle of  $u$ 
5:   go  $Msg4Me$  nodes to the right // returning to the LSP
6:   for ( $i = 0; i \leq Msg4Me; i++$ ) do // compute the new work assignment
7:      $Steps = \lceil Steps/2 \rceil$  // again, floor function will be used by the left agent
8:   end for
9:   execute EXPLORE( $Steps$ )
10: end procedure

```

Correctness

In this section, we discuss the correctness of algorithm *Divide with Token +*.

Lemma 3 *At most two agents will become active.*

Proof: According to the Elimination technique explained in Subsection 3.2.1, any other agent wakes up later than the first two agents will become *Passive* immediately once noticing the tokens in its \mathcal{HB} .

□

The following observation is trivial to prove:

Observation 3 *The nodes in the part of the ring that is between two Last-Safe-Places (LSPs) and includes the \mathcal{HB} , are safe.*

Given this, let us start by addressing the correctness of what we introduced earlier as the communication (as defined in subsection 3.2.2) facet of our algorithm:

Lemma 4 *Communication between agents a_1 and a_2 works correctly.*

Proof: As previously explained, M is the number of times a_1 has finished its work assignment before a_2 explores another node in the *unexplored region*. Recall that M is captured by leaving a token (called an *end-point marker*) in the middle of a node n . Agent a_1 leaves this token by counting M nodes from the LSP of a_2 . Also recall that when a_1 notices the message has not been read, it must increment it. This is achieved by a_1 picking up and moving the *end-point marker* to the neighbor of n away from the LSP of a_2 . Most importantly, because of the FIFO requirement, no race condition can occur between a_1 and a_2 during this moving of the *end-point marker*: a_2 cannot overtake a_1 and therefore will find the *end-point marker* in the correct location.

Once a_2 detects the message of a_1 , it removes the *end-point marker*. Consequently, the next time a_1 seeks a_2 , it will not encounter an *end-point marker* and thus will end up in the LSP of a_2 . Given we do not allow overtaking, the behavior of the agents will depend on which of the two arrives first at the LSP of a_2 . In this context:

- if a_1 arrives first, then a_2 will notice an extra token (as previously mentioned) requiring it to go and check the message a_1 left for it. Consequently, a_2 will follow a_1 until a_1 leaves its *end-point marker*. Then a_2 will proceed with its new work assignment.
- if a_1 arrives after a_2 , then a_1 will follow a_2 . Agent a_2 will move its LSP to the node it just *explored*, and leave its *CWWT* tokens on the next *dangerous* port of that node. Agent a_1 will consequently expand the size of the *explored region*

and proceed to leave a message for a_2 . To do so, a_1 will add a token to that *dangerous* port and then go and place its *end-point marker* before resuming its own exploration.

Recapitulating, we have shown that:

- M can be correctly incremented (through the moving of the *end-point marker*)
- once the *end-point marker* is removed: If a_1 arrives first at the LSP of a_2 , then a_1 will place a new *end-point marker* that a_2 will correctly pick up to continue its exploration. Furthermore, if instead a_2 arrives first at its LSP, then a_2 will continue its exploration, and a_1 will increment the size of the *explored region* before proceeding with leaving a message for a_2 and resuming its own exploration.

Thus we conclude that communication between a_1 and a_2 works correctly within the context of the algorithm at hand. More specifically, communication does not affect the ability of this algorithm to expand the *explored region* until the BH is found.

□

Next, we introduce lemmas that address the other facets of the algorithm at hand.

Lemma 5 *An agent that walks without using CWWT will only do so on safe edges/links.*

Proof:

According to the algorithm described earlier, the possible situations in which an agent walks without *CWWT* are:

- An agent a_1 goes to check the message the other agent a_2 left for it. Given the location of a_1 and the fact that this message (in the form of a token in the middle of a node) is at most $\log n$ away from a_1 (see 3.2.2), this walk is definitely between the two LSPs.

- An agent a_1 goes back to its LSP after seeking its partner a_2 or checking for a message left by a_2 .

In both cases above, the agent necessarily walks between the two LSPs. And we know this is *safe* given Observation 3. In fact, there is only one hypothetical situation in which an agent could possibly go beyond the *explored region*:

- An agent a_1 goes to seek the other agent a_2 and ends up overtaking a_2 and possibly ending in the *unexplored region*. This situation is impossible. More precisely, because of the previously introduced FIFO requirement and because a_1 cannot use the link of a port marked with token(s) by a_2 , it is impossible for a_1 to overtake a_2 .

Having shown that the situations in which an agent walks without *CWWT* occur only in the *explored* (or equivalently, *safe*) region, we conclude the Lemma is proven.

□

In the following Lemma we prove that at most one agent dies in the BH using this algorithm. Consequently one agent survives and locates the BH.

Lemma 6 *At most one agent dies.*

Proof: First, the elimination technique mentioned earlier guarantees there are only two agents used to locate the BH at the start of the search. Second, as previously mentioned, we know agents will never go through the same *unexplored* (i.e., *dangerous*) link at the same time. Consequently, no two agents will die in the BH using the same link. Third, through the use of our reduction technique from Subsection 3.2.2), as soon as an agent notices that the size of the *explored region* is $n - 1$ nodes, the task is achieved and the BH is located.

It follows from these three observations that after the first agent dies in the BH (leaving behind it a *dangerous* port), the second agent will not traverse that *dangerous* port but instead grow the *explored region* till it includes all the nodes but the BH. The key observation is that this second agent will never have to explore a *dangerous* port since the size of the *explored region* reaching $n - 1$ nodes will stop the algorithm before that.

□

Lemma 7 *Within finite time, one agent will determine the location of the BH.*

Proof: From Lemma 6, we know one agent will survive and find the BH. The question then is to know how many moves (i.e., link traversals) are required. We observe that the worst case, that is, the case that requires the greatest number of moves (as will be explained below), occurs when the BH is the immediate neighbor of the \mathcal{HB} . In this case, one agent immediately dies, and the other agent must grow the *explored region* to $n - 1$ nodes. Given our reduction technique, this requires $n \log n$ moves. And since each such move takes finite (but unpredictable) time, we can conclude that a finite amount of time is required, even in the worst case, for one agent to locate the BH.

□

Hence we conclude:

Theorem 1 *Algorithm Divide with Token + correctly locates the BH after finite time.*

Complexity Analysis

Let us start with two important observations:

Observation 4 *Because the agents start from the same \mathcal{HB} , even if the ring is un-oriented, the agents can agree on a common (be it clockwise or counter-clockwise)*

direction for the ring.

Observation 5 *Regardless how many agents start in the same \mathcal{HB} , we can always reduce the number of agents to two using the elimination technique described in Subsection 3.2.1. This technique requires two tokens.*

Theorem 2 *Algorithm Divide with Token + correctly locates the BH in a ring with n nodes, employing two co-located agents and ten (10) tokens in total, requiring $O(n \log n)$ moves.*

Proof: The proof of number of tokens used per agent follows:

Claim 1 *There are 4 situations in which token(s) are required:*

- *Eliminate extra agents in the \mathcal{HB} : one token is used by each of the two first agents to wake up. Those agents each mark one without token port of the \mathcal{HB} upon waking up. As previously explained, this guarantees 2 agents are used by the algorithm.*
- *CWWT: as previously explained, two tokens are used by each agent in order to explore the ring using CWWT.*
- *An agent uses one token (placed in the middle of a node) as an end-point marker (as explained at length earlier).*
- *An agent a_1 that leaves a message must use one token to notify its partner a_2 that a message (in the fork of an end-point marker) awaits a_2 . This notification token is placed as a third token on the dangerous port explored by a_2 .*

It follows that 5 tokens are used by each agent. Hence, algorithm *Divide with Token +* uses a total of 10 tokens for 2 agents.

Let us now briefly consider the number of moves required by each agent (based on the earlier description of the algorithm).

- Most importantly, in procedure “Explore” an agent explores its work assignment using *CWWT*. Each port explored costs 3 moves and there at most $n - 1$ nodes to explore. So exploration entails $3(n - 1)$ moves.
- Because an agent a_1 does not start to seek the other agent a_2 before a_1 has finished exploring approximately half of the *unexplored region*, and because an agent a_1 does not go to check the message the other agent a_2 left to it until the a_2 has finished exploring approximately half of the *unexplored region*, we infer that there will be at most $\log n$ calls to procedures “Seeking” and “Checking”.
- Let us now consider the cost, in terms of moves, for a single call to procedure “Checking” or procedure “Seeking”. The key observation is that an agent performing either of these procedures always walks in the *safe* region, which has a maximum of $n - 1$ nodes. Consequently, a call to procedure “Checking” or to procedure “Seeking” requires at most n moves.

Hence, there is a maximum of $3n + 2n \log n$ moves performed by the two agents. It follows that algorithm *Divide with Token +* correctly locates the BH employing two agents and ten (10) tokens in total, requiring $O(n \log n)$ moves.

□

3.2.5 Algorithm *Divide with Token* –

In this algorithm, our goal is to locate the BH while minimizing the token cost. Consequently, the use of tokens is carefully planned. We will distinguish:

- the presence from the absence of a token in a specific location
- using i tokens in one location from using i tokens in another location.
- for an agent a_1 , using i tokens in location l when a is in state s_1 from using i tokens in the same location l when a_1 is in state s_2

Detailed Description

The following variables, local to each agent, are used in this algorithm:

- *Steps* — the size of the remaining work assignment
- *HBpos* and *LSPpos* — relative position with respect to the \mathcal{HB} and the LSP of the agent. This allows an agent to know when it is at the \mathcal{HB} or at its LSP without using tokens for marking these
- *MsgLeft* — used to store the value of M that the agent at hand is to leave for the other agent (as in previous algorithm)
- *DistC* — used to calculate the distance between LSPs
- *Msg4Me* — used to store the value of M that the other agent left for the agent at hand (as in previous algorithm)

From the previous algorithm, we also reuse the definitions of what is a *safe* and what is a *dangerous* port.

For this algorithm, there are two key differences from the previous one. First, whereas two tokens were used to mark a port under exploration in algorithm *Divide with Token +*, here we use only one. Second, when agent a_1 wants to indicate it leaves a message for its partner agent a_2 , then a_1 “steals” the token a_2 left on the port it is exploring. a_1 uses this stolen token as the *end-point marker* for the message it leaves for a_2 . Most interestingly, agent a_2 detects a message was left for it by noticing that the token it left on the port under exploration is absent. Except for these two differences, the algorithm at hand works essentially like the previous one.

In this algorithm we will use tokens only to:

- mark a *dangerous* port (as usual, as part of the *CWWT*)

- encode M using an *end-point marker* (as in the previous algorithm)
- in the \mathcal{HB} in order to limit the number of active agents to two (as per the elimination technique introduced earlier)

Having a port under exploration use only one token (that may be stolen) is quite straightforward, except in the \mathcal{HB} . More precisely, because there are several possible combinations of tokens in the \mathcal{HB} , it is useful to enumerate them, along with their interpretations. To do so, we will use a triplet giving respectively the number of tokens on the left port of the \mathcal{HB} , its middle, and its right port:

- $(0, 0, 2)$ — the right port is *dangerous* (i.e. the \mathcal{HB} is the LSP of the right agent), the left port is *unexplored* (the second agent has not woken-up yet)
- $(0, 0, 1)$ — the right port is *safe* (the LSP of the right agent already moved to the right), the left port is *unexplored*
- $(2, 1, 0)$ — both the right and the left port are *dangerous*. There is a message of value 0 waiting for the right agent. Here the second agent woke up before the first agent explored its first port. In its initial step, the second agent changed $(0, 0, 2)$ to $(2, 0, 1)$, but then immediately transformed that to $(2, 1, 0)$ by executing the procedure “Seeking” (as per the description of the initialization step found shortly below).
- $(1, 0, 1)$ – the right port is *dangerous*, the left port is *safe*
- $(2, 0, 0)$ – the left port is *dangerous*, the right port is *safe*
- $(1, 0, 0)$ – both ports are *safe*
- $(1, 1, 0)$ – if seen by the right agent whose LSP is the \mathcal{HB} : the left port is *safe* and there is message 0 waiting for me; if seen by the left agent whose LSP is the \mathcal{HB} : the right port is *safe* and there is message 0 waiting for me

- other configurations do not occur

Note that although configuration $(1, 1, 0)$ has two possible interpretations, there is no ambiguity because this configuration will not occur when the \mathcal{HB} is the LSP of both agents. Please recall that at least one agent must have finished its assignment in order to leave a message.

The algorithm below is described for the right agent a_1 (using the first person of the singular). The algorithm for the left agent a_2 is almost identical. The only differences are using opposite directions, and using the floor function instead of the ceiling function when calculating the work assignment.

At the beginning of the algorithm, in what we will call the initialization step, the elimination technique is used to limit the number of active agents to two, as well as to choose the right agent a_1 and the left agent a_2 .

As previously explained, the procedure “Checking” is executed by an agent that detects that a message has been left for it. The right agent reads the message by traversing leftward until an *end-point marker* is found. Note that for the first message, there will be zero leftward moves. Moreover, a message is left only when both agents are active. Therefore there is no confusion if the *end-point marker* is found at the \mathcal{HB} (as explained above).

In procedure “Explore” an agent explores its work assignment using Cautious Walk, checking for a message from its partner agent.

Algorithm 4 Algorithm *Divide with Token* – — Procedure “Initialization” and “Explore”

```

1: INITIALIZATION:(upon initial wake-up in the  $\mathcal{HB}$ )
2: if the right port has no token on it then
3:   put two tokens on the right port
4:   execute procedure EXPLORE( $n - 1$ ) as the right agent  $a_1$ 
5: else if the left port has no token on it then
6:   move one token from the right port to the left port and add an additional token to
   the left port
7:   execute procedure SEEKING() as the left agent  $a_2$ 
8: else
9:   become Passive immediately
10: end if

11: procedure EXPLORE(Steps)
12:   while true do
   // might enter the BH in this step
13:     go from the current node  $u$  to its right neighbor  $v$ 
14:     return back to node  $u$  // doing the Cautious Walk here
15:     if there is a token on the right port of  $u$ 4 then // no message for me yet
16:       remove the token from the right port of  $u$ 
17:       move to  $v$ , put a token on the right port of  $v$  and decrement Steps.
18:       if Steps = 0 then
   // finished exploring my assignment, now find the other agent
19:         exit the loop and execute procedure SEEKING()
20:       end if
21:     else // there is no token on the right port, i.e. message waiting for me
22:       exit the loop and execute procedure CHECKING()
23:     end if
24:   end while
25: end procedure

```

In procedure “Seeking” the agent determines the distance between the LSPs and either locates the BH (if there is single unexplored node remaining) or leaves/updates the message for the other agent.

⁴Please note that here the number of tokens might have changed from 2 to 1 because u is the \mathcal{HB} and the second agent had waken-up meanwhile. But that still does not entail a message notification. Also, it is important to understand that here the code for the left agent is not simple mirror image of the code for the right agent. In fact, the corresponding test by the left agent will be: there is a token on the left port of u , or u is the \mathcal{HB} and there are two tokens on the left port of u .

Algorithm 5 Algorithm *Divide with Token* – — Procedure “Checking”, “Seeking” and “Check & Split”

```

1: procedure CHECKING
2:   go left until a token (end-point marker) is found in the middle of a node  $u$ , counting
   in  $Msg4Me$  the number of nodes traversed
3:   remove this token, return to your LSP and put the token on the right port
4:   for ( $i = 0; i \leq Msg4Me; i++$ ) do // compute the new work assignment
5:      $Steps = \lceil Steps/2 \rceil$  // again: we use floor in the case of the left agent
6:   end for
7:   execute EXPLORE( $Steps$ )
8: end procedure

9: procedure SEEKING
10:  go left until a token is found at node  $u$ , counting in  $DistC$  the distance traveled
11:  if found a token on the left port of a non- $\mathcal{HB}$  node, or two tokens on the left port
   of the  $\mathcal{HB}$  then
   //  $u$  is the LSP of the other agent, leave a message 0
12:    move a token from the left port to the middle of  $u$ 
13:    execute CHECK&SPLIT( $DistC$ )
14:  else if found one token in the middle of a node  $u$  then
   // this is the end-point marker of my last message, agent  $a_2$  did not read it yet
   // update/increment the message
15:    move the end-point marker one node to the right and increment  $MsgLeft$ 
16:    execute CHECK&SPLIT( $DistC + MsgLeft - 1$ )
17:  else if found one token on the left port of the  $\mathcal{HB}$  then //  $\mathcal{HB}$ , ignore
18:    ignore and continue on Line 19 as if nothing found
19:  end if
20: end procedure

21: procedure CHECK&SPLIT( $Dist$ )
22:  if  $Dist = n - 2$  then // single unexplored node remaining
23:    become DONE, the BH is in the remaining unexplored node
24:  else
25:    return to your LSP and execute Explore( $\lceil (n - Dist)/2 \rceil$ )
26:  end if
27: end procedure

```

Correctness and Complexity Analysis

Lemma 8 *At most two agents will become active.*

Proof: See Lemma 3

□

Lemma 9 *If a message is left for an agent, that agent detects the presence of that message and correctly computes its contents.*

Proof: We prove the lemma for the agent a_1 exploring to the right (i.e., for the right agent). The proof for the left agent a_2 is analogous. When a_1 finds out that there is a message waiting for it, by not finding its token when returning from *CWWT*, Lines 13 – 14 of procedure “Explore”, according to Line 11 of procedure “Checking” it travels to the left to locate the *end-point marker*, that is, the token placed in the middle of a node. As explained for in algorithm *Divide with Token +*, a_1 cannot overtake agent a_2 , even if a_2 is concurrently incrementing its message. Moreover, the searching for the *end-point marker* is *safe*,: a_1 will not travel past a_2 's LSP. Also, a non-zero message is left only by an agent that already explored its assignment. Thus, the distance between LSPs must be at least $\lfloor n/2 \rfloor$.

□

Let us define the *work assignment* of the right/left agent as follows:

- If the agent is traversing the *dangerous* link from its LSP and there is a message waiting for it, the work assignment of the agent is only the node on the other side of the *dangerous* port
- Otherwise, the work assignment of the agent is the *Steps* nodes to the right/left of agent's LSP.

Lemma 10 *At any moment, the work assignments of the right and left agents are disjoint. Moreover, if there is no message waiting for an agent then the work assignments form a partition of the part of the ring delimited by the LSPs (or the LSP of the right agent and the \mathcal{HB} , if there is single agent active) and not containing the \mathcal{HB} .*

Proof: By induction over the execution. According to Line 4 in procedure “Initialization”, the initial work assignment of the right agent a_1 covers all nodes between the \mathcal{HB} and a_1 ’s LSP (which is the \mathcal{HB}). This property is maintained by construction of procedure “Explore”, until the second agent a_2 leaves a message for a_1 .

If there is a message waiting for an agent (say a_1), the agent a_2 at the moment it left the message computed its work assignment as half of the part remaining *unexplored* between the LSPs. Since this part contains at least two nodes (otherwise a_2 would terminate), half of it (= b ’s assignment) does not contain the node a_1 is currently heading to.

Finally, if there are no messages waiting, the execution of the algorithm is either at the beginning when there is only one agent (a case we have already addressed above) or after one agent (say a_1) read a message M and recomputed its *Steps* according to Lines 13 – 14 of procedure “Checking”. Consider the value d of variable *DistC* of agent a_2 at the moment when it left the first message (of value 0) for agent a_1 . Since a_2 has just finished its assignment, by our induction hypothesis, the current value of a_1 ’s *Steps* equals to $n - d$ (i.e. the initial value of a_2 divided by 2). At the moment a_1 reads the message M , a_2 has halved its *Steps* $M + 1$ times (Line 37 in procedure “Check & Split”), and that is exactly what a_1 does in Lines 13 – 14 of procedure “Checking” (also applying Lemma 9). The partitioning works properly even when halving odd-sized workloads because one agent uses ceiling and the other uses floor. Afterwards, the invariant of the lemma is maintained by construction of *Explore* until another message is left.

□

Since, by construction, the only previously *unexplored* nodes an agent enters are in its work assignment, we immediately get:

Corollary 1 *At most one agent enters the BH.*

Another consequence of Lemma 10 is that, at any time, there is at most one message present in the ring, and at most one agent executing procedure “Seeking” or “Check & Split”.

From construction (Line 37 of procedure “Check & Split” and Lines 13 – 16 or CHECKING), it follows that the parameter STEPS is at least halved in each consecutive call to procedure “Explore” (i.e. the *unexplored region* is at least halved). Since no waiting is specified in any place of the algorithm, procedure “Seeking”, “Check & Split” and “Checking” each take at most $n - 2$ moves and either terminate or are followed by a call to EXPLORE. Consequently, the algorithm terminates in $O(n \log n)$ steps. And because the only way to terminate is to detect there is a single *unexplored* node, this node must contain the BH.

Most importantly, note that, at any time, at most three tokens are present in the ring: the one remaining at the \mathcal{HB} and one used by each agent for marking a *dangerous* port and for messaging. This follows from:

- In Line 3 of procedure “Initialization”, the first agent places two tokens as it wakes up. The second agent adds one more token (Line 6) and no more agents become active (Line 8)
- Nowhere else in the execution of this algorithm are new tokens introduced. That is, when an agent places a token, it is a token that this agent is reusing. This observation holds when an agent moves a token to the next port to explore, when it steals the token of its partner to leave a message (reusing the stolen token as *end-point marker*), when it increments a message (by moving the *end-point marker*), and when it reads a message (by recuperating its stolen token).

Putting all the observations together, and following the lower bound from the whiteboard model presented in [45], we conclude:

Theorem 3 *Algorithm Divide with Token – correctly locates the BH in a ring with n nodes, employing two co-located agents and three (3) tokens in total, using $O(n \log n)$ moves.*

3.3 BH Search in an Oriented Ring by Scattered Agents

3.3.1 Basic Observation

We have already observed that communication in an algorithm that uses tokens is less straightforward to understand than in a model relying on whiteboards. As we will see below, the use of scattered agents instead of co-located agents complicates communication further.

Let us start with a few observations/assumptions:

- The agents start from different nodes, that is, from different homebases \mathcal{H}_s .
- The agents agree on the same sense of direction, that is, all the agents agree on the same orientation (i.e. right and left, or clockwise and counterclockwise).
- No agent knows the location of the other agents.
- The agents may or may not know the team size k (that is, the total number of agents). We consider both situations later in this section.

3.3.2 Algorithm *Gather Divide*

Basic Ideas and General Description

Before attempting to generalize our results, in this subsection, we study BHS under the following conditions:

- The total number of agents, k , is known.
- The orientation of the ring is known.

- One token per agent is available.

We show that, the number of moves is $kn + n \log n$ using the algorithm *Gather Divide* we now introduce.

An intuitive solution for locating the BH using scattered mobile agents is to somehow have two of the surviving agents eventually form a pair of agents that use a same node as \mathcal{HB} , and then simply have these two paired agents use algorithm *Divide with Token* - to find the BH. We think of two paired agents that execute algorithm *Divide with Token* – (i.e., the optimal solution we obtain for BHS in ring topology with co-located agents) as having (conceptually) gathered in a same \mathcal{HB} .

The gathering process works as follows. When an agent wakes up, it goes right (since there is orientation) to explore, using *CWWT* (one token on a port is used as *CWWT* token). As soon as it meets another agent that is exploring “in front of it”, it becomes a “follower” of that agent: it follows this explorer wherever the latter goes (see details below). In particular, a follower moves its token to the middle of the node it is currently visiting in order to signal its presence. Clearly, a explorer may die in the BH. When a follower is the first agent to arrive at a node in which there are $k - 2$ followers, it will start algorithm *Divide with Token* – [49] acting as a “right paired agent”, but only after all the other followers at that node become *Passive*. After this, the sole explorer will start executing *Divide with Token* – as a left paired agent immediately, if it did not die in the BH.

Detailed Description

The sketchy description given above can be formalized as follows (referring explicitly to the agent executing this algorithm as “the agent”):

Algorithm 6 Algorithm *Gather Divide*

- 1: The agent wakes up, then explores to the right (using *CWWT*) until it finds a token in the node it enters, at which point the agent becomes a follower.
 - 2: As a follower, the agent leaves a token in the middle of the current node and then waits to continue walking until the token on the right port of the current node disappears.
 - 3: **if** upon arrival at a node, there are the tokens of $k - 2$ agents in the middle of that node **then**
 - 4: The agent becomes the Left Paired Agent, and leaves a token on the left port of this node. It then waits until either A or B happens:
 - 5: A: there is no token in the middle of the current node and there is a token on the right port; B: there is no token in the middle of the current node and there is no token on the right port
 - 6: **if** A happens **then**
 - 7: The agent picks up the token on the right port then leaves it in the middle of the current node
 - 8: **else if** B happens **then**
 - 9: The agent picks up the token on the left port and immediately starts algorithm *Divide with Token* – as the Left Paired Agent.
 - 10: **end if**
 - 11: **else**
 - 12: The agent erases its token then becomes *Passive* as soon as a token appears on the left port of the current node.
 - 13: **end if**
 - 14: **if** When, as a right explorer, the agent comes back to its last-safe-place and sees that there is a token on the left port or that its own token was moved to the middle **then**
 - 15: This agent becomes the Right Paired Agent and immediately starts algorithm *Divide with Token* –
 - 16: **end if**
-

Analysis

Most importantly, because there is orientation in the ring, all agents share a common understanding of “right” and “left”. This is critical to ensure all scattered agents start exploring to the right. In turn, because all scattered agents explore in the same direction, at most one agent will die in the BH before the Left and Right Paired Agents are identified. This stems from having followers have to wait for their explorer to come back from a *dangerous* link before these followers can follow.

Furthermore, it is known that during the execution of algorithm *Divide with Token* –, at most one of the two active agents dies in the BH. Consequently, it follows that:

Corollary 2 *At most two agents enters the BH.*

Theorem 4 *Using k ($k > 2$) scattered agents, one token per agent, after $O(kn + n \log n)$ moves, algorithm *Gather Divide* correctly locates the BH in an oriented ring with n nodes.*

Proof: For determining the Left and Right Paired agents, it is key to understand that the worst case is to have k agents travel all the ring except for the BH, which is $O(kn)$. Then we already know that algorithm *Divide with Token* – correctly locates the BH using $O(n \log n)$ moves. It follows that algorithm *Gather Divide* correctly locates the BH in $O(kn + n \log n)$ moves.

□

The question then is: can we do better? The answer is yes, as explained next.

3.3.3 Algorithm *Pair Elimination*

Introduction

In this next algorithm, which we call algorithm *Pair Elimination*, we achieve locating the BH using $O(n \log n)$ moves in total and four (4) tokens per agent (still using

anonymous agents and nodes). This is a significant improvement over algorithm *Gather Divide* [53], costing only 3 more tokens per agent.

Interestingly, we remark that the algorithm *Pair Elimination* also solves the Leader Election problem and the Rendezvous [3, 4, 11, 12, 36, 58, 112] problem despite the presence of a BH. We will elaborate on this later.

Finally, in the algorithm *Pair Elimination* as in Algorithm *Gather Divide*, all agents have the same behavior (i.e., follow the same algorithm), but start at different nodes. Also, agents may start at different and unpredictable times. The \mathcal{HB} s are marked before the execution of the algorithm starts.

General Description

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

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

The \mathcal{HB} of an agent is identified by having one token in the middle of that node. An agent a_i starts exploring the ring from its \mathcal{HB} , moving to the right until it becomes part of a pair or dies in the BH. Its *explored region* is a segment with \mathcal{HB} as one of

the two end nodes. Following terminology of this chapter, we call the other end node of a_i 's *explored region* the *LSP (Last Safe Place)* of a_i .

A node u , in which two agents are paired, is called the *BP (Birth Place)* of that pair. Paired agents explore the ring in opposite directions. When a pair is formed, there are two *LSPs*: the two end nodes of the union of *explored region* of the two paired agents.

The *Birth Place* of a pair is identified by having three or more tokens in its middle.

Each Birth Place has a level. In particular, if it has $i + 3$ tokens in the middle of the node, then that node is identified as a *i-BP (i-Level Birth Place)*. An *BP* starts being a *0-level-BP*. The level of a *BP* is also the level of its pair. Also, we will show that the highest level of a pair is at most $\log n$.

A *BP* is called a *CP (Crown Place)* if at least one of the agents of its pair has finished exploring half of the ring size ($\lfloor (n - 1)/2 \rfloor$). A Crown Place *CP* is identified by having two tokens in the Middle. When a *BP* becomes crowned, its level becomes higher than any other (with the exception that its level will be the same as the other *CP*, if ever such a second *CP* is set, as will be explained later. As we will prove in Subsection 3.3.3, there will be maximum 2 *CPs* during each algorithm execution.) The agents of a *CP* are referred to as *crowned agents*, and have their level set so that they can eliminate all non-crowned agents. We will also use the expression *crowned pair*. (How one *CP* eliminates the other is explained later. The key idea is that, once down to a single *CP*, we can reuse algorithm *Divide with Tokens*.)

The priority relationship is shown in Figure 3.1.

Level(1) Left Pair	Level(1) Right Pair	Level(2) Left Pair	Level(logn) Left Pair	Level(logn) Right Pair	Crowned Pair Left	Crowned Pair Right
Low level Low priority					→	High level High priority	

Figure 3.1: Pair levels table

Let us now describe the algorithm:

Let a_1 meet a single agent a_2 going to the right. Note that, since agents do not see each other, this situation is detected by a_1 finding tokens in some predefined position. In this case, a_1 leaves a message for a_2 and becomes a left paired agent. When a_2 sees the message from a_1 , it becomes a right paired agent. As usual, since there are no whiteboard to write messages on, this communication is done only moving and placing tokens according to predefined rules.

Because of the complexity of this algorithm with respect to the use of tokens, we reuse ideas introduced earlier in the chapter but introduce an improved terminology. In this algorithm, an agent leaves a (*Message Notice*) — *MN* for its (paired) partner by stealing the token in this partner's LSP and placing it on the port of a node between the two LSPs. Once so placed, the token becomes a *Message Sign (MS)* for its partner. As usual, the distance between the *MS* and the partner's LSP represents the information the agent wants to deliver to its partner.

After a pair is formed, the two paired agents start exploring the ring in opposite directions. They keep exploring a pre-defined number of nodes (initially, $\lfloor (n-1)/2 \rfloor$).

If one finishes its work assignment, as in previous algorithms, it goes to seek its partner, and calculates the unexplored region according to the location of its partner. It leaves a message notice and a message sign to the partner then goes back to its LSP. We say this paired agent finished a *stage*.

In general, the following chain of actions by a paired agent a_1 constitutes one *stage*: it explores a pre-calculated number of nodes, it checks the location of the partner, it leaves a message for it, and then goes back to its LSP. The information in the message left for its partner is, as usual, the number of times a_1 finished exploring its pre-calculated portion of the ring. This information is used by a paired agent to calculate the number of nodes to visit in the next stage. When a paired agent a_2

detects a message notice, it goes to check the message, and uses it to calculate the number of nodes it needs to explore in the next stage.

If a paired agent goes into the *BP* of a lower level pair, it terminates the pair of agents corresponding to that *BP* by removing all the tokens in that *BP*. It then goes back to its LSP and continues exploring.

When a left paired agent arrives at the *BP* of a same level pair, it terminates the corresponding pair by removing all the tokens in their *BP*. It then goes to its BP to increase its pair level, and goes back to its LSP and continues exploring.

When a left paired agent goes into a *BP* of a higher level pair, it becomes *Passive* immediately.

When a right paired agent a_2 arrives at the *BP* of a higher or same level pair, it goes back to its *BP*. If there is no token there, then a_2 becomes *Passive*. Otherwise, a_2 goes to the *BP* of the pair with the same level on the right to terminate that pair.

At any time, if a paired agent a_1 encounters a single agent a_3 , a_1 terminates a_3 by stealing its token.

We can now proceed with a more procedural presentation of the algorithm. We use the point of view of a right explorer (RE for brevity) to describe the algorithm. Most of the procedure for left explorers (LEs for brevity) can be achieved by changing the words “right” into “left” and vice versa in the procedures for the REs. The only procedure in which LEs and REs behave differently is Right (Left) Exploring. We will explain this in subsection 3.4.2.

Finally, in the algorithm:

- parameter *BPdist* is used for an explorer to remember its *BP*
- *EDist* records the number of explored nodes between the two LSPs
- *steps* records the number of steps one explorer needs to explore in each stage
- *message* is used by an explorer to remember the message the partner left for it.

Procedure “Form Pair”

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

- A: it finds two or more tokens in the middle of a node. In this case, the node is either a pair’s *i-BP* ($3+i$ tokens in the middle), or a crowned place (two tokens in the middle).
- B: it finds one token in the middle of a node for the second time. This means the agent knows there are at least two agents on its right side. Given these two agents both go right, and this sense of direction is common to both of them, eventually they will form a pair, even though one may already die in the BH. (Later we will prove that even in this case, BH still can be located.) So the agent can become *Passive* immediately.
- C: it finds a token on the right port of a node (which corresponds to meeting an agent walking with *CWWT* to the right). The action for this event is given in the pseudo code below.
- D: it finds out that there is no token in the port where it left its *CWWT* token, and there are no tokens in the center. It means that a paired agent (with a greater or equal level but to its right) terminated it.
- E: it finds out that there is no token in the port where it left its *CWWT* token, and there are two or more tokens in the middle of the node. It means it now forms a pair with another agent.

Algorithm 7 Algorithm *Pair Elimination* — Procedure “Initialization” and “Form Pair”

```

1: Initialization: Wake up; go to the right with CWWT,  $steps = 0$ ,  $BPdist = 0$ ,  $EDist = 0$ 
2: procedure FORM PAIR
3:   keeps walking right using CWWT until A, B, C, D or E happens
4:   if A, B or D happens then
5:     becomes Passive
6:   else if C happens then
7:     becomes a Left Paired agent, moves the token from the right port to the middle
8:     leaves two extra tokens in the middle as a BP mark. Then executes LEFT
    EXPLORING( $steps$ ,  $EDist$ )
9:   else if E happens then
10:    becomes a Right Paired Agent and executes RIGHT EXPLORING( $steps$ ,  $BPdist$ ,
     $EDist$ )
11:   end if
12: end procedure

```

Procedure “Right Exploring”

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

- A: it goes into a node with at least three tokens, but fewer tokens (in the middle) than its level. This is a *BP* that has a lower level than the one of RE.
- B: it goes into a node with at least three tokens, the number of tokens (in the middle) being greater or equal to its level. This is a *BP* of equal or greater level than the one of RE.
- C: it finds no token on the port where it left its *CWWT* token. That is, its token has been stolen.
- D: it finishes exploring the last of its assigned nodes. The number of nodes to explore is given by: $\lfloor (n - EDist)/2 \rfloor$.
- E: it finds a token on the right port of a node.

- F: it determines $EDist = n - 2$. This means the *explored region* contains $n - 2$ nodes.

Algorithm 8 Algorithm *Pair Elimination* — Procedure “Right Exploring”

```

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

```

Procedure “Left Exploring”

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

- A: it goes into a node with at least three tokens, but fewer tokens (in the middle)

than its level. This is a *BP* that has a lower level than the one of the RE.

- B: it goes into a node with at a number of tokens (in the middle) equal to its level. This is a *BP* that has the same level than the one of the RE.

- C: it goes into a node with at a number of tokens (in the middle) greater than its level. This is a *BP* that has a greater level than the one of the RE.

- D: it finds *CWWT* token was stolen in its LSP.

- E: it finishes the pre-calculated portion of ring assigned to it. This pre-calculated part could be $\lfloor (n - EDist)/2 \rfloor$ after “seeking” procedure, or $\lfloor steps/2^{message} \rfloor$ (it needs to half the *unexplored region message* times according to the message that the partner left to it) after procedure “Checking”. Here *UEDist* is the number of nodes left unexplored in each stage.

- F: it finds a token on the left port of a node.

- G: it determines $EDist = n - 2$.

Algorithm 9 Algorithm *Pair Elimination* — Procedure “Left Exploring”

```

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

```

Procedure “Checking”

Procedure “Checking” is executed when the *CWWT* token of a paired agent is stolen. It can mean either it got eliminated by a higher level pair, or received a *Message Notice* from the partner. But it cannot determine which case it is until later. In the first case, this agent will go back to its *BP* and realize all the tokens (in the middle) disappeared. In the second case, the agent will find a token on a port (it is a right port, if it is a right paired agent executing procedure “Checking”; it is a left port, otherwise). This token is a *Message Sign*. Also, this *Message Sign* could be in its *BP*. The agent then calculates the number of nodes to explore in the next stage, using the

message it just collected. As in previous algorithms, the number of steps between an explorer's LSP and the MS is the number of stages the partner finished.

Algorithm 10 Algorithm *Pair Elimination* — Procedure “Checking”

```

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

```

Procedure “Seeking”

Procedure “Seeking” is used for an explorer to seek its partner. When a paired agent executes procedure “Seeking”, it means it finished exploring the number of nodes assigned to it at this stage. The number of nodes an agent needs to explore in the very first stage is $\lfloor (n-1)/2 \rfloor$. Recall, a pair is called a *crowned pair* and a BP is called CP as soon as one of the paired agent explored $\lfloor (n-1)/2 \rfloor$ nodes in the ring. We observe that when a paired agent executes procedure “Seeking” for the first time it implies this pair reached crowned level. This agent goes back to its BP to make it a *crowned place*. If there is no token in this agent's BP , it means this pair was eliminated by another pair earlier. It then becomes *Passive* immediately. If there are more than 3 agents in the middle of the node, this left paired agent then picks all but 2 tokens in order to crown the pair. It then execute procedure “Seeking” to the right. If there are 3 tokens in the middle of the BP , this means the right paired agent has not come back from its $CWWT$ walk as a single agent. So, the left paired agent

does not need to go to leave a message for the partner.

Algorithm 11 Algorithm *Pair Elimination* — Procedure “Seeking”

```

1: procedure SEEKING — RIGHT PAIR( $EDist, BPdist$ )
2:   keep walking to the left and increasing  $EDist$  until back at the  $BP$ 
3:   if there is no token in its  $BP$  then
4:     become Passive
5:   else if there are more than two tokens in the middle then
6:     pick up all but two tokens // crown the pair
7:   end if
8:   if there are 3 tokens then
9:     keep walking to the right until go back to its LSP
10:    execute RIGHT EXPLORING( $steps, BPdist, EDist$ ) with  $\lfloor (n - EDist)/2 \rfloor$  new
    steps.
11:   else
12:     keep going to the left, keep increasing  $EDist$  until there is a token on the left
    port // the left pair's LSP or MS
13:     if  $EDist = n - 2$  then
14:       becomes DONE
15:     else
16:       pick up the token, move to the right neighbor, put a token on the left port.
17:       keep walking to the right until go back to its LSP
18:       execute RIGHT EXPLORING( $steps, BPdist, EDist$ ) with  $\lfloor (n - EDist)/2 \rfloor$ 
    new steps.
19:     end if
20:   end if
21: end procedure

```

Analysis to Algorithm *Pair Elimination* — Correctness

First, observe that pairs will indeed be formed:

Lemma 11 *At least one pair is formed.*

Proof: There are at least two agents in the ring and the orientation of the ring is known. Upon the algorithm starts, all the agents go right. Sooner or later, one will catch up with another, then they form a pair. When there are only two agents a_1 and a_2 , if a_1 dies in the BH, a_2 will eventually see the token a_1 left before it died. a_2 will

still form a pair with a_1 (assuming a_1 is alive) by moving the token into the middle and adding two more tokens.

□

Lemma 12 *Let a_1 and a_2 be the only 2 agents in the ring. If a_1 dies in the BH before a_1 and a_2 forms a pair, a_2 can still locate the BH correctly.*

Proof: According to procedure “Form Pair”:

- Before a_1 disappeared in the BH, it left one token (given a_1 is a single agent) on the right port of the node u , which is to the left of the BH.
- a_2 keeps going to the right until it encounters the last token a_1 left in u before it died in the BH. a_2 then moves the token into the middle of u and adds two more tokens.

According to procedure “Left explorer”:

a_2 keeps exploring to the left until it realizes that $\lfloor (n-1)/2 \rfloor$ nodes were explored.

It then execute procedure “Seeking”.

According to procedure “Seeking”, a_2 goes back to u to update the level of the BP of a_1 and a_2 . When a_2 sees that there are 3 tokens in the middle of its BP , it goes back to its LSP instead of keep going right to try to leave a message for a_1 . This prevents a_2 from dying into the BH. Once a_2 goes back to its LSP, it will start exploring the ring in a new stage.

The above procedures will be repeated until sooner or later a_2 counted the *explored region* includes $n - 2$ links and $n - 1$ nodes during a “Seeking” procedure. This terminates the algorithm before a_2 dies into the BH. Hence, even if a_1 died in the BH before a_1 and a_2 forms a pair, a_2 can still locate the BH correctly.

□

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

Lemma 13 *Let x_i denote the number of pairs that reach level i . Then $2x_i \leq x_{i-1}$.*

Proof: Consider three consecutive level $i - 1$ pairs: A , B , and C ; and assume that B reaches level i . This can happen only if (see Figure 3.2): the right agent of pair B or the left agent of pair A have not reached the other pair's $(i - 1) - BP$ before the left agent of pair B or the right agent of pair C reached the other $(i - 1) - BP$, then pair C is eliminated by pair B . Otherwise, pair B will be eliminated by pair A . In general, taking any two neighboring pairs, at most one of them will survive and reach level i , the other one will be eliminated. Hence, the number of pairs that reached level i is at most half of the number of pairs that reached level $i - 1$.

□

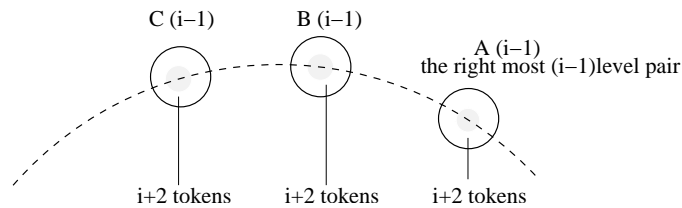


Figure 3.2: *The BPs of three consecutive pairs that reached level $i - 1$*

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

Lemma 14 *Let x_i denote the number of pairs that reach level i . Then $x_i \geq 1$.*

Proof: Because of the priority rules (i.e., higher level pairs can eliminate lower level pairs), the right most pair that reaches level $i - 1$ will eventually kill its “left neighboring” level $i - 1$ pair, and it will not be eliminated. So it will eventually reach level i .

□

By Lemma: 11, 13 and 14, the maximum possible level follows:

Corollary 3 *The maximum level reached is $\log n$.*

Proof: According to Lemma 13, the number of pairs that reached level i is at most half of the number of agents that reached level $i - 1$. Namely, in each level, at most half of the pairs will reach a higher level. From $\lfloor (n - 1)/2 \rfloor$ pairs in level 1 at the very beginning, when there is only one pair left, the maximum level of the pair is $\log n$.

□

Lemma 15 *Eventually at least one and at most two crowned pair(s) will be formed.*

Proof:

- If there is only one pair left at level i , then it will be *crowned*.
- For a pair to become *crowned*, one of its agents must have explored $\lfloor (n - 1)/2 \rfloor$ nodes and its *BP* must have not been reached by another pair.

It is a fact that if a paired agent reaches the *BP* of another pair, one of the two pairs will eventually be eliminated according to the elimination technique explained earlier.

Hence, we define a segment S with more than $\lfloor (n - 1)/2 \rfloor + 1$ nodes in between the *BPs* of a i -level-pair ($i < \log n$) and its neighbor pair, a j -level-pair ($j < \log n$), in order for a paired agent to have a chance to finish exploring $\lfloor (n - 1)/2 \rfloor$ nodes (in order to be *crowned*) without reaching the *BP* of the other.

Assume that there are two or more such non-overlapping segments S in the ring, in order to have three or more *crowned pairs*. It is clear that because

$(\lfloor (n - 1)/2 \rfloor + 1) * 2 \geq n$, the segments must overlap with each other. In this case, it is not true that there can be two or more such segments in the ring without overlapping. Hence there is maximum of one such segment contained in the ring.

For any highest level $(\log n)$ pairs (according Corollary 3) that have more than $\lfloor (n - 1)/2 \rfloor$ nodes in between its *BP* and the neighbor pair's *BP*, it has a chance of having one of the paired agent finish exploring $\lfloor (n - 1)/2 \rfloor$ nodes without reaching the *BP* of the other. As we know, if there are two segments with $\lfloor (n - 1)/2 \rfloor$ nodes in each, they will definitely overlap ($(\lfloor (n - 1)/2 \rfloor + 1) * 2 \geq n$). But this will no longer cause an increase in the level of the pair. In this case, there are maximum of one such segment between two *BP* contained in the ring. This means, there are maximum of two pairs have more than $\lfloor (n - 1)/2 \rfloor$ node between the two *BPs*. Hence, there can be at most two *crowned pairs*. See Figure 3.3

□

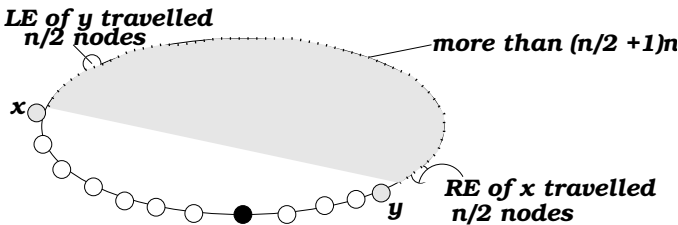


Figure 3.3: Two crowned pairs.

Theorem 5 *Algorithm Pair Elimination correctly locates the BH in an oriented ring with n nodes.*

Proof: By Lemma 15 there will be either one or two *crowned Pairs*. If there are two *crowned pairs*, since they have the same level, according to the priority rules, the

left *crowned pair* will be eliminated by the other; hence within finite time only one *crowned pair* (the right one) will remain.

Next, we observe that the rules of algorithm *Pair Elimination* for a *crowned pair* [49] are precisely the ones of algorithm *Divide with Token* – for a pair of co-located agents. This observation holds because the *BP* of the pair can be seen as the shared \mathcal{HB} of that pair. Hence, once a single crowned pair is set, the correctness of the test of the algorithm is the same as the correctness of algorithm *Divide with Tokens* –.

□

Analysis to Algorithm *Pair Elimination* — Complexity

Theorem 6 *Algorithm Pair Elimination correctly locates the BH within $\Theta(n \log n)$ moves in a ring with n nodes.*

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

- By single agents: the worst case is that there is one agent per node except for the BH. They all start doing *CWWT* to the right. Because of the pair forming procedure, some of the agents' *CWWT* may be interrupted. Consequently, for those, the number of moves before a an agent is paired will be reduced from 3 to 2. But the worst case is still 3 moves per *CWWT*. So, there are $3(n - 1)$ moves in total if there are $n - 1$ single agents.
- By all paired agents of level i : each paired agent explores the ring using *CWWT*. It goes back to the *BP* to either increase the level of this pair after a left paired

agent having gone into the *BP* of another pair that is in the same level, or check the current level of the pair after a right paired agent having gone into the *BP* of another pair that is in the same or higher level. This agent then either becomes *Passive* (if a right paired agent realized that its pair's current level is still the same or lower than the level of the *BP* it just visited) or goes back to its LSP. So the total number of moves with $n - 1$ *i - level* paired agents, is $(3 + 1 + 1) * (n - 1)$. According to Corollary 3, there are at most $\log n$ levels of each agent. Hence, the total number of moves by all paired agents is $5(n - 1) \log n$.

- By *crowned agents*: There are at most two *crowned pairs*, and the algorithm has the right *crowned pair* win over the left *crowned pair*. The number of moves required for the right *crowned pair* to eliminate the left *crowned pair* is constant. Eventually there is one *crowned pair* left, and its agents share a same *crowned place*. Consequently, these *crowned agents* can locate the BH according to Algorithm *Divide with Token -*, and we already know this requires $O(n \log n)$ moves. The lower bound follows from [45].

Hence, the total cost for locating a BH in an oriented ring with two or more scattered agents is $\Theta(n \log n)$ moves.

□

Theorem 7 *Algorithm Pair Elimination correctly locates the BH with two or more agents, four (4) tokens per agent in a ring with n nodes.*

Proof:

- One token is needed to mark the \mathcal{HB} .
- One token is needed for *CWWT*.

- When forming a pair, for marking a *BP*, an agent steals one token from its partner, then puts two more tokens in order to have three tokens in the middle of their *BP*. So two more tokens are needed.
- Each pair will not increase its level unless it first eliminates another pair. A pair that eliminates another, picks up the tokens in the *BP* of the other pair. Finally, that pair uses one token to increase its level in its *BP*. Consequently, no extra token is required for pair elimination and for increasing the level of a pair.
- When an agent crowns its *BP*, it will then need to leave a message for its partner. Crowning a *BP* requires only two tokens in the middle of that node (which necessarily had 3 or more tokens in its middle). Thus the agent gains one or more tokens. Clearly, one of these tokens can be reused to leave the message for the partner.
- Finally, as previously explained, leaving and checking a signal requires a token, but this token is stolen from the partner agent. So such communication does not require any extra tokens.
- Thus, we can conclude that, in this algorithm, the total number of tokens each agent needs is 4.

□

Leader Election and Rendezvous Problem

In distributed computing, leader election is one of the most often used solutions to solve several recurring problems. Whether used as a solution for simplifying many complex distributed computing problems, or because of the nature of the problem itself, the idea of selecting a single coordinator from a population of autonomous

symmetric entities plays a crucial role in distributed computing. The task of selecting a single coordinator is known as the problem of *Leader Election*. Formally, the task consists in moving the system from an initial configuration where all entities are in the same state (usually called available) into a final configuration, where all entities are in the same state (traditionally called follower), except one that is in a different state (traditionally called leader). There is no restriction on the number of entities that can start the computation, nor on which entity should become leader [100, 102].

The rendezvous search problem for mobile agents is a search optimization problem based on the following question: How should mobile agents move along the n nodes of a network in order to minimize the time required to meet or rendezvous? [3, 4, 11, 12, 36, 58, 112]

Theorem 8 *Algorithm Pair Elimination solves the problem of electing a Leader (or Leader Election) among the scattered agents, in spite of the presence of a BH. This algorithm also solves, with the same cost, the Rendezvous problem in the ring topology despite the presence of a BH.*

Proof: From the following two facts:

- Lemma 3.3.3 illustrated that eventually at least one and at most two *crowned pair(s)* will be formed.
- The orientation of the ring is known.

We know that eventually there is only one *Crowned Place* left in the ring.

First, let the right *CP* be the leader node, and the right paired agent of this *crowned pair* be the leader agent. We conclude that a *Leader* (agent or a node) can be *elected* among the scattered agents on an anonymous ring, in spite of the presence of a BH.

Second, once the BH is located (algorithm *Pair Elimination* terminated), the right paired agent of the right *crowned pair* becomes the leader agent. Let the leader agent go to collect all the other surviving agents to go to any of the non-BH nodes as desired.

Hence, algorithm *Pair Elimination* solves the problem of *electing a leader* among the scattered agents, in spite of the presence of a BH. It also solves with the same cost the *Rendezvous* problem despite the presence of a BH.

□

3.4 BH Search in an Unoriented Ring by Scattered Agents

3.4.1 Introduction

As discussed in 3.3.1, BHS is more complicated and demanding using scattered agents than with co-located agents. The situation becomes even more complicated when the ring is unoriented, that is, when there is no common sense of direction, no agreement on what is left and right.

In this section, we first introduce two algorithms that solve BHS without relying on *CWWT*:

- *Shadow check without CWWT* that requires a minimum of 3 agents (which is optimal), $O(n^2)$ moves and 5 tokens per agent.
- *Modified ‘Shadow check without CWWT’* that can locate the BH with $O(n \log n)$ moves, which is optimal, with only 1 more agent being used than with algorithm *Shadow check without CWWT*.

We then briefly sketch out the algorithm *Shadow Check*, which solves the BHS problem with as low as 3 agents, 1 token per agent, and n^2 moves in total. (It works in both oriented and unoriented rings.)

These three algorithms demonstrate that locating the Black Hole in an anonymous ring network using tokens is feasible even if the agents are scattered and the orientation of the ring is unknown. In turn, this leads us to conclude that, for BHS in a ring, the token model offers solutions that are as efficient as those developed for the whiteboard model, even if both agents and nodes are *anonymous*.

Also, we observe that:

- It is possible to solve BHS with scattered agents in an unoriented ring even without imposing the FIFO requirement and without using *CWWT*.
- Ultimately, when choosing a solution to BHS, one has a choice between different members of a family of algorithms, each member ideally optimizing one (or more) performance facets of this problem. For example, algorithm *Shadow check without CWWT* optimizes the number of agents, whereas *Shadow check with CWWT* optimizes the number of agents and the number of tokens used. Conversely, *Modified Shadow check without CWWT* optimizes the number of moves.

3.4.2 Algorithm *Shadow Check without CWWT*

Basic Ideas and General Description

We call a node/link *explored* if it is visited by an agent. As usual, an *explored region* consists of contiguous explored nodes and links, and the last node an agent explored its Last-Safe-Place (LSP for brevity). In the co-located agents case, as soon as the algorithm starts, an *explored region* is created. Throughout the execution, there is one and only one *explored region* and this region keeps growing larger until includes $n - 2$ edges and $n - 1$ nodes in it. In contrast, in the case of scattered agents, there are more than one *explored regions* in the ring. Our goal is to merge all the *explored*

regions into one, which eventually includes all the nodes and links with the exception of the BH and the two links leading to the BH. Let us describe how this goal is going to be achieved.

Upon waking up, an agent becomes a *Junior Explorer (JE)*, exploring the ring to the right (from the viewpoint of that agent) until it sees another agent⁵. We say two agents *meet* when at least one of the two agents sees the token(s) of the other agent. When two agents walking in different directions meet, we say they come “face to face”. When two agents walking in the same directions meet, we say one “catches up with” the other. When two *JEs meet*, they both become *Senior Explorers (SE)*, and start exploring the ring in opposite directions. We call the explored area between these two SEs a *safe region*. A SE explores the ring, growing its *safe region* and checking after each newly explored node whether the *safe region* contains $n - 1$ nodes (i.e., all but the BH). When two SEs moving in opposite directions *meet*, their two *safe regions* merge into a bigger *safe region*. The two meeting SEs become *checkers* and check the size of the new *safe region*. There could be more than one such *safe region*. When a *JE* detects a *safe region* (by encountering a SE), it becomes *Passive*.

When there is no cycle interruption (see below), each SE repeats the following cycle: it leaves two tokens on the port (if there is no token on this port) of the unexplored link on which it is going to move next. Once it reaches the node (if it is not the BH), the SE leaves two tokens on the port from which it did not enter that node. It returns to the previous node, picks up the token(s) on the port it used as *CWWT* tokens, then returns to the last explored node.

If, between cycles, an agent notices any unusual event (e.g., token situation change on certain ports of a node), it stops the cycle and acts according to this interruption.

⁵More precisely, finds a token of another agent

Using Tokens for Communication and coordination

- One token on a port means a JE is exploring the link via this port.
- Two tokens on a port means a SE is exploring the link via this port.
- One token on a port and one token in the middle of a node means this is the node where two agents working in opposite directions meet.
- One token on each port of a node means this is a node where one agent catches up with another agent working in the same direction.

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

Procedure “Initialization” and “Junior Explorer”

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

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

- Case 1

The JE, we call it a_1 , puts 1 token in the middle of node u , then goes back to

the left node. If a SE caught up with a_1 , then a_1 becomes a Checker to its left. If the agent it just met, let's call it a_2 , in the opposite direction, also left a_1 a sign (i.e., a token in the middle of u), then a_1 will become a SE to the left. If another JE a_3 catches up with a_1 , a_1 will pick up all the tokens in the current node, then become a Checker to the right.

- Case 2

The JE a_1 goes back to the left node. If another JE catches up to a_1 , then a_1 will become a checker to its right. If a_1 notices that the SE it just met in the opposite direction left a_1 a sign (i.e., a token in the middle), then a_1 will immediately become *Passive*. If a JE met with a_1 "face to face" and left a_1 a sign before a SE caught up to that JE, then a_1 will become a SE to the left.

- Case 3

The agent a_1 puts one token on the left port, then goes back to its left node. If a_1 's token is still there, it will move this token to the left port, add one more token on the left port, and then become a SE to its left. If either a_1 sees the sign a SE walking to its left direction left to it, or another JE caught up to a_1 , it will become *Passive* immediately.

- Case 4

The agent a_1 goes back to its left node. If a_1 's token is still there, then it will pick the token then become *Passive*. If a SE caught up to a_1 , then it will become *Passive*. If another JE caught up to a_1 , it will pick up the tokens, then become a Checker to the right.

- Case 5

The agent a_1 returns to the left node. If a SE caught up to a_1 , then it become *Passive*. If another JE caught up TO a_1 , it will become a checker to the right. If it notices that the JE it just met left it a sign (i.e., a token in the middle),

then it will move its 2 tokens to the left port and become a SE to the left.

- Case 6

The agent a_1 puts a token on the right port of n then goes back to the left node.

If a_1 's token is still there, then a_1 will pick the token and continue as a JE. If a SE caught up to it, then it will become *Passive*. If another JE caught up to a_1 , then it will become a SE to the right.

Procedure *Checker*

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

Procedure *Senior Explorer*

A senior explorer s_1 will eventually either end up in the BH or locate the BH, or become a Checker upon *meeting* another SE or a potential SE. SE s_1 is taken to walk to its right node. If it *meets* another SE in the different direction, that is, if they come “face to face”, s_1 will pick up all the tokens in the current node and become a Checker to the right. If s_1 realizes it is the node where two JEs working in different directions met, it will then become a Checker to the right. If s_1 realizes this node is where two JEs working in the same direction met, it will then go back to the left

port, pick up all the tokens and become a Checker. If it meets a JE going to the left, then it will pick the token on the left port, put two tokens on the right port, go back to the left node and pick up the two tokens on the right port. Then s_1 will execute the *check phase* to the left. If it meets a JE going to the right, then it will put one more token on the right port, go back to the left node, pick up the two tokens on the right port, then execute the *check phase* to the left. If the node is empty, s_1 will then put two tokens on the right port, go back to the left node, pick up the two tokens on the right port, then execute the *check phase* to the left.

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

Pseudo Code

A JE will eventually either end up in the BH or become a *Checker* upon *meeting* a SE or a potential SE, or become a SE upon *meeting* another JE. The pseudo code of procedure “Initialization” and “Junior Explorer” are in Algorithm 12, 13, 14, 15.

A checker is created when an agent realizes it is in the middle of two SEs exploring in different directions. The purpose of the checker is to check the distance between the two SEs. If the distance is $n - 2$, that means that two agents died in the BH, and the only node left is the BH. Otherwise, this checker becomes *Passive*. The pseudo code of procedure “Checker” is in Algorithm 16

Algorithm 12 Algorithm *Shadow Check without CWWT* — Procedure “Initialization” and “Junior Explorer”

```

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

```

Algorithm 13 Algorithm *Shadow Check without CWWT* — Procedure “Junior Explorer” — Cases 1 and 2

```

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

```

Algorithm 14 Algorithm *Shadow Check without CWWT* — Procedure “Junior Explorer” — Cases 3 and 4

```

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

```

Algorithm 15 Algorithm *Shadow Check without CWWT* — Procedure “Junior Explorer” — Cases 5 and 6

```

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

```

Algorithm 16 Algorithm *Shadow Check without CWWT* — Procedure “Checker”

```

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

```

The pseudo code of procedure “Senior Explorer” is in Algorithm 17.

Algorithm 17 Algorithm *Shadow Check without CWWT* — Procedure “Senior Explorer” — right agents

```

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

```

Analysis of Algorithm *Shadow Check without CWWT*

According to Lemma 1, we assume there are at least three agents in the ring network.

Because of the way an SE is defined, we remark that:

Lemma 16 *There is at least one SE.*

Proof: Given there are at least three agents in the ring, there will be at least two JEs exploring the ring in the same direction. Sooner or later, the third JE will meet one of the other JE. Hence, in such case, at least two SE will be created. But consider the situation in which only three agents wake up during the entire execution, and two agents die in the BH as JEs. In this case, the third agent sooner or later sees the token that a JE left before it went into the BH. The third JE then becomes a SE and starts exploring in the other direction. Eventually, it will reach the token that the other dead JE left. Given the two JE both died in the BH, the distance (on the explored segment) between the two tokens is $n - 2$. So the sole SE will be able to tell the location of the BH correctly.

□

Corollary 4 *At most two agents enter the BH.*

Proof: Before any explorer e (JE or SE) explores a new node, it leaves one or two tokens in the current node n as markers. Because of such marking, no other agent (a JE, a SE or a Checker) will go beyond node n . Also, if e successfully explores an unexplored node, then it goes back to node n (as in *CWWT*). If n is the same as when e left, e will then pick up the marking token(s) and continue exploring the next node along the ring. This mechanism ensures that there is no more than one agent that explores a node via the same link. Given there are only two links adjacent to each node, there are two links leading to the BH. Hence there is at most one agent

that enters the BH from either link connected to the BH. Thus, at most two agents enter the BH.

□

Lemma 17 *Two agents that meet in opposite directions will never pass without noticing each other, if there is no interruption from other agents.*

Proof: This is trivial to prove when two agents are in the neighboring nodes, because as soon as one agent arrives in the other node, it will notice the token(s) on the port. Instead, we are going to prove with one node between the agents and two nodes between the agents. When there are two nodes between the two agents, assume that the two agents has the same speed. Sooner or later, each agent will advance one node. Hence, two agents will end up in two neighboring nodes and become the same as when two agents start from two neighboring nodes. If, on the other hand, one agent is faster than the other one, once this agent advanced one node before the other agent, the situation will become the same as when two agents have only one node between them. Consequently the only situation we need to address is when there is one node between two agents (each in a distinct node). In this case, the faster agent (let us say it is the agent going to the left) will leave one token on the left port if it is a JE, leave two tokens on the left port if it is a SE, upon seeing no token in the node. Then the agent will go back to the right node, pick up its token(s), then go back to the left node. At this point, we are back to our trivial case of two agents in neighboring nodes.

□

Lemma 18 *A safe region can be created.*

Proof: Lemma 16 shows there is at least one SE during the execution. According to the definition of a SE, when two JE meets (that is, one JE sees the token of the other

JE) they will create two SEs. The two created SEs will explore the ring in opposite directions starting from the same node. Hence one segment of explored node(s) is created. If only three agents wake up during the entire execution, two JEs will die in the BH before meeting any other agent (see Lemma 1). The third JE will sooner or later meet a token that a JE left before dying in the BH. A segment of *safe region* is also created between the third JE and the token of another JE.

□

Lemma 19 *The length of a safe region keeps increasing until contains $n - 2$ links or $n - 1$ nodes.*

Proof: If there are at least two SEs in the network, the two SEs at each end of the *safe region* keep advancing. Each of them does not stop until it meets another SE or dies in the BH or finds out that the length of its *safe region* includes $n - 2$ links or $n - 1$ nodes during *checking* phase.

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

Otherwise, when a SE meets a JE, the JE picks its token if the SE has not picked it up, then becomes *Passive* immediately. The SE will keep exploring after it picks up the token of the JE.

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

□

Lemma 20 *The safe region length is checked after each growth.*

Proof: Since a *safe region* is between two SEs, each SE traverse its *safe region* after exploring one more node until it reaches the other SE. Hence a *safe region* is checked

after each growth by design of the algorithm.

□

Lemma 21 *A safe region will eventually contain $n - 2$ links and $n - 1$ nodes in it.*

Proof: Each SE goes to check the location of the partner SE after exploring one more node in each stage. Once the distance between the two LSP is $n - 2$, an agent stops exploring. This strategy guarantees that if there are only two SE left, they will not both die in the BH. If one of these two SEs dies in the BH, the surviving agent will sooner or later advance to the node next to the BH, then while seek its partner. It will then notice the distance between the two LSP is $n - 2$ and will locate the BH as the only node left between the two unexplored links.

□

Theorem 9 *Algorithm Shadow Check without CWWT correctly locates the BH in a ring with 3 or more scattered agents, each having 5 tokens. The total cost is $O(n^2)$ moves.*

Proof: Consider there are i safe regions in the ring. An agent traverses a *safe region* each time it explores one more node. Given a maximum of $n - 2$ moves (tokens or nodes) per agent to traverse such a *safe region*, and 2 agents per region, a maximum of $2(n - 2)$ moves are required to traverse a *safe region*. There are at most n nodes in the ring, thus there are at most $n - 1$ such traversals of *safe regions*. Hence, a maximum $(n - 1) * 2(n - 2)$ moves ($O(n^2)$) is taken for traversing *safe regions*.

In procedure “Checker”, the maximum number of each check is $2(n - 2)$ since each checker only traverses a *safe region* twice during its lifetime (because it becomes *Passive* after.). A checker is formed once two SEs or one JE and one SE meet “face to face”. Hence, there are no more than $n/2$ such checkers during the entire

algorithm. Thus, the total number of moves in procedure “Checker” is no more than $2(n-2) * n/2 = n(n-1)$. Hence, the total cost is $O(n^2)$ moves.

A JE uses one token on the port to mark its progress. Once a JE meets another JE, one extra token is used to mark the node in which two JEs meet and form a pair of SEs. This token will stay in the node until the algorithm terminates. A SE puts two tokens on a port as soon as it starts existing. It puts another two tokens on the port of the next node to mark the progress of the exploration process. The first two tokens are picked up and reused as soon as the second pair of tokens is put. Hence, a maximum of $1 + 2 + 2 = 5$ tokens are used by each agent.

□

3.4.3 Algorithm *Modified Shadow Check without CWWT*

Motivation

In the previous section, we presented algorithm *Shadow Check without CWWT*, which handles BHS in an unoriented ring with minimum of 3 scattered agents and 5 tokens per agents. According to Theorem 9, an agent on one of the i *safe regions* traverses this *safe region* in order to check the size of the *safe region* every time it finishes exploring one more node. This design stems from wanting a team of minimal size and consequently, having, by design, only one checker. So the explorers have to both explore the ring and check the size of their *safe region*. This cost $O(n^2)$ moves in the worst case.

We now consider the impact of adding a second checker. The modified algorithm *Shadow Check without CWWT* is designed so that:

- it works for a minimum of 4 scattered agents instead of 3
- eventually 2 checkers are formed and this allows the number of moves required

to be reduced to $\Theta(n \log n)$.

Modification

We can obtain algorithm *Modified Shadow Check without CWWT* from the following modifications to Algorithm *Shadow Check without CWWT*:

- change all actions “become *Passive*” of a JE in procedure “Junior Explorer” to “become a SE in the same direction reusing the two tokens of the SE that caught up”, whenever a SE caught up to this JE.
- delete the *check phase* in procedure “Senior Explorer”. Instead, as soon as a SE catches up with a JE, it will become a *Checker* working in the opposite direction.
- the procedure “Checker” in Subsection 3.4.2 is modified as follows:

A checker is created when that agent realizes it is in the middle of two SEs exploring in different directions. Once an agent becomes a checker, it checks the size (i.e., number of nodes) of the *safe region*. If this size is i , then the Checker follows the SE for $\lfloor (n - i)/2 \rfloor$ moves. We call this a *check*. A checker keeps *checking*, until the size of the *safe region* is $n - 1$. Then it concludes the only node left is the BH.

The pseudo code of procedure “Checker” follows:

Algorithm 18 Algorithm *Modified Shadow Check without CWWT* — Procedure “Checker”

```

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

```

Correctness and complexity

Given there are at least 4 agents in the ring and given Lemma 16, we know:

Corollary 5 *There are at least two SEs formed in algorithm Modified Shadow Check without CWWT.*

Lemma 22 *There are at least two checkers.*

Proof: Assume there are only 4 agents in the ring and all the agents are still JEs even after two of them died in the BH. The remaining JEs will either meet each other and then become two SEs or eventually see the token of a JE that died in the BH. In the first case, the two SEs will then explore the ring in opposite directions. Eventually each of them will see the token of a JE that died in the BH, then the SE

will become a *Checker*. Hence there are 2 *Checkers* eventually created. In the second case, there would be two pairs of SEs formed if the two JEs did not go into the BH. This is proved in Corollary 5. The two surviving SEs will then explore the ring in the opposite directions. Eventually they will meet and form two *Checkers*. The same holds if there are more than 4 agents. Hence, there are at least two checkers when there are 4 agents or more.

□

Theorem 10 *Algorithm Modified Shadow Check without CWWT correctly locates the BH with a minimum of 4 scattered agents, each using 5 tokens in a un-oriented ring topology with n nodes. The total cost is $\Theta(n \log n)$ moves.*

Proof: According to Corollary 5, 4 and Lemma 22, eventually there will be 2 checkers formed/left that keep checking the size of *safe regions* until the only *safe region* in the ring contains $n - 1$ nodes or $n - 2$ links. Hence the BH is correctly located.

Assume there are i *safe regions* in the ring. A SE keeps exploring nodes along the ring in one direction. Each SE thus traverses a maximum of n nodes. In procedure “Checker”, the maximum number of moves for a single check is $2n$ (given 2 SEs). There are no more than $\log n$ checks, given the procedure does not proceed with the next *check* until a checker follows a SE for $\lfloor (n - i)/2 \rfloor$ steps. So the total number of moves in procedure “Checker” is no more than $2n \log n$. Hence, the total cost is $O(n \log n)$ moves. The lower bound follows from the whiteboard model presented in [45]. Hence, the total cost of moves is $\Theta(n \log n)$.

Finally, because the modification does not affect the number of tokens used by each agent, by Theorem 9, 5 tokens per agent are used.

□

3.4.4 Algorithm *Shadow Check*

Basic Idea and General Description

To conclude, let us briefly sketch out algorithm *Shadow Check*, which uses $O(n^2)$ moves with only one (1) token per agent and a minimum of 3 agents using *CWWT* to locate the BH. It can be seen as a slight variation of Algorithm *Shadow Check without CWWT*. The only difference is that, using of *CWWT*, exploration can be handled using a single token. The algorithm can be summarized in the following Figure 3.4.

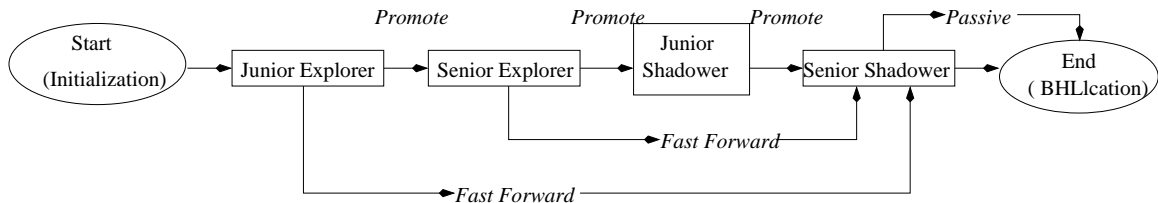


Figure 3.4: Flow Chart for Algorithm *Shadow Check*

For completeness, we now include a description of the procedures of this algorithm.

Initialization

The procedure “Initialization” works as follows.

Algorithm 19 Algorithm *Shadow Check* — Procedure “Initialization”

- 1: **procedure** INITIALIZATION
 - 2: $dist = 0$
 - 3: wake up and execute JUNIOR EXPLORER($dist$) to the right
 - 4: **end procedure**
-

Procedure *Junior Explorer*

Junior explorer (JE) goes to its right with *CWWT*, until either it dies in the BH or meets another explorer. If a JE meets a senior explorer (SE), then it becomes a

checker, otherwise becomes a SE upon meeting another JE.

Algorithm 20 Algorithm *Shadow Check* — Procedure “Junior Explorer”

```

1: procedure JUNIOR EXPLORER(dist)
2:   keep exploring the ring to the right with CWWT (one token on the port), dist ++,
   until A, B, C or D happens
3:   if A: detect a token on a port then
4:     moves the token to the middle of the node, dist = 0
5:     execute SENIOR EXPLORER(dist) in the direction that had no token on its port
6:   else if B: the CWWT token was moved to the middle of the node then
7:     picks up the token, dist = 0 execute SENIOR EXPLORER(dist) in the same direc-
   tion of exploration
8:   else if C: detects a token in the middle of the node then
9:     dist = 0, flag = 0, execute CHECK in the same direction of exploration
10:  else if D: dist =  $n - 2$  then
11:    becomes DONE
12:  end if
13: end procedure

```

Procedure *Senior Explorer*

A senior explorer will eventually either end up in the BH or locate the BH. The important issue is that a SE puts or removes one token in the middle of nodes while doing *CWWT*.

Algorithm 21 Algorithm *Shadow Check* — Procedure “Senior Explorer”

```

1: procedure SENIOR_EXPLORER(dist)
2:   explores the ring with CWWT (one token in the middle of the ring) for one step
3:   if A: detects a token on the exiting port then
4:     moves the token to the middle of the node,  $dist = 0$ ,  $flag = 0$ 
5:     execute CHECK in the opposite direction of exploration
6:   else if B: detects a token in the middle of the destination node then
7:      $dist = 0$ ,  $flag = 0$ , execute CHECK in the same direction of exploration
8:   else
9:     leaves a token in the middle of the node
10:    continues exploration in the opposite direction,  $dist ++$ , until detects another
    token in the middle of a node
11:    if  $dist = n - 2$  then
12:      becomes DONE
13:    else
14:      continues exploration in the opposite direction and  $dist --$  until its LSP
15:       $dist = 0$ , execute SENIOR_EXPLORER(dist) to the forward direction
16:    end if
17:  end if
18: end procedure

```

Procedure *Check*

Algorithm 22 Algorithm *Shadow Check* — Procedure “Check”

```

1: procedure CHECK(flag, dist)
2:   keeps walking and  $dist ++$  until A or B happens
3:   if A:  $dist = n - 2$  then
4:     becomes DONE
5:   else if B: sees a token in the middle of a node then
6:     if  $flag = 0$  then
7:        $flag = 1$ ,  $dist = 0$  executes CHECK in the opposite direction
8:     else
9:       becomes Passive
10:    end if
11:  end if
12: end procedure

```

Analysis of Algorithm *Shadow Check*

According to Lemma 1, we assume there are at least three agents in the ring network.

Because of the way the SE is defined, it is very clear:

Lemma 23 *There is at least one SE.*

Proof: There are two cases:

- Only three agents wake up during the execution, two agents die in the BH as JEs. In this case, the third agent will sooner or later reach the LSP of a died agent, then become a SE and start exploring in the opposite direction. Sooner or later it will reach the LSP of the agent. Given the two JEs both died in the BH, the distance (of the explored segment) between the two LSP is $n - 2$. So this sole SE will locate the BH correctly.
- Given there are at least three agents, and given a maximum of two agents can die in the BH (since their *CWWT* tokens will prevent others from using the dangerous links), and given we are not dealing with the previous case, then two agents will meet sooner or later, according to the definition of SE, and there will be at least two SEs. Hence, there is at least one SE.

□

Lemma 24 *An SE will locate the BH correctly if there is only one SE during the execution.*

Proof: See Case 1 in Lemma 23.

□

Lemma 25 *Let $S(t)$ denote a two-ends-increasing segment, when t ($t \geq 0$) denotes time. If $|S(t)| < n - 2$ then $\exists t'$ ($t' > t$), $|S(t')| > |S(t)|$.*

Proof: Assume there are at least two SEs in the network. The two explorers at each end of a $S(t)$ keep advancing. Each of them does not stop exploring until it meets another SE or JE. When a SE meets another SE that is at the end of a $S(t)'$,

according to the algorithm, both of them become checkers. In this case, after a finite time, the two segments will merge into one: $|S(t')| = |S(t)| + |S(t)'|$. $\therefore |S(t)| < |S(t)| + |S(t)'|$, $\therefore |S(t')| > |S(t)|$. Otherwise, when a SE meets a JE, the JE becomes the SE and continues exploring while the SE becomes a checker. Hence, $|S(t')| > |S(t)|$ still.

□

Lemma 26 *A two-ends-increasing segment will eventually contain $n - 2$ links and $n - 1$ nodes in it.*

Proof: Since each SE goes to check the location of the partner after exploring one more node in each stage, and as soon as the distance between the two LSPs is $n - 2$, an agent stops exploring. In this way, it is guaranteed that if there are only two SEs left, they will not both die in the BH. Once one of these SEs dies in the BH, the surviving agent will sooner or later advance to the node next to the BH, then while seeking its partner, it will notice the distance between the two LSPs is $n - 2$ and will locate the BH as the only node left between the two unexplored links.

□

Corollary 6 *At most two agents enters the BH.*

Proof: Due to *CWWT*, at most one agent enters the BH from any link connected to the BH. Given there are only two links that lead to the BH in ring topology, there are at most two agents that can enter the BH.

□

Theorem 11 *Algorithm Shadow Check correctly locates the BH with at least 3 scattered agents, each having 1 token in an un-oriented ring. The total cost is $O(n^2)$ moves.*

Proof: Consider there are i two-end-increasing segments in the ring. Each time an agent makes one more exploration, it traverses the segment back to the partner’s LSP and then back to its own LSP. This requires at most $2(n - 2)$ moves (since there are $n - 1$ non-BH nodes and $n - 2$ links that do not lead to the BH). And there are at most $n - 1$ such traversals (since, again, there are $n - 1$ non-BH nodes to explore). In procedure “Check”, there are no more than $n - 1$ such checks. So the total number of moves in procedure “Check” is no more than $(n - 1)(n - 2)$. Hence, the total cost is $O(n^2)$ moves.

□

3.5 BHS in an Anonymous Ring: Summary

In this chapter we studied BHS with tokens in the ring topology using both co-located agents and scattered agents.

For the co-located agents case, we first prove that a team of two agents is sufficient to locate the BH in finite time even in this coordination model, which is weaker than the whiteboard model. Furthermore, we prove that this can be accomplished using only $O(n \log n)$ moves in total, which is optimal, the same as with whiteboards. Finally, we show that the agents need to use only $O(1)$ tokens. These results are established constructively: we do present algorithms that allow a team of two agents to correctly locate the BH with that number of moves and with those few tokens. The protocol “Divide with Token +” uses a total of 10 tokens, while in algorithm *Divide with Token -*, the number of tokens used is reduced to 3.

Hence we show that, although tokens are a simpler means of communication and coordination than whiteboards, their use does not negatively affect solvability and it does not even lead to a degradation of performance. On the contrary, whereas the protocols using whiteboards assume at least $O(\log n)$ dedicated bits of storage at each

node [45], the ones proposed here use only three tokens in total. See Table 3.1.

	The case of Co-located Agents		
	Divide with Token	Divide with Token +	Divide with Token -
	Orientation	No	No
FIFO	Yes	Yes	Yes
uses <i>CWWT</i>	Yes	Yes	Yes
Knows k	No	No	No
Team Size	2 or more	2 or more	2 or more
Token Cost	$\log n$	5/agent	3 when $k = 2$ or 2/agent if $k > 2$
Move Cost	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$

Table 3.1: Summary of BHS in an Anonymous Ring with Co-located Agents

In the scattered agents case we investigate solutions for oriented and unoriented anonymous ring. We presented five different algorithms using scattered mobile agents to locate the BH in an anonymous ring topology. The summary is shown in Table 3.2.

When the orientation of the ring is available, we propose two Algorithms: *Gather Divide*, and *Pair Elimination*. We show that using algorithm *Gather Divide*, a minimum of 2 agents can locate the BH within $O(kn + n \log n)$ moves using only 1 token per agent. Then we prove that algorithm *Pair Elimination* can locate a BH in an anonymous ring by anonymous asynchronous and scattered agents, using $O(n \log n)$ moves in total and four (4) tokens per agent without any knowledge of the team size.

It is important to note that the proposed algorithm also solves (see Subsubsection 3.3.3), with the same cost, the problem of *Leader Electing* among the scattered agents, in spite of the presence of a BH. It also solves with the same cost the *Rendezvous* problem despite the presence of a BH, extending to tokens the results with whiteboards of [43].

Later we prove that locating the BH in an anonymous ring network using tokens

is feasible even if the agents are scattered and without a sense of direction. Thus, we prove that, for BHS, the token model is as powerful as the whiteboard model regardless of the initial position of the agents and orientation of the ring.

We prove that in an unoriented ring, a team of three or more scattered agents can locate the BH within n^2 moves, each agent using $O(1)$ tokens without knowing the number of the agents k . Interestingly, we can reduce the number of moves to $O(n \log n)$, which is optimal, with a minimum of 4 scattered agents using $O(1)$ tokens per agent. These results hold even if both agents and nodes are *anonymous*.

	Ring				
	The Case of Scattered Agents				
	Gather	Pair	Shadow	Shadow Check	Modified Shadow Check
	Divide	Elimination	Check	without <i>CWWT</i>	without <i>CWWT</i>
Orientation	Yes	Yes	No	No	No
FIFO	Yes	Yes	Yes	No	No
uses <i>CWWT</i>	Yes	Yes	Yes	No	No
Knowledge of k	Yes	No	No	No	No
Team Size	2 +	2 +	3 +	3 +	4 +
Token Cost	1/agent	4/agent	1/agent	5/agent	5/agent
Move Cost	$O(kn + n \log n)$	$\Theta(n \log n)$	$O(n^2)$	$O(n^2)$	$\Theta(n \log n)$

Table 3.2: Summary of BHS in an Anonymous Ring with Scattered Agents

In conclusion, we know BHS depends on three “performance impact factors”, namely: the minimum number of agents, the presence or absence of a sense of direction in the ring, and the number of tokens used. We observe that there is a tradeoff between the team size (i.e., number of agents) and the number of moves required by an algorithm. Given all the proposed algorithms require only a constant number of tokens per agent, we are unable to simulate the *distance identity* presented in [45]. *Distance identity* is crucial in order to achieve $\Theta(n \log n)$ moves with optimal team

size (3 agents) in the whiteboard model [45]. From the result we get in the token model, we conclude: with one more agent, the token model is as powerful as the whiteboard with respect to locating the BH in an unoriented ring. But with respect to memory requirements, our algorithms represent a considerable improvement on the whiteboard model.

Chapter 4

Black Hole Search in Hypercubes

4.1 Topological Characteristics

4.1.1 The Hypercube and Its Labeling

The hypercube is a generalization of a 3-cube to d dimensions, for any integer $d \geq 0$, also called a d -cube or measure polytope [110], see Figure 4.1.

If there are n nodes and they are numbered from 0 to $(n-1)$ in such a way that the addresses of the connected nodes differ by exactly 1 bit in their binary representation as illustrated in Figure 4.2, then such a labeled hypercube exhibits some desirable properties. For example, the labeled hypercube is useful in routing.

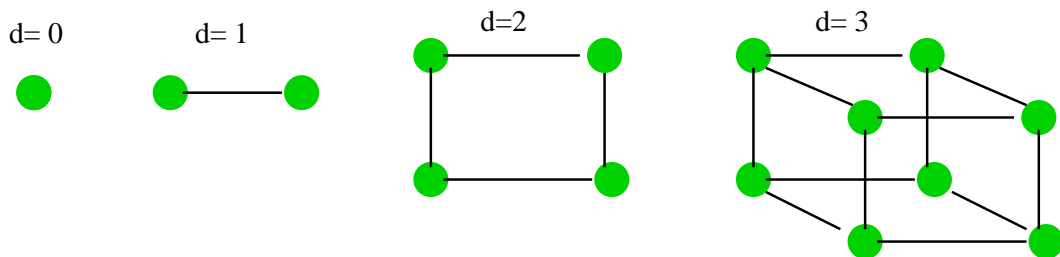


Figure 4.1: Hypercube Interconnection Strategy.

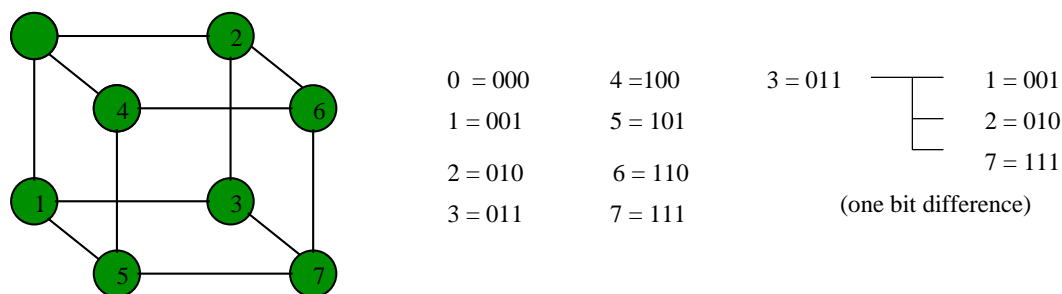


Figure 4.2: Hypercube Processor Numbering Scheme

The routing in the cube is reduced to finding any direct neighbor that reduces the number of bit differences between the address of the recipient and the address of the messenger. In Figure 4.2 for example, to route a message from node 0 (000 in binary) to node 7(111), the path through node 4(100) and 6(110) can be used since at each processor along the path the bit differences with 7 are reduced by one. This routing methodology is very efficient since it is “local” and “memoryless”, that is, it requires neither network access nor internal database lookup. Furthermore, randomly selecting a neighbor on a path toward a destination eliminates bottlenecks by distributing the routing on all possible candidates. This numbering scheme has another advantage:

From studying the characteristics of the hypercube topology we know it is possible to induce a sub-tree in the hypercube according to the following simple formula: the “parent” of a processor is its direct neighbor with the lowest address. This is illustrated for an 8-node hypercube in Figure 4.3 where the original hypercube is shown on the left and its sub-tree is shown on the right.

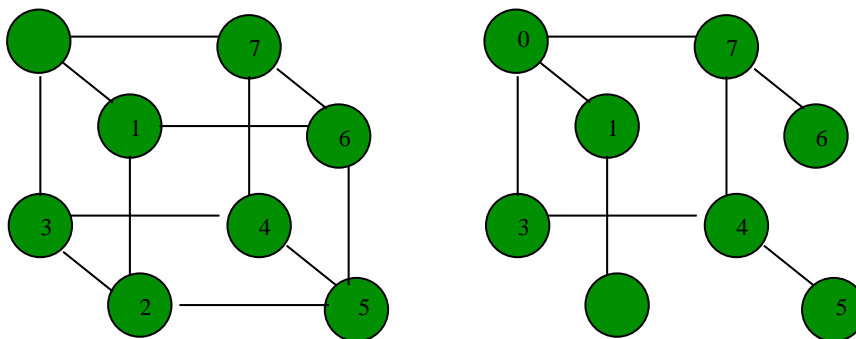


Figure 4.3: Sub-tree in a Hypercube of 8 Nodes

The “depth” of a processor in the tree is the number of non-zero bits in its address in binary representation. The “latency” between two nodes in the tree is $\log n$, where n is the number of nodes. For example in Figure 4.3, node 7 has three non-zero bits in its binary address (111) and is three “hops” away from the root node, node 0. Since the path from 0 to 7 is the longest, the latency is three. The sub-tree is useful for certain kinds of inter-processor communication, such as broadcasting (or flooding) a message to all nodes in the network topology. Since its latency is so low, the sub-tree enables such operations to be accomplished relatively rapidly [33].

Two issues draw our attention from the routing and broadcasting problems briefly described above:

- the labeling of the hypercube;
- the construction of a sub-graph/network.

Let us now introduce a labeling strategy that gives orientation (i.e., sense of direction) to the hypercube. In the rest of this section, we will then explain what kind of sub-graph/network we are going to construct in order to solve the BHS problem with minimum cost.

Let \mathcal{L} denote the *Natural* edge labeling [84] of a hypercube defined as follows:

- each node is represented by a d -bit binary string.

- an edge connecting nodes $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$ is labeled as i if and only if $x_i \neq y_i$.

4.1.2 Gray Code

A Gray code is an encoding of numbers so that adjacent numbers have a single digit differing by 1. The term “Gray code” is often used to refer to a “reflected” code, or more specifically still, to the binary reflected Gray code [17, 67, 71, 83].

The binary-reflected Gray code for n bits can be generated recursively by prefixing a binary 0 to the Gray code for $n-1$ bits, then prefixing a binary 1 to the reflected (i.e. listed in reverse order) Gray code for $n-1$ bits. The base case, for $n=1$ bit, is the most basic Gray code, $G = 0, 1$. (The base case can also be thought of as a single zero-bit Gray code ($n = 0, G = \text{“”}$) which is made into a one-bit code by a recursive process, as demonstrated in the example below).

Here are the first few steps of the above-mentioned reflect-and-prefix method, see Figure 4.4:

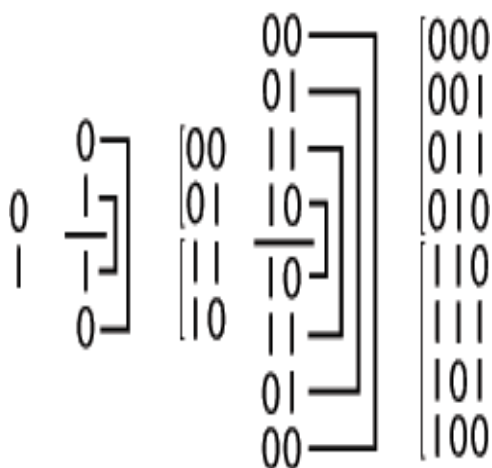


Figure 4.4: The first several steps in constructing a k -bit Gray code

The above paragraph suggests a simple and fast method of translating a binary value into the corresponding binary reflected Gray code. Each bit is inverted if the

next higher bit of the input value is set to one. This can be performed in parallel by a right bit-shift and a exclusive-or operation if they are available.

4.1.3 Hamiltonian Cycle

A Hamiltonian circuit, also called a Hamiltonian cycle, is a graph cycle (i.e., closed path) through a graph that visits each node exactly once [105]. A Hamiltonian path, also called a Hamilton path, is a path between two vertices of a graph that visits each vertex exactly once. A Hamiltonian graph, also called a Hamilton graph, is a graph that has a Hamiltonian circuit [110].

Well known relations between Hamiltonian circuits and Gray Code in a Hypercube are the follow:

- All hypercubes are Hamiltonian, and any Hamiltonian circuit of a labeled hypercube defines a Gray code [105].
- A Gray code also forms a Hamiltonian cycle on a hypercube, where each bit is seen as one dimension.

It is obvious that if we have a Gray code labeling in a hypercube, it is trivial to construct a ring. In turn, since we know how to solve the BHS problem in a ring, we would have a solution for the hypercube. However we remark that

- requiring Gray code labeling is a rather specific constraint.
- solving the BHS problem in a ring topology is rather expensive with respect to the cost of moves.

So the immediate issues we need to solve are:

- can we eliminate the need for a Gray code?
- can we reduce the number of moves by using the extra connectivity of the hypercube (compared with the one of a ring).

4.2 Basic Technique and Observations

As we mentioned in chapter 1, the most important parameters for a BHS solution strategy are the number of agents it requires (the team size), the number of tokens used and the total number of moves. Recall that in [44] 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. Here, M is the number of edges in the graph, n is the number of nodes in the graph, Δ is the maximum number of degree of the graph. In [48], BHS by mobile agents is studied in some interconnection networks including the hypercube topology. In that research, $\Theta(n)$ (n is the number of nodes) moves are required for a team of two co-located agents with the presence of *whiteboards* in the hypercube and with the help of a map. In the token model, if we consider using two tokens per port, we can achieve the same marking (of nodes and links as safe or dangerous) as when using whiteboards. The tokens will stay for the entire algorithm execution, once being left in the nodes or ports. In this case, lots of tokens are used (namely $n * (2^n)$). Whether we can use only $O(n)$ moves (as was achieved in a hypercube using whiteboards) becomes an open question, which is addressed in the next section. There, we will not only prove that the BHS problem can be solved (as efficiently as with a whiteboard model) using a token model, but also show that such solution can locate the BH using a constant number of tokens and minimum team size, which is an improvement over both [44] and [48].

4.3 Model and Assumptions

We denote a d -hypercube by \mathcal{Q}_d . \mathcal{Q}_d is an anonymous d dimensional hypercube with $n = 2^d$ nodes. All the links use natural labeling: There are 2^{i-1} links labeled as i

in each dimension, see Figure 4.5 and Figure 4.6. In Figure 4.5: on the left is a 2-hypercube with Hamiltonian cycle marked in red and two ($2^{2-1} = 2$) 2-dimension connecting links marked with red cross; on the right is a 3-hypercube with Hamiltonian cycle marked in red and four ($2^{3-1} = 4$) 3-dimension connecting links marked with red cross. From Figure 4.7 we can see: a 4-hypercube consists of two 3-hypercubes (depicted using purple dash lines) with eight ($8 = 2^{4-1}$) links labeled as 4. These links labeled as 4 are marked with a red cross in the figure. In fact, a well known property of hypercube is the following:

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

Noticing this interesting property of hypercubes helps us reduce the cost of the BHS problem for this topology.

Operating on \mathcal{Q}_d are k distinct mobile agents $a_1 \dots a_k$. In the case of co-located agents, all the agents start from the same \mathcal{HB} . Each agent has available $O(1)$ tokens. There is no other form of communication between the agents except for the tokens. Most importantly, unlike for the ring topology described in Chapter 3, we do not require that all the links and nodes obey the FIFO rule; but like for the ring topology, *CWWT* technique is used through out this chapter.

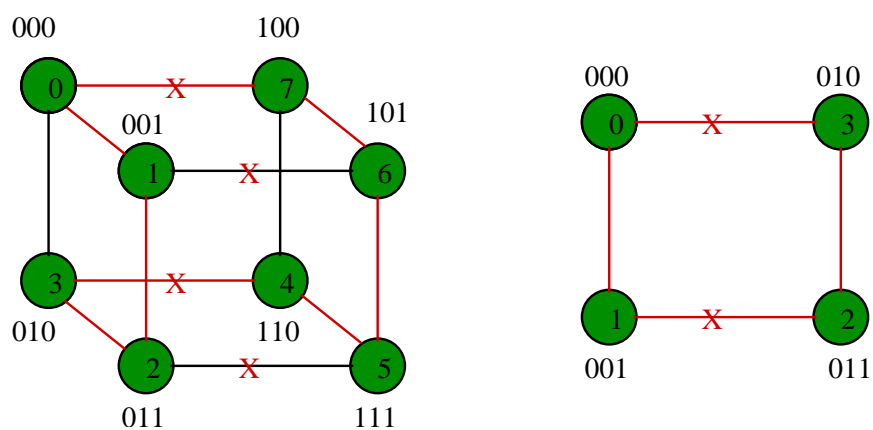


Figure 4.5: Hamiltonian cycles and connecting links in a 2-hypercube and a 3-hypercube

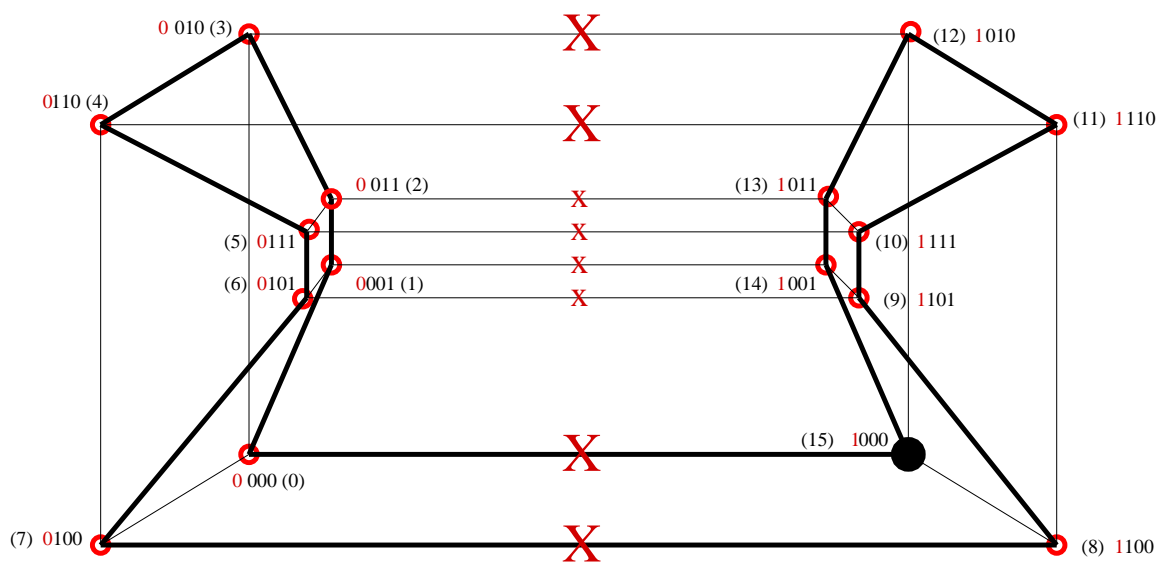


Figure 4.6: A 4-hypercube with Hamiltonian cycle highlighted in bold and eight ($2^{4-1} = 8$) dimension 4 connecting links marked with red cross.

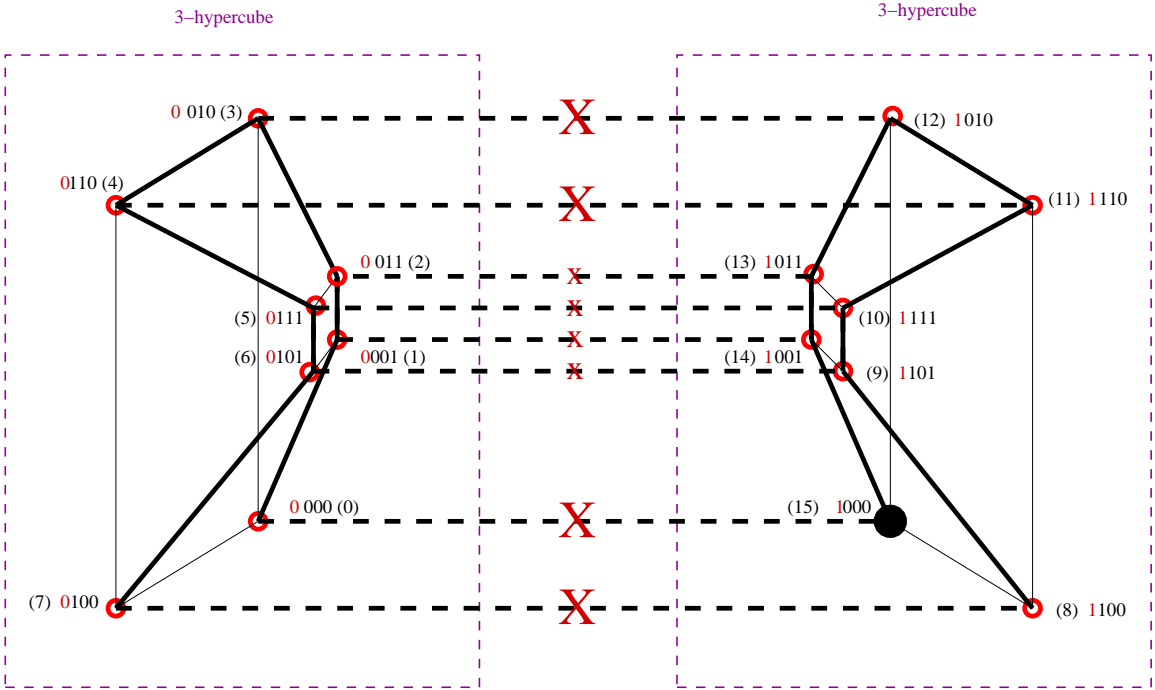


Figure 4.7: A 4-hypercube consisting of two 3-hypercubes (in purple dashed lines) with eight ($8 = 2^{4-1}$) 4-dimension connecting links marked with red cross.

4.4 Basic Ideas

The cost of the BHS problem in a ring network is the worst among the BHS cost in all the network topologies in whiteboard model. It is not surprising that the ring, as the sparsest of bi-connected network topologies, has the worst BHS cost. The key factor here is the connectivity. It is proved in [46] that edge bi-connectivity is required for BHS in asynchronous systems. An $\Omega(n \log n)$ lower bound is established for the BHS problem in an anonymous asynchronous ring (which is the least connected network topology) in both whiteboard [46] and token models [49]. We also know that if we want the mobile agents to locate the BH, it is necessary for the mobile agents to traverse the whole network [46].

Given the key property of hypercubes mentioned in Property 1, we find a way for two mobile agents (as we proved earlier, 2 is the minimum team size for BHS problem)

to traverse the hypercube with tokens. The basic idea can be carried out using the following four steps:

- let one agent stay in their common \mathcal{HB} , and the other agent move to the other ring through the connecting link using $CWWT$.
- have both agents explore their own ring (Hamiltonian Circuit of each $(d - 1)$ -hypercube) according to the permutation (see below) with $CWWT$. After an agent has finished exploring its ring, we call this ring a *safe* ring. Given the BH in the network can only be in one of the two rings, we 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 according to the permutation until it sees the marker of the other agent.
- 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* (see below) through a *safe* ring to the next node on the ring being currently explored.
- the agent that explores $n - 1$ nodes will survive and report the location of the BH.

Here we use an important technique we call *bypass*. This *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 general description of *bypass* technique is given in the following subsection, and the details of procedure “*Bypass*” will be explained in subsection 4.5.2.

For now, the immediate detail we need to solve is how do we make the agents only walk on the 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 : $\{p_1, p_2, \dots, p_{n/2}, p_1, p_2, \dots, p_{n/2}\}$.

The sequence is constructed as follows:

- 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}^i_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_{i-1} : $\{1, \mathcal{P}^i_{i-2}, i - 1, \mathcal{P}^i_{i-2}, 1, \mathcal{P}^i_{i-2}, i - 1, \mathcal{P}^i_{i-2}\}$
- when $d = i$, $n = 2^i$, \mathcal{P}^i_i : $\{1, \mathcal{P}^i_{i-1}, i, \mathcal{P}^i_{i-1}, 1, \mathcal{P}^i_{i-1}, i, \mathcal{P}^i_{i-1}\}$

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

- replace the second occurrence of '1' found in the sequence by 'd';
- 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 12 *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 $d = 2$, $\mathcal{P}_2: \{1, 2, 1, 2\}$, see Figure 4.8, it is obvious a Hamiltonian cycle is constructed correctly.

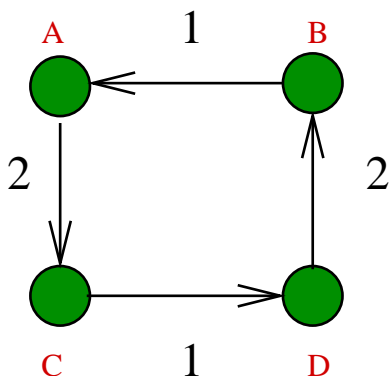


Figure 4.8: A Hamiltonian cycle constructed according to the permutation we proposed in a 2 dimensional hypercube

Now assume that when $d = i$, following the order of links indicated in $\mathcal{P}_i: \{1, \mathcal{P}_{i-1}, i, \mathcal{P}_{i-1}, 1, \mathcal{P}_{i-1}, i, \mathcal{P}_{i-1}\}$, 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:

- remove one of the two links labeled 1 in both $\mathcal{R} - a$ and $\mathcal{R} - b$.
- 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.
- 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} : $\mathcal{P}_{i+1}: \{1, \mathcal{P}_{i+1}, i + 1, \mathcal{P}_{i+1}, 1, \mathcal{P}_{i+1}, i + 1, \mathcal{P}_{i+1}\}$, a Hamiltonian cycle can also be constructed correctly. See Figure 4.9 This concludes the proof of:

Permutation \mathcal{P}_d computed by an agent constructs a Hamiltonian cycle of \mathcal{Q}_d ($n = 2^d$).

□

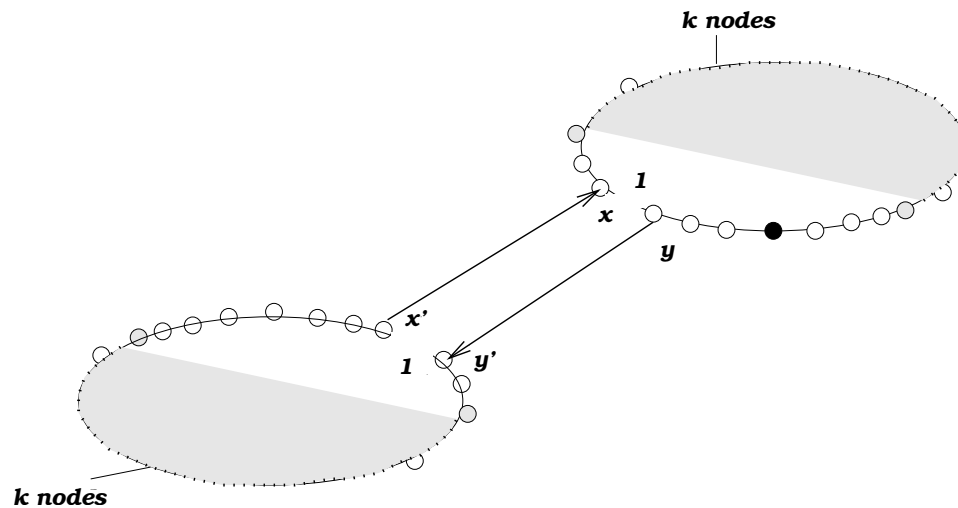


Figure 4.9: Merging Hamiltonian cycles of two i -hypercubes into a Hamiltonian cycle of $(i + 1)$ -hypercube. Here $k = 2^i$.

4.5 Co-located Agents Case — Algorithm *Two Rings*

4.5.1 General Description

In the case of co-located agents, 2 mobile agents a_1 and a_2 operate on \mathcal{Q}_d . They both start from the same \mathcal{HB} . 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 in Section 4.4. Before it starts exploring this sub-hypercube \mathcal{Q}_a , this agent leaves a token in the middle of its \mathcal{HB} 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. Once it 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{HB} 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. Recall that when an agent finishes exploring its ring, this ring becomes a *safe* ring. Once a_2 wakes up, it will notice the token in its \mathcal{HB} . 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 cycle, 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{HB} . 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 described in the next section until they locate the BH.

After a_1 finishes exploring \mathcal{R}_a , there are two possible token positions, see Table 4.1:

Tokens position	Meaning
There is no token in the node	a_2 is exploring \mathcal{R}_b
There is one token on the port leading to link d	a_2 is going to \mathcal{R}_b through link d

Table 4.1: Token positions and their explanation — 1

After a_2 wakes up, it may see three possible token positions, see Table 4.2:

Token Positions	Meaning
One token in the middle of the node	a_1 is exploring \mathcal{R}_a
One token in the middle of the node and one token on a port	a_1 is exploring the neighbor node of \mathcal{HB} on \mathcal{R}_a
There is only one token on the port leading to link d	a_1 is exploring \mathcal{R}_b before a_2 wakes up

Table 4.2: Token positions and their explanation — 2

4.5.2 Bypass Technique

The purpose of the *bypass* technique is to let one agent use the links available to it (beyond those that belong to one ring) in order to be able to explore the node next to the node that is currently under exploration by the other agent. This will ensure that two agents do not explore the same node at the same time. It also ensures 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. Given one token on a port means the next node is under exploration, a port can be classified as *with token* or *without token* at any point of time. An agent can only go through a port *without token*. The *bypass* technique will be and only be used after one of the two rings in the hypercube is fully explored, that is, once a *safe* ring exists. Before going into the *bypass* step, both agents are trying to finish exploring their respective ring. After an agent finished exploring the *safe* ring, it walks according to the permutation until it sees a node with token. This makes the agent get to the “*bypass*” procedure. The

other agent, which is still trying to finish exploring its ring, keeps exploring until it notices its token was moved by the other agent. This triggers that agent to advance to the “*bypass*” procedure.

Once in the “*bypass*” procedure, an agent acts differently whether advancing in the *safe* ring or in the *dangerous* ring. When an agent is in the *dangerous* ring, it moves the token to the port that leads to the link connecting the two rings if there is no token on this port in the same node. Otherwise, it picks up the token from this port; it will then walk to the *safe* ring through the link that has a port marked.

When an agent is in this *safe* ring, it goes two (2) steps according to the permutation. If the node is with a token, this agent will move this token to the port leading to the link labeled according to the next bit in the permutation, and go to the node through this link. This little procedure gets repeated until the agent arrives in a node without a token. If/when there is no token in the node, it will leave a token at the port that leads to the link connecting back to the *dangerous* ring, then it becomes ready to go back to the *dangerous* ring.

After an agent is ready to go back to the *dangerous* ring, it will go to explore one node in the *dangerous* ring, then come back to the *safe* ring through the same link in order to pick up the token it left on the port. But this token will either still be there or it will have been moved to the port leading to the link labeled according to the next bit in the permutation by the other agent. If the token is still at the same port, (meaning the other agent did not visit this node), then the agent will pick it up and go back to the *dangerous* ring. It will repeat exploring a new node in the *dangerous* ring with a *CWWT* token until it notices the token was moved to another port of the same node. It then starts another execution of the “*bypass*” procedure. See Figure 4.10, 4.11 and 4.12.

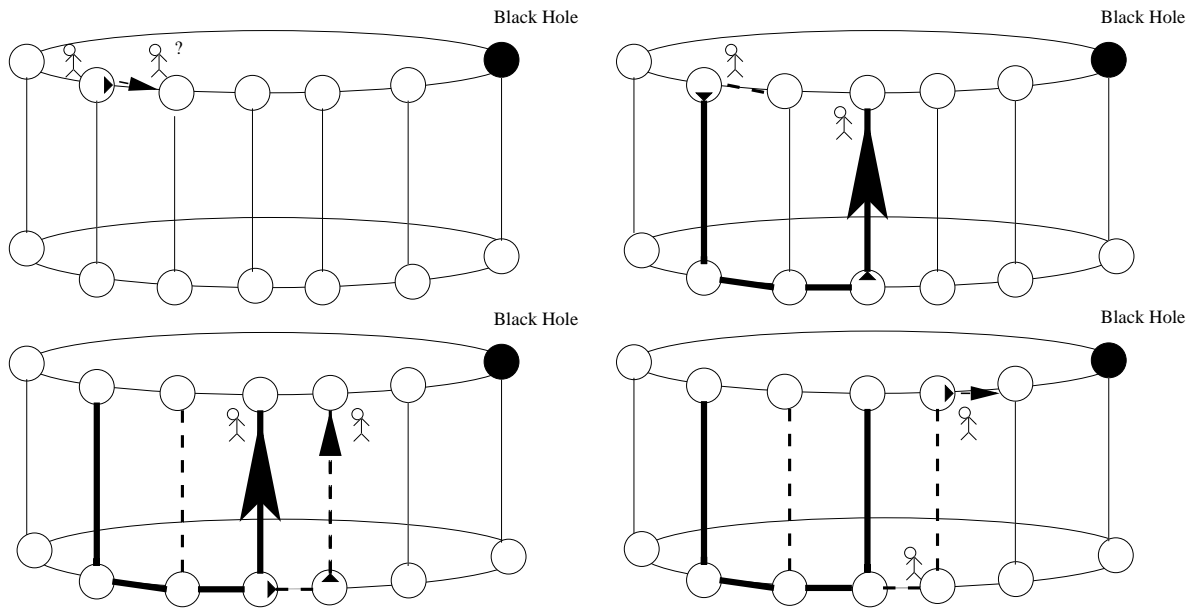


Figure 4.10: Bypass Technique — Steps 1, 2, 3 and 4

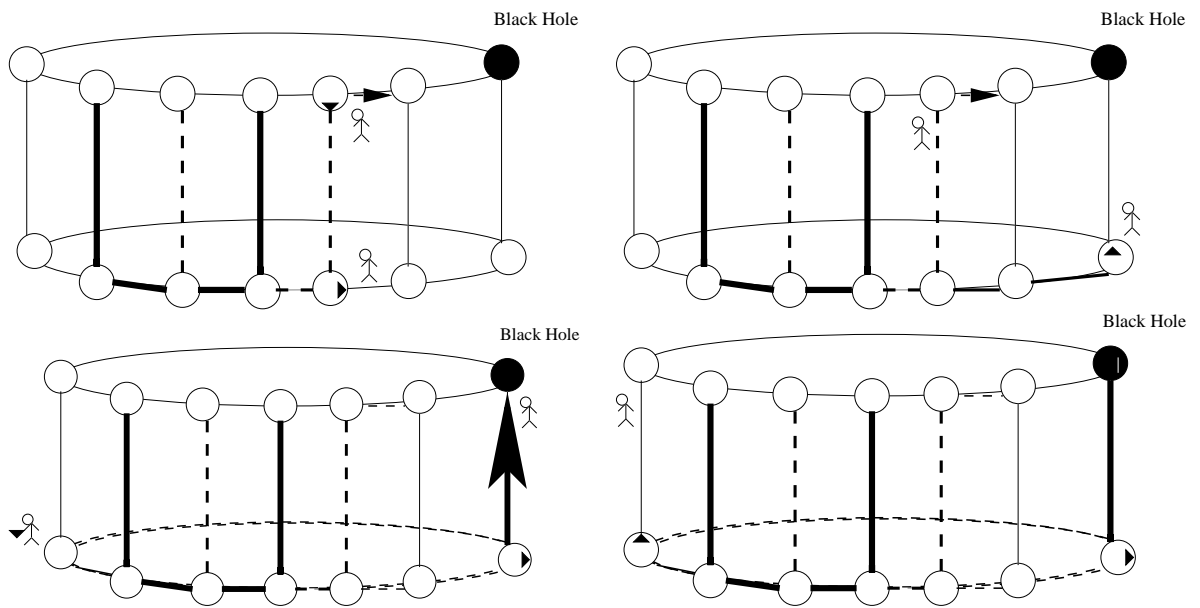


Figure 4.11: Bypass Technique — Steps 5, 6, 7 and 8

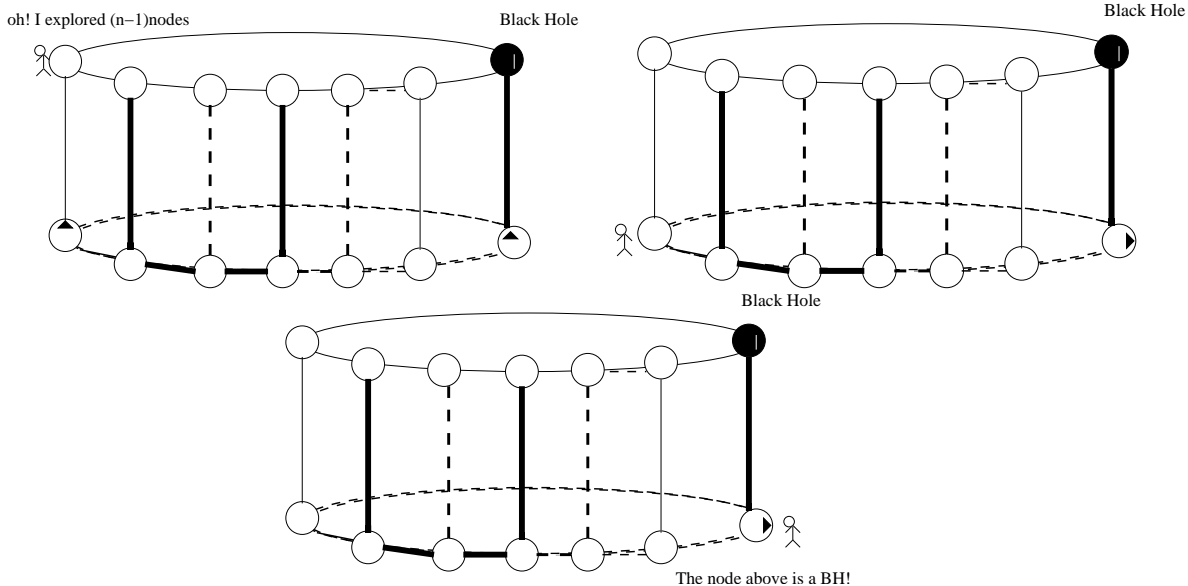


Figure 4.12: Bypass Technique — Steps 9, 10 and 11

4.5.3 Procedure “Initialization” and “Find a safe ring”

High Level Description

Two agents start from the same \mathcal{HB} . As soon as the first agent wakes up, it leaves a token in the middle of the node, then starts exploring the first Hamiltonian cycle, which is given in the permutation it constructed upon waken up. An agent explores a ring with $CWWT$ until it either finishes exploring this ring or sees a token in a node. When the second agent wakes up, it picks the token and goes to the second Hamiltonian cycle and starts exploring that ring with $CWWT$ until it either finishes exploring that ring, or sees a token in a node, or finishes exploring all the nodes in the hypercube except for the BH.

When an agent finishes exploring its ring, it goes to the other ring through a link labeled d using $CWWT$. Once on the other ring, this agent keeps walking with $CWWT$ according to the indication of its permutation until it sees a token in a node

or notices its *CWWT* token is moved by the other agent. Then both agents start executing procedure “*Bypass*”.

It is important to mention that we use variable *nCount* to record the total number of nodes an agent explores/*bypasses*.

Pseudo Code

The pseudo code of procedure “Initialization” and “Explore on a Ring” are in Algorithm 23, 24

Algorithm 23 Algorithm *Two Rings* — Procedure “Initialization”

```

1: procedure INITIALIZATION
2:   wake up,  $nCount=1$ 
3:   if there is no token in the  $\mathcal{HB}$  then //  $a_1$  wakes up
4:     put a token in the middle of the node
5:     execute EXPLORE ON A RING( $nCount$ )
6:   else if there is only one token on a port then //  $a_2$  wakes up after  $a_1$  started
   exploring the second ring.
7:     pick up the token
8:     execute EXPLORE ON A RING( $nCount$ )
9:   else if as long as there is a token in the middle then
10:    pick up the token in the middle
11:    execute EXPLORE ON A RING( $nCount$ )
12:   else if there are two tokens on a port then //  $a_2$  wakes up after  $a_1$  finished exploring
   the safe ring and has gone to the dangerous ring.
13:     if  $nCount < n/2$  then
14:        $nCount = nCount + n/2$ 
15:     end if
16:     pick up one token, execute Bypass( $nCount$ )
17:   end if
18: end procedure

```

Algorithm 24 Algorithm *Two Rings* — Procedure “Explore on a Ring”

```

1: procedure EXPLORE ON A RING( $nCount$ )
2:   if the agent is in the  $\mathcal{HB}$  then
3:     then go to the other ring through link  $d$  with  $CWWT$ 
4:   end if
5:   if A (notice your  $CWWT$  token was moved to another port) happens then
6:     if  $nCount < n/2$  then
7:        $nCount = nCount + n/2$ 
8:     end if
9:     execute  $Bypass(nCount)$ 
10:  else
11:    repeat
12:      start exploring the ring with  $CWWT$ ,  $nCount + +$ 
13:    until A or B or C happens
14:    if A (notice its  $CWWT$  token was moved to another port) happens then
15:      if  $nCount < n/2$  then
16:         $nCount = nCount + n/2$ 
17:      end if
18:      execute  $Bypass(nCount)$ 
19:    else if B (finished exploring its ring —  $nCount = n/2$ ) happens then
20:      if there is no token in the node then //  $a_1$  is in the  $\mathcal{HB}$  or  $a_2$  finished  $\mathcal{R}_b$ 
21:        go to the other ring through a link labeled  $d$ 
22:        go along the ring following the other agent, until see a token in the node
23:        execute  $Bypass(nCount)$ 
24:      else if there is one token in the middle then
25:        move this token to the port leading to link  $d$ 
26:        execute EXPLORE ON A RING( $nCount$ )
27:      else if there is one token on the port leading to link  $d$  then
28:        execute  $Bypass(nCount)$ 
29:      end if
30:    else if C ( $nCount = n - 1$ ) happens then // explored  $n - 1$  nodes
31:      become  $DONE$ , the only node it never visited is the BH
32:    end if
33:  end if
34: end procedure

```

4.5.4 Procedure “*Bypass*”

Detailed Description

Once an agent starts *bypass*, if it is on a *safe* ring, it will move the *CWWT* token of the other agent (which triggered “*Bypass*” procedure) to the port leading to the link labeled with the next bit in the permutation. Then it executes procedure “Back to the *Dangerous* Ring”.

When an agent goes (i.e., either walks with or without *CWWT*) into a node with a token in it on a *dangerous* ring, it moves the token to the port of the link leading to the *safe* ring, then go to the *safe* ring through the connection link between the two rings. See Figure 4.10.

After arriving on the *safe* ring, the agent, let’s call it a_1 , goes through two links labeled according to the next two bits in the permutation. There are two possibilities:

- First, there is a token in this node, there is one and only one explanation:

the other agent, let’s call it a_2 , finished exploring the node in the *dangerous* ring and noticed its *CWWT* token was moved by a_1 . So a_2 followed a_1 going on the *safe* ring. Eventually a_2 overtook a_1 and left that token in the node.

- Otherwise, a_1 will not see any token in this node.

Now, a_1 is ready to go to the *dangerous* ring and will execute procedure “Back to the *Dangerous* Ring”.

Procedure “Back to the *Dangerous* Ring” is as follows: An agent explores the next node on the *dangerous* ring. After exploring a node on the *dangerous* ring, an agent comes back to pick up its *CWWT* token. If it is still at the same port on which that agent left it, then this agent will pick up the token and go back to the node it just explored in the *dangerous* ring.

It then will explore the next node in the *dangerous* ring with *CWWT*. This agent goes to the next node through the link labeled with the next bit in the permutation. This step gets repeated until this agent notices its token is moved to another port in the same node. It then start to *bypass* again through the *safe* ring.

When an agent is on the *safe* ring, and its token is moved by the other agent to another port in the same node, it then picks up its token, and moves to the node through the link labeled with the next bit in the permutation. If there is a token in the node again, then it goes to the next node through the link labeled with the next bit in the permutation, until it ends up in the node without any token. Then it becomes ready to go to the *dangerous* ring again.

Once an agent is ready to go to the *dangerous* ring, it will execute procedure “Back to the *Dangerous* Ring” as mentioned above.

Pseudo Code

The pseudo code of procedure “*Bypass*” is in Algorithm 25.

4.5.5 Correctness and Complexity Analysis

Correctness

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 [59] between \mathcal{R}_1 and \mathcal{R}_2 .*

Proof: The hypercube has node and edge symmetries [59]. 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 $1, 2, 3, \dots, n$ such that $\pi(k') = k$ where k and k' are the respective dimensions

Algorithm 25 Algorithm *Two Rings* — Procedure “*Bypass*”

```

1: procedure Bypass
2:   if see a token on the port leading to link  $d$  when it is on a safe ring then // This
   only happens when an agent finished exploring the ring includes the  $\mathcal{HB}$ 
3:     move the token to the port leading to the link indicated by the first bit in the
   permutation
4:     execute BACK TO THE Dangerous RING( $nCount$ )
5:   end if
6:   if its CWWT token is moved to another port then
7:     pick up the token
8:   else
9:     move the token to the port leading to the node on the safe ring
10:  end if
11:  go to the safe ring through a link labeled  $d$ 
12:  go through two links that labeled with the next two bits in the permutation
13:  keep increasing  $nCount$ 
14:  if there is a token on a port then
15:    move it to the port leading to the link labeled with the next number in the
   permutation
16:     $nCount++$ 
17:    go to the next node through the link labeled with the next number in the per-
   mutation
18:  end if
19:  execute BACK TO THE Dangerous RING( $nCount$ )
20: end procedure
21: procedure BACK TO THE Dangerous RING( $nCount$ )
22:  leaves a token on the port leading to the node on the dangerous ring
23:  go to the node on the dangerous ring through this port
24:  go back to the safe ring through the same link
25:  if there is a token on the port where it left its token then
26:    pick up the token then go to the node on the dangerous ring that the agent just
   explored
27:    execute EXPLORE ON A RING( $nCount$ )
28:  else if the token is moved to another port in the same node then
29:    pick up the token,  $nCount++$ 
30:    go to the next node according to the next bit in the permutation
31:    if there is a token on a port then
32:      move it to the port leading to the link labeled with the next bit in the per-
   mutation
33:      go to the node through the link labeled with the next bit in the permutation
34:       $nCount++$ 
35:    end if
36:    execute BACK TO THE Dangerous RING( $nCount$ )
37:  end if
38: end procedure

```

of (u, v) and (u', v') , [59, 86]. We call v the *symmetric node* of v' . From Theorem 4.4, 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 27 *Either \mathcal{R}_1 or \mathcal{R}_2 is a safe ring.*

Lemma 28 *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{HB} . Before it starts, it will leave a token in the middle of the \mathcal{HB} 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 29 *One and only one agent dies in the BH.*

Proof: Given:

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

We can conclude that the two agents share the same sense of direction on both rings. Once an agent, say a_1 found 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, there is only one agent dies in the BH.

□

Theorem 13 *The BH is correctly located by the surviving agent.*

Proof: According to Lemma 28, and Lemma 27, at least one agent will eventually finish exploring a *safe* ring. As we mentioned in Lemma 29, this agent will go to help the other agent exploring the second ring using the *bypass* technique. We know from Lemma 29, that there is one and only one agent survives. The algorithm terminates as soon as an agent explored $n - 1$ nodes. Hence the BH is correctly located.

□

Complexity Analysis

Lemma 30 *Two (2) tokens in total are sufficient to locate the BH in a labeled hypercube with co-located agents.*

Proof:

- When the algorithm starts, one token is needed for the agent that wakes up first.
- Each agent needs one token to do *CWWT* in both exploring and *bypass* stages.

- The token used by the first agent in order to inform the second agent can be reused by the second agent, that is the one that wakes up later than the first one.

Hence, two (2) tokens in total are sufficient to locate the BH in a labeled hypercube with co-located agents.

□

Lemma 31 *A linear total number of moves is sufficient using Algorithm Two Rings.*

Proof: 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 Q_d . So, $O(n)$ moves is sufficient. In procedure “*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 [42], we can conclude:

Theorem 14 *Using two (2) co-located agents, two (2) tokens in total and $\Theta(n)$ moves, the BH can be successfully located in a edge labeled hypercube with n nodes.*

Chapter 5

Black Hole Search in Tori

5.1 Topological Characteristics

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.

A torus of dimensions $a \times b$ has $n = ab$ nodes $v_{i,j}$ ($0 \leq i \leq a-1, 0 \leq j \leq b-1$); each node $v_{i,j}$ is connected to four nodes $v_{i,j+1}, v_{i,j-1}, v_{i+1,j}$ and $v_{i-1,j}$. All the operations on the first index are *modulo* a , while those on the second index are *modulo* b (see Figure 5.1).

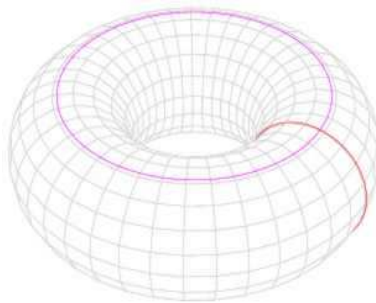


Figure 5.1: A simple Torus

Given these specific topology characteristics, we develop an algorithm *Cross Rings*, to locate the BH in an oriented torus with co-located agents. Later, we modify Algorithm *Cross Rings* in order to locate the BH in an oriented torus with scattered

agents. In both cases the BHS problem can be solved with minimum number of agents with $O(1)$ tokens in total and $O(n)$ moves, which is optimal. We also invent another Algorithm *Single Forward* which solves the BHS problem with 1 token per agent for k scattered mobile agents after executing $O(k^2n^2)$ moves.

5.2 Algorithm “Cross Rings” — The Case of Co-located Agents

5.2.1 Assumption, Basic Ideas and General Description

In this section, we study the BHS problem under the following assumption: there are k co-located mobile agents to locate the BH in a labeled torus (the ports in the torus are consistently labeled: *East, West, North, South*). Here k is not known to the agents. We prove later in this section that 2 co-located agents are enough to locate the BH with 4 tokens in total. The extra agents can be eliminated in their common \mathcal{HB} , according to the Elimination technique presented in Subsection 3.2.1 with only 1 more token. Hence, the description in the rest of this chapter is based on a team of 2 agents. The number of nodes n in this torus is known to both agents, but the dimensions of the torus is unknown. Most importantly, unlike for the ring topology described in Chapter 3, we do not require that all the links and nodes obey the FIFO rule; but like for the ring topology, *CWWT* technique is used through out this section.

As previously discussed, it is necessary for a team of agents to traverse the whole network in order to locate the BH. Obviously, the move cost of locating a BH is dependent on the path used by each agent to traverse the network. The number of moves remains a constant number if the BHS is executed by a constant number of co-located agents, each of which traverses the network for a maximum constant number of times. Figure 5.2 shows an obvious path that allows an agent to traverse

a torus and back to its \mathcal{HB} .

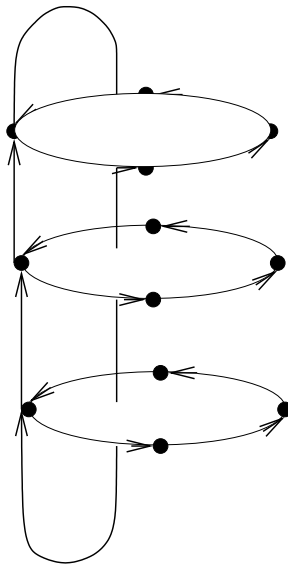


Figure 5.2: All the nodes of a $3 * 4$ Torus connected by minimum number of rings.

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, $\mathcal{R} - EW$ a *east-west* ring. Start 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 5.3):

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

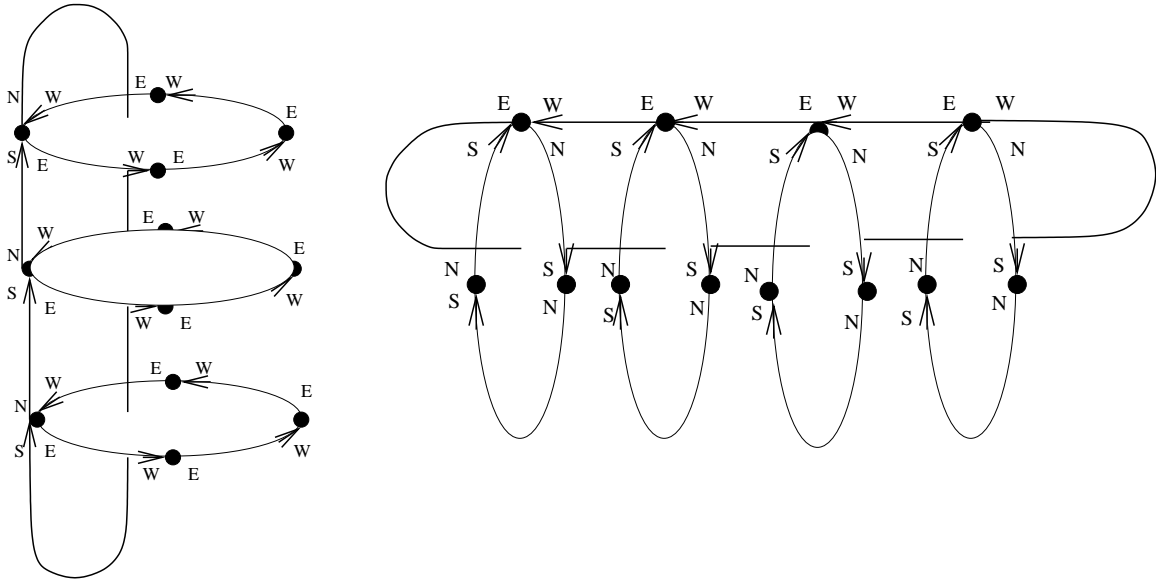


Figure 5.3: Two paths that allow an agent to traverse all the nodes in a labeled 3×4 torus

As we mentioned in Section 5.1, each node in a torus ‘has exactly 4 links that are labeled consistently. Hence, each node v is on both

- the *north-south* ring $\mathcal{R} - NS$ that includes the *North* and *South* links connected to v ;
- and the *east-west* ring $\mathcal{R} - EW$ that includes the *East* and *West* links connected to v .

If we let v (not a BH) be the \mathcal{HB} of a team of 2 agents, then the BH cannot be both on the $\mathcal{R} - NS$ and the $\mathcal{R} - EW$. This being said, the following observation becomes obvious:

Observation 6 *Let 2 agents start from v . If we let one agent traverse the $\mathcal{R} - NS$, and another agent traverse the $\mathcal{R} - EW$ ⁶, then there is at least one agent that survives this traversal.*

⁶The details of ring assignment for these 2 agents is explained in Section 5.2.2.

If there is only one agent that finished traversing a ring (the other agent died in the BH.), then we call this ring a *Base* ring. If both agents finished traversing their rings, then we call the ring that is traversed the earliest, a *Base* ring. A *Base* ring is also a *safe* ring.

From this point on, we assume that the *Base* ring is a *north-south* ring. Hereafter, all descriptions and algorithm procedures are based on this assumption. Importantly, if in fact the *Base* ring is a *east-west* ring, the descriptions and algorithm procedures can be obtained by exchanging all the keywords according to Table 5.1. Algorithm 30 is an example of obtaining procedure “Explore the *east-west* Rings” from procedure “Explore the *north-south* Rings” by changing the keywords according to Table 5.1.

words in the description and algorithm when a <i>north-south</i> ring is a <i>Base</i> ring	words in the description and algorithm when a <i>north-south</i> ring is a <i>Base</i> ring
<i>east</i>	<i>north</i>
<i>west</i>	<i>south</i>
<i>north</i>	<i>east</i>
<i>south</i>	<i>west</i>
<i>east-west</i>	<i>north-south</i>
<i>north-south</i>	<i>east-west</i>
<i>x</i>	<i>y</i>
<i>y</i>	<i>x</i>

Table 5.1: Keywords equivalence table

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* (defined in chapter 3) node using *CWWT* with 1 token on a port.

Before one agent starts exploring a *east-west* ring $\mathcal{R} - EW$, it puts 1 token in the middle of a node 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). An agent explores only the *east-west* ring that starts from the neighbor node to the *north* of the *UET*. Each time an agent finishes exploring one *east-west* ring, it will walk to the *north* until it sees a *UET*. It then put a *UET* in the next node to the *north*, come back to pick up the first *UET*, go to the next node to the *north* again and start exploring the *east-west* ring.

Sooner or later, one of the two agents will finish its *east-west* ring and advance to the *north* with the *UET* in the *north-south* ring. Eventually, 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 *RUE*. This situation highlights an important fact: a ring can be under exploration even though there is no *UET* in the starting node u .

Given there is only one BH, and there is no common node(s) shared by any two *east-west* rings, we obtain the following Lemma:

Lemma 32 *Eventually all but one east-west rings are explored.*

After a_1 finished exploring all but one *east-west* rings, it will go and help a_2 to explore the ring a_2 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* $\mathcal{R} - OCCU$ until it sees the *CWWT* token of a_2 . Now a_1 and a_2 will execute procedure “*Bypass on Torus*” procedure, of which the idea is introduced in Subsection 4.5.2, chapter 4. 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 5.2.

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

Table 5.2: Token positions and their explanation in Algorithm Cross Rings

The details of each procedure are given in the following three subsections.

5.2.2 Procedure “Initialization” and “Find a *Base* Ring”

When the first agent a_1 wakes up, there is no token on any port of the node. a_1 will put two tokens on the *north* port in its \mathcal{HB} to inform the agent a_2 that wakes up later: first, the *north-south* ring is a *RUE*; second, a_2 should explore the *east-west* ring starting from the \mathcal{HB} . Agent a_1 then keeps walking to the *north* with *CWWT*, until either it notices that its *CWWT* token is moved from *east* port to the *south* port or it goes back to the starting node in this *north-south* ring. When the first case happens, it means that a_2 , the second agent to wake up, found a *Base* (a *east-west*) ring and explored all the *north-south* rings except for the one a_1 is exploring. a_2 now is trying to *bypass* a_1 . Hence, a_1 executes procedure “*Bypass* on Torus” immediately.

In the second case, there are three possibilities again: there are either 2 tokens on the *north* port or 2 tokens in the middle, or 2/3 tokens on the *east* port. In the first case, *north-south* ring becomes the *Base* ring. a_1 informs a_2 (did not wake up) of this result by moving the 2 tokens from the *north* port to the middle of the node. It then executes procedure “Explore the *east-west* Rings” to the *east*. In the later case, 2 tokens in the middle of the \mathcal{HB} shows that the second agent a_2 woke up and finished

exploring the *east-west* ring before a_1 finished exploring the *north-south* ring. So, the *east-west* ring becomes the *Base* ring. a_1 then picks up the two tokens in the middle and keeps walking to the *east* until it sees 1 token in the middle of a node. It then executes procedure “Advance in the *Base* ring” to the *east*. In the last case, a_1 knows that a_2 woke up and exploring the first *east-west* ring to the *east*. a_1 then move two tokens (maybe 2 or 3 tokens on that port.) from the *east* port to the middle. a_1 then put the first *UET*, goes to the next node u to the *north* and put the second *UET*. a_1 immediately goes back to the \mathcal{HB} pick up the first *UET*, then goes back to u . Now a_1 starts executing procedure “Explore the *east-west* Rings” from node u .

When the second agent a_2 wakes up, there are either 2 tokens in the middle of the \mathcal{HB} or 3 tokens in the middle of the \mathcal{HB} or 2 tokens on the *north* port. In the first case, a_1 informed a_2 that the *north-south* ring is the *Base* ring. Then a_2 picks up the two tokens then keeps walking to the *north* until it arrives in the node with 1 token (the *UET*) in the middle. It then executes procedure “Advance in the *Base* Ring” to the *north*. In the second case, a_2 sees 3 tokens in the middle. This means that not only does a_1 tell a_2 the *north-south* ring becomes the *Base* ring, but also that a_1 is trying to put a *UET* in the next node to the *north*. Then a_2 picks up two tokens from the middle of the node and executes procedure “Advance in the *Base* Ring” to the *north* immediately. In the third case, a_2 follows the “instruction” from a_1 to go explore the *east-west* ring. a_2 first moves two token on the *north* port to the *east* port to inform a_1 that a_2 is exploring the first *east-west* ring. a_2 keeps advancing to the *east* with *CWWT* until either it notices that its *CWWT* token is moved from the *east* port to the *south* port or it goes back to the starting node in this *east-west* ring. When the former happens, a_2 executes procedure “*Bypass* on Torus”. When the latter happens, there are again two possible token position situations in the node:

1. There are 2 tokens on the *east* port. This means a_1 is still exploring the first

north-south ring, so this *east-west* ring becomes the *Base* ring. a_2 moves the two tokens to the middle of the \mathcal{HB} , in order to inform a_1 of this news. Then a_2 walks to the next node to the *east* and executes procedure “Explore the *north-south* Rings” to the *north*.

2. There are 2/3 tokens in the middle. This means that while a_2 is exploring the *east-west* ring, a_1 finished exploring the *north-south* ring. So, the *north-south* ring becomes the *Base* ring. If there are two tokens in the middle of the node, a_2 picks up the two tokens in the \mathcal{HB} then keeps walking to the *north* until it arrives in a node with 1 token in the middle. Then a_2 executes procedure “Advance in the *Base* Ring” to the *north*. But if there are 3 tokens in the middle, then a_2 knows that not only a_1 finished exploring the first *north-south* ring (the *north-south* ring becomes the *Base* ring), but also a_1 is trying to put a *UET* in the next node to the *north*. Then a_2 picks up two tokens in the middle then executes procedure “Advance in the *Base* Ring” to the *north* immediately.

The pseudo code of procedures “Initialization” and “Find a *Base* Ring” in algorithm *Cross Rings* is given in Algorithm 26 and 27.

Algorithm 26 Algorithm *Cross Rings* — Procedure “Initialization”

- 1: **procedure** INITIALIZATION
 - 2: $n = 0$
 - 3: wake up and execute FIND A *Base* RING
 - 4: **end procedure**
-

Algorithm 27 Algorithm *Cross Rings* — Procedure “Find a *Base Ring*”

```

1: procedure FIND A Base RING
2:   if there is no token on any port of the node then
3:     put two tokens on the north port; keep advancing north with CWWT, increasing
       xCount, until either A or B happens
4:     if A: CWWT token is moved to the east port then
5:       execute Bypass ON TORUS(xCount,y)
6:     else if B: back to the starting node then
7:       if there are 2 tokens in the north port then
8:         move the 2 tokens add one token to the middle,  $x = xCount$ ; execute
       EXPLORE THE east-west RINGS(x, yCount) to the east
9:         else if there are 2 tokens in the middle of the node then
10:          pick up the 2 tokens; keep walking east until see a node with 1 token in
            the middle, then execute ADVANCE IN THE Base RING(north, xCount, x)
11:          else if there are 3 tokens in the middle of the node then
12:            pick up 2 tokens, then execute ADVANCE IN THE Base RING(north,
              xCount, x)
13:          else if there are 2/3 tokens on the east port then
14:            move 2 tokens and add one token to the middle; go north;  $x++$ ,  $yCount+$ 
              +, put 1 token in the middle; come back to the  $\mathcal{HB}$  and pick the tokens, go north again;
            execute EXPLORE THE east-west RINGS(x, yCount) to the east
15:          end if
16:        end if
17:      else if there are 2/3 tokens on the north port then
18:        move 2 tokens to the middle
19:      repeat
20:        keep advancing to the east with CWWT, increasing yCount
21:      until either A: back to the starting node or B: its CWWT token is moved from
        east port to the south port
22:      if B happens then
23:        execute Bypass ON TORUS(yCount, x)
24:      else if A happens and there are 2 tokens on the east port then
25:        move the two tokens to the middle; put a token in the middle, go to the next
        node to the east;  $yCount++$ , put another token in the middle; go back to the  $\mathcal{HB}$ ,
        then pick up 1 token in the middle; walk to the next node to the east,  $y = yCount$ ,
        then execute EXPLORE THE north-south RINGS(y, xCount)
26:      else if A happens and there are 2 tokens in the middle then
27:        pick up the 2 tokens; keep walking to the north until see a node with 1 token
        in the middle, execute ADVANCE IN THE Base RING(north, xCount, x)
28:      else if A happens and there are 3 tokens in the middle then
29:        pick up 2 tokens; execute ADVANCE IN THE Base RING(north, xCount, x)
30:      end if
31:    else if there are 2 tokens in the middle then
32:      pick up the 2 tokens; keep walking north until see 1 token in the middle
33:      execute ADVANCE IN THE Base RING(north, xCount, x)
34:    else if there are 3 tokens in the middle then
35:      pick up 2 tokens; execute ADVANCE IN THE Base RING(north, xCount, x)
36:    end if
37: end procedure

```

In procedure “Advance in the *Base* Ring”, when an agent a_1 sees a *UET*, it will walk on the *Base* ring to the next node to the *north*. There are two possible token configurations:

- a *UET* in the node. This means the other agent a_2 walked faster than a_1 and this token was put by a_2 . Now a_1 goes to the *north* until sees a node without any token.
- there is no token in this node. This means a_1 is faster than the other agent a_2 .

Now, a_1 goes to put a second *UET* in the node, then goes back to the node on the *south* in order to pick up the previous *UET*. Once a_1 picks up the first *UET*, it then goes back to the node to the *north*. It is possible that the *UET* is still in the node, or the node is empty now. If the node is empty, it shows that the other agent a_2 overtook a_1 : a_2 put another *UET* in a node and a_2 picked up the *UET* that a_1 left in this empty node. Regardless whether the node is empty or not, a_1 is going to explore the *east-west* ring. The pseudo code of procedure “Advance in the *Base* Ring” in algorithm *Cross Rings* is in Algorithm 28.

Algorithm 28 Algorithm *Cross Rings* — Procedure “Advance in the *Base* Ring”

```

1: procedure ADVANCE IN THE Base RING(Direction, xCount, x)
2:   xTemp = 0 // xTemp is used to remember the position of the RUE in a north-south
   ring
3:   go to the next node to the north, xTemp ++, xCount ++
4:   if there is a UET in the node then
5:     go to the north and xTemp ++, xCount ++, until entering an empty node
6:     put a UET in this node, then go back to the node to the south
7:     pick up the UET, then go to the node to the north again
8:   else if xCount = x then
9:     go back to the node xTemp links away from to the South
10:  end if
11:  yCount = 0
12:  keep walking to the east and yCount ++, until see a token on the east port
13:  execute Bypass ON TORUS(yCount, x)
14: end procedure

```

5.2.3 Procedure “Explore the *east-west/north-south* Rings”

The pseudo code of procedure “Explore the *north-south* Rings” and “Explore the *east-west* Rings” are in Algorithm 29 and 30.

Algorithm 29 Algorithm *Cross Rings* — Procedure “Explore the *east-west* Rings”

```

1: procedure EXPLORE THE east-west RINGS( $x, yCount$ )
2:   repeat
3:     advance with CWWT to the east,  $yCount++$ 
4:   until A: the token is moved from the east port to the south port, or B: back to the
      first node it was, in this east-west ring
5:   if A happens then
6:     execute Bypass ON TORUS( $yCount, x$ )
7:   else if B happens then
8:     if there is no token in the middle of this node then
9:       keep going north and increase  $y$  until arriving to the node that has 1 token
      in the middle
10:    end if
11:    execute ADVANCE IN THE Base RING( $north, xCount, x$ )
12:  end if
13: end procedure

```

Algorithm 30 Algorithm *Cross Rings* — Procedure “Explore the *north-south* Rings”

```

1: procedure EXPLORE THE north-south RINGS( $y, xCount$ )
2:   repeat advance with CWWT to the north,  $xCount++$ 
3:   until A: the token is moved from north port to the west port, or B: back to the
      first node it started from in this north-south ring
4:   if A happens then
5:     execute Bypass ON TORUS( $xCount, y$ )
6:   else if B happens then
7:     if there is no token in the middle of this node then
8:       keep going east and increase  $x$  until the node which has 1 token in the middle
9:     end if
10:    execute ADVANCE IN THE Base RING( $east, xCount, x$ )
11:  end if
12: end procedure

```

5.2.4 Procedure *Bypass* on Torus

Bypass on Torus Technique

As we explained in Subsection 4.5.2, Chapter 4, the purpose of the *bypass* technique is to let one agent use the links available to it (beyond those that belong to one ring) in order to be able to explore the node next to the node that is currently under exploration by the other agent. This will ensure that two agents do not explore the same node at the same time. It also ensures 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.

In this chapter, we still use a technique similar to the *bypass* one to solve BHS in a torus. In this section, we still use one token on a port as one agent's *CWWT* token. In other words, a token on a port means the next node through this port is under exploration; a port can be classified as *with token* or *without token* at any point of time. An agent can only go through a port *without token*. The *bypass* technique will be and only be used after all but one *east-west*⁷ ring is still under exploration.

After an agent a_1 has finished exploring $x - 1$ *east-west* rings, it walks to the *east* on the only *east-west* ring left (*CWWT* walk is not necessary), it also keeps increasing variable $yCount$, until it sees a node with a token on the *east* port. This makes the agent get to the *bypass* step. The other agent a_2 , which is still trying to finish exploring its *east-west* ring, keeps exploring until it notices its *CWWT* token was moved by the other agent a_1 . This triggers a_2 to execute the “*Bypass* on Torus” procedure. But it is possible that, by the time a_1 arrives in the node in which a_2 left its *CWWT* token, a_2 has already disappeared in the BH. Then a_1 will just continue executing the “*Bypass* on Torus” procedure and eventually locate the BH when it

⁷Recall that we assumed the *north-south* ring that includes the \mathcal{HB} , is the *Base* ring at the beginning of this chapter.

notices that $y - 1$ ($yCount = y - 1$) nodes on this *east-west* ring are explored.

Once it starts the procedure “*Bypass on Torus*”, an agent acts differently whether advancing in a *safe* ring (all the *east-west* rings that are explored by an agent) or in a *dangerous* ring (the only *east-west* ring that is under exploration, namely, the *RUE*). When an agent is in the *dangerous* ring, it either moves the token to the *north* port that leads to the link connecting the two rings if there is no token on it; or, otherwise, it picks up the token that has been moved to the *north* port. It will then walk to the *safe* ring through the *north* port.

When an agent is in a *safe* ring, it goes two (2) steps to the *east*. If the node has a token on the *south* port, this agent will move this token to the *east* port. It then goes to the node on the *east*. This procedure gets repeated until the agent arrives in a node without a token. It is important to know that an agent keeps increasing variable $yCount$ each time it traverse a link on the *safe* ring.

If there is no token in the node, this agent will leave a token on the *south* port that leads to the link connecting back to the *dangerous* ring, then it becomes ready to go back to the *dangerous* ring.

After an agent is ready to go back to the *dangerous* ring, it will go to explore one node in the *dangerous* ring through the *south* port, then come back to the *safe* ring through the same link in order to pick up the token it left on the *south* port in the *safe* ring. But this token will either still be there or it will have been moved to the *east* port of the node by the other agent. If the token is still on the same port, meaning the other agent did not visit this node, then the agent will pick it up and go back to the *dangerous* ring. It will repeat exploring a new node to the *east* in the *dangerous* ring with a *CWWT* token and keeps increasing variable $yCount$ each time it traverse a link, until it notices its *CWWT* token was moved to the *north* port of the same node. It then starts another execution of the “*Bypass on Torus*” procedure.

This keeps executing until an agent notices the variable $yCount$ reaches $y-1$ which means there are $y-1$ nodes were explored in this *east-west* ring. Hence, the algorithm terminates. The only node that is not explored is the BH. See Figure 5.4, 5.5 and 5.6.

Detailed Description

When an agent goes (i.e., either walks with or without *CWWT*) into a node with a token in it on a *dangerous* ring, it will move the token to the *north* port (which leads to the *safe* ring), then it will go to the *safe* ring through this *north* port. See Figure 4.10.

After arriving on the *safe* ring, the agent, let's call it a_1 , goes two steps to the *east* and keeps increasing $yCount$. There are two possibilities:

- First, there is a token on the *south* port of this node. In this case, there is one and only one explanation:
the other agent, let's call it a_2 , finished exploring a node in the *dangerous* ring and noticed its *CWWT* token was moved by a_1 . So a_2 followed a_1 going on the *safe* ring. Eventually a_2 overtook a_1 and left that token in the node (which is possible because we do not assume the FIFO requirement on any link or node).
- Otherwise, a_1 will not see any token in this node.

When the former case happens, a_1 moves this token to the *east* port and walks to the next node to the *east*. This step is repeated until a_1 arrives in a node without any token in it.

If there is no token in the node a_1 arrives into, it will leave a token on the *south* port. Now, a_1 is ready to go to the *dangerous* ring and will execute procedure “Back to the *Dangerous* Ring — Torus”.

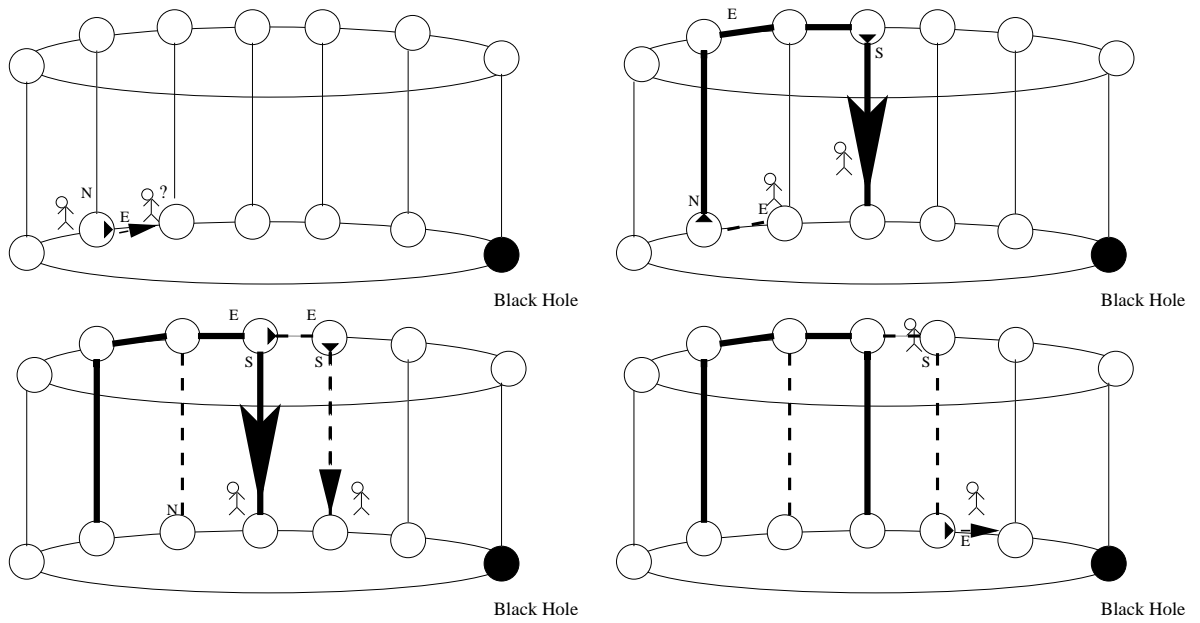


Figure 5.4: Bypass Technique on Torus — Steps 1, 2, 3 and 4

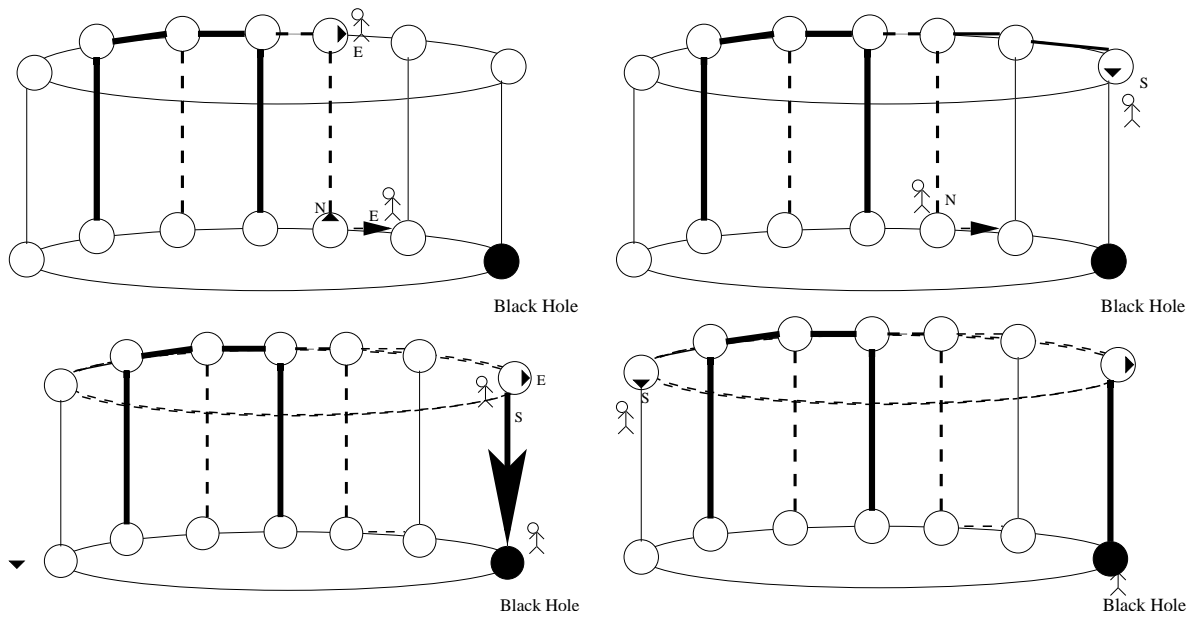


Figure 5.5: Bypass Technique on Torus — Steps 5, 6, 7 and 8

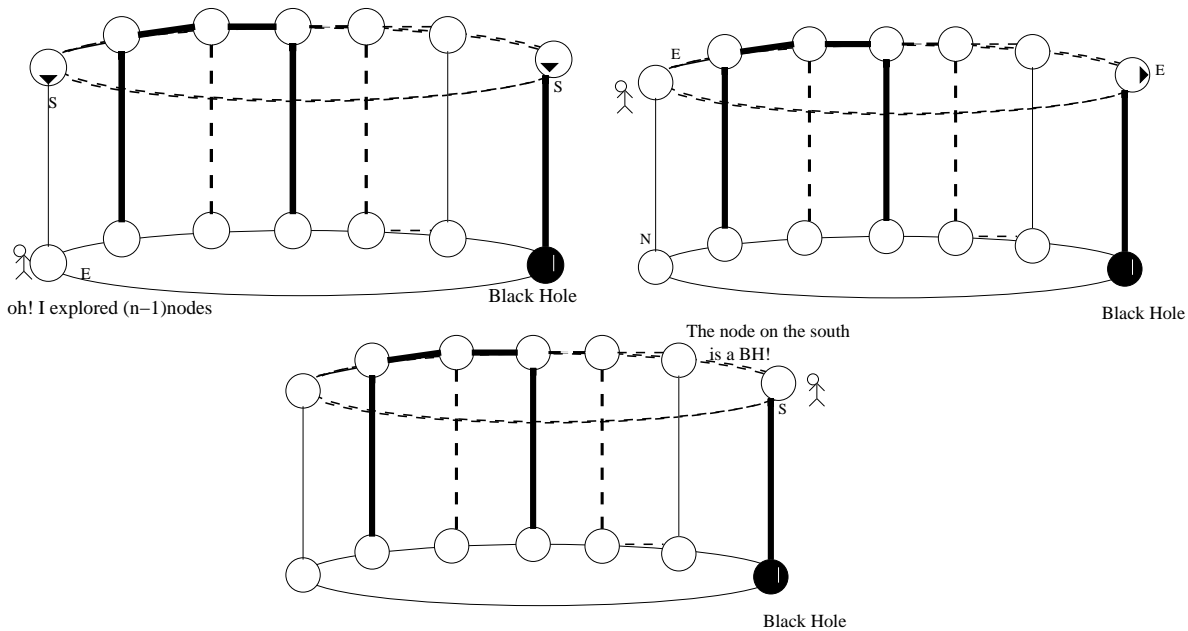


Figure 5.6: Bypass Technique on Torus — Steps 9, 10 and 11

When an agent is on the *safe* ring, if its token is moved by the other agent to the *east* port in the same node, it then picks up its token, and moves to the node through the *east* port. If there is a token in the node again, then it goes to the next node through the *east* port, until it ends up in the node without any token. Then it becomes ready to go to the *dangerous* ring again.

Once an agent is ready to go to the *dangerous* ring, it will execute procedure “Back to the *Dangerous* Ring — Torus”. Let us elaborate.

Procedure “Back to the *Dangerous* Ring — Torus” is as follows: An agent explores the node on the *dangerous* ring through the *south* port. After exploring a node on the *dangerous* ring, an agent comes back to pick up its *CWWT* token. If this token is still on the *south* port on which that agent left it, then this agent will pick up the token and go back to the node it just explored in the *dangerous* ring.

It then will explore the node in the *dangerous* ring with *CWWT* through the *east* port. This step gets repeated until this agent notices its token is moved to the *north*

port in the same node. It then start to *bypass* again through the *north* port to the *safe* ring.

Pseudo Code

The pseudo code of procedure “*Bypass* on Torus” and “Back to the *Dangerous* Ring — Torus” are shown in Algorithm 31 and 32.

Algorithm 31 Algorithm *Cross Rings* — Procedure “*Bypass* on Torus”

```

1: procedure Bypass ON TORUS(yCount, x)
2:   if its CWWT token is moved from east port to the north port then
3:     pick up the token
4:   else
5:     move the token from east port to north port
6:   end if
7:   go to the node through the north port; go through two links through the east port;
   yCount ++
8:   if yCount  $\neq$   $n/x - 1$  then
9:     if there is a token on the south port then
10:      move it to the east port; keep going to the next node to the east and yCount +
   +, until a node without a token on the south port
11:    end if
12:    leave a token on the south port; go to the node on the dangerous ring through this
   port; go back to the safe ring through the same link; execute BACK TO THE Dangerous
   RING — TORUS(yCount, x)
13:   else
14:     become DONE.
15:   end if
16: end procedure

```

5.2.5 Correctness and Complexity Analysis

Correctness

According to the Observation 6 and Lemma 32 we introduced in subsection 5.2.1, we can obtain the following Lemma:

Lemma 33 *At least one agent will find a Base ring in the torus.*

Algorithm 32 Algorithm *Cross Rings* — Procedure “Back to the *Dangerous Ring* — *Torus*”

```

1: procedure BACK TO THE Dangerous RING — TORUS( $yCount, x$ )
2:   if there is a token on the south port then
3:     pick up the token then go to the node through the south port; execute EXPLORE
     THE east-west RINGS( $yCount, x$ )
4:   else if the token is moved to the east port of the same node then
5:     pick up the token, go to the node to the east;  $yCount++$ 
6:     if  $yCount = n/x - 1$  then
7:       become DONE, the only node that is not visited is the BH
8:     else
9:       execute BACK TO THE Dangerous RING — TORUS( $yCount, x$ )
10:    end if
11:  end if
12: end procedure

```

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

Lemma 34 $x - 1$ *east-west rings* will be explored eventually.

Proof: According to Lemma 32, 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 35 The UET advances on the *north-south ring* correctly.

Proof: When the first agent starts exploring the first *east-west* ring, it puts a UET for the first time. An agent does not move a UET (one token in the middle of a node) until it finished exploring an *east-west* ring. This agent then walks to the next node to the *north* of the current node, puts a UET, then comes back to pick up the old UET. Then and only then does it explore the *east-west* ring starting from the node with the UET.

We show in Figures 5.7, 5.8 and 5.9 that even without the FIFO assumption, the *UET* can be advanced correctly. In Figure 5.7, we show that instead of picking up the *UET*, and carrying it to the next node to the *north*, an agent a_1 can correctly advance the *UET* by doing the following three steps:

- leaves a second *UET* in the next node to the *north*
- goes back to the node to the *south* and picks up the first *UET*.
- goes to the node to the *north* (where the second *UET* was left).

This avoids the partner agent a_2 overtaking a_1 without seeing the *UET*.

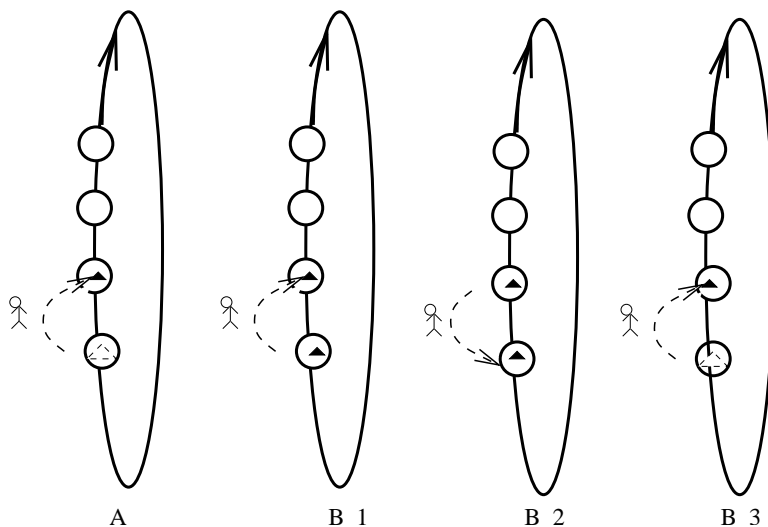


Figure 5.7: The case of one agent involved in moving the *UET*.

There are four scenarios that will possibly happen, when both agents are in front of a *UET*:

- Scenario A: both agents are in the node N_1 with the first *UET*:

Then they both try to put a *UET* (the second) in the next node N_2 to the *north*. See Figure 5.8. This triggers the next scenario.

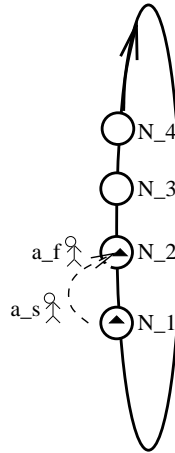


Figure 5.8: a_f puts the second *UET* before a_s does.

- Scenario B: both agents are in the node with the second *UET* N_2 for the first time: See a) and b) in Figure 5.9. The faster agent a_f will then put a *UET* in the node, and then go back to N_1 to pick up the first *UET* in the node to the *south*. When the other agent a_s (the slower one) arrives in the node N_2 , it will notice that the node has a second *UET* in it. It then will go to the next node N_3 to the *north*. So far, there is no conflict in the above scenarios.

At this point, a_f and a_s have two different destination nodes N_1 and N_3 . The first *UET*, will be picked up by a_f without any problem. But there are two possible situations in N_3 See b), c) in Figure 5.9:

1. N_3 has a *UET* in it. This means a_f overtook a_s , and left a *UET* there. So, a_s keeps going to the next node to the *north*, until it arrives in an empty node N_i ($i \geq 4$).
2. N_3 is empty.

Now a_s puts a *UET* in N_j ($j \geq 3$). See d) in Figure 5.9. This means that a_f is still exploring the *east-west* ring that starts from N_{j-1} . Then a_s goes back

to N_{j-1} intending to pick up the *UET* in it. It is possible that a_f may be still exploring the *east-west* ring or it finished exploring the *east-west* ring that starts from N_{j-1} . We observe that even a_f finished exploring the *east-west* ring that starts from N_{j-1} , it will go to N_j first in order to put a *UET* according to the algorithm. Given a_s has already put a *UET* in N_j , this leads a_f to go to N_{j+1} (instead of possibly go to N_{j-1} that is on the *south* of N_j) to the *north* of N_j . Hence, a_s will pick up the *UET* in N_{j-1} without any conflict with a_f .

After a_s picked up the *UET* in N_{j-1} , it goes back to N_j through the *north* port. Once a_s is in N_j , two other scenarios can occur:

- Scenario C: the *UET* is still there. See e) in Figure 5.10. This means a_f is either still exploring the *east-west* ring that starts from N_{j-1} , or it overtook a_s and is trying to leave another *UET* in the node N_{j+1} next to the *north* of N_j .
- Scenario D: the *UET* is missing. See f) in Figure 5.10. There is only one explanation for this situation:

a_f overtook a_s and left another *UET* in the node N_{j+1} next to the *north* of N_j . It came back to N_j and picked up the *UET*.

Regardless what happens in node N_j now, a_s will start exploring the *east-west* ring. This shows there is no conflict in these two scenarios between a_s and a_f . Hence, advancing the *UET* between the two agents can be executed correctly.

In Figure 5.11, we illustrate one of the four situations: if a_2 sees two *UET*s in a row, it will know the second *UET* is the real *UET*, and act accordingly. Namely, it will go to the next node to the *north*, put a second *UET*, then come back to pick up the first *UET*.

When an agent that intends to advance the *UET* goes back to its \mathcal{HB} , it stops advancing on the *Base* ring. Instead, it picks up the *UET* and goes back to the node

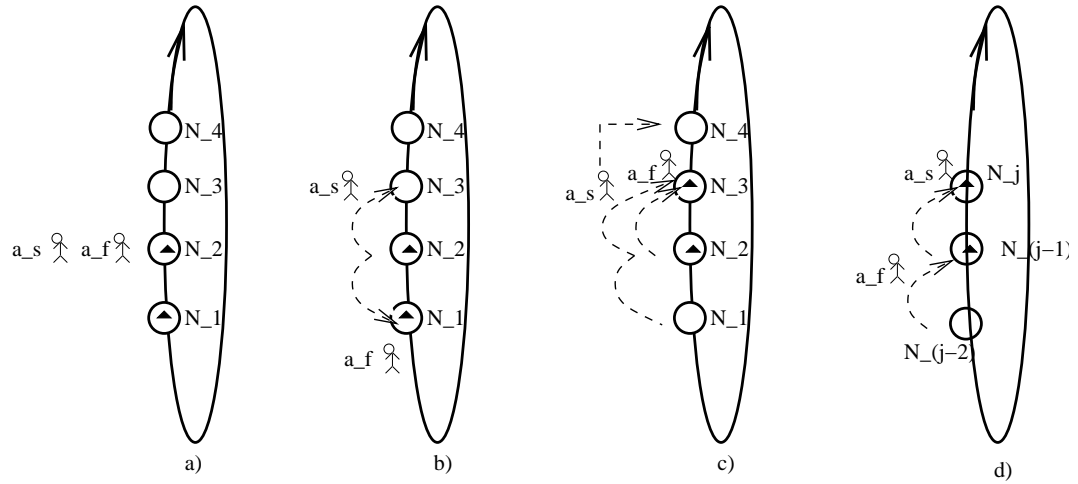


Figure 5.9: The first two scenarios when two agents are both ready to move the UET.

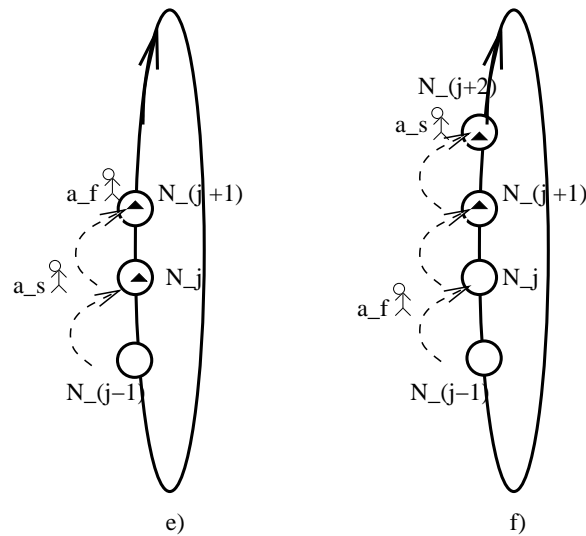


Figure 5.10: The last two scenarios when two agents are both ready to move the UET.

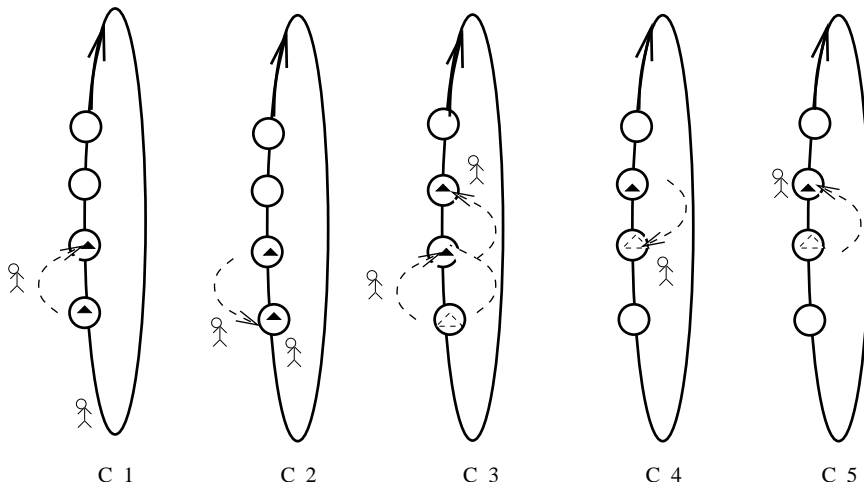


Figure 5.11: One example of how two agents can handle the UET correctly.

in which the other agent left the *UET* for the last time and tries to help the other agent explore the last *east-west* ring left.

□

Lemma 36 *None of the east-west rings will be explored more than once.*

Proof: An agent does not explore an *east-west* ring until it finds a node with the *UET* (one token in the middle of a node). This agent then moves the *UET* to the next node to the *north* of the current node. Then and only then does it explore the *east-west* ring that starts from the node with the *UET*. This shows that all the explored *east-west* rings will not be explored for a second time.

According to Lemma 35, the *UET* advances on the *north-south* ring correctly. Hence, none of the *east-west* rings will be explored more than once.

□

Lemma 37 *At most 1 agent dies in the BH.*

Proof:

According to algorithm *Cross Rings*, before any agent starts procedure “*Bypass on Torus*”, both agents only travel on:

- either one *north-south* ring and all the *east-west* ring. In this case, the *north-south* ring is the *Base* ring. According to the definition of a *Base* ring, the BH can only be on one of an *east-west* ring. Considering all the possible links both agents may traverse, only two links on an *east-west* ring may lead an agent to the BH.
- or an *east-west* ring and all the *north-south* rings. In this case, the *east-west* ring is the *Base* ring. According to the definition of a *Base* ring, the BH can only be on one of a *north-south* ring. Considering all the possible links both agents may traverse, only two links on a *north-south* ring may lead an agent to the BH.

Given the torus is labeled, each *north-south* ring or *east-west* ring is also oriented. If on the one hand, an agent has died in the BH through a link on the last *RUE*, the other agent will start procedure “*Bypass on Torus*” and eventually terminate the algorithm before it traverses the other link.

On the other hand, we assume that both agents are alive and they start *bypassing* each other. It is possible that an agent dies in the BH through a link that connects a *safe* ring and the only *dangerous* ring left. Given *bypass* technique is used to prevent two agents from exploring the same node, none of the other link that leads to the BH will be used again. So, maximum one agent dies in the BH during procedure “*Bypass on Torus*”.

Hence, at most one agent dies in the BH.

□

Lemma 38 *Within finite time one agent will determine the location of the BH.*

Proof: From Lemma 33 and 34 we know that two agents will sooner or later find a *Base* ring and both agents keeps 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 36, 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 37 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.

□

Complexity Analysis

Lemma 39 *Two co-located agents with five (5) tokens in total are sufficient to locate the BH in a labeled torus.*

Proof:

According to Lemmas 37 and 38, 2 agents are sufficient 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 decided, it can pick up the 2 tokens and eventually reuse them.
- Each agent needs one token to do *CWWT* for exploring and for *bypassing* (including the “Back to the *Dangerous* Ring”). As we just mentioned in the previous item, the second agent can reuse the 2 tokens in the middle of their *HB* once the *Base* ring is decided.

- One token is used as a *UET*.
- As we explained in Lemma 35, before one agent picks up the only *UET* 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 . From this fact, we observe that:
 - An agent can use the *CWWT* token to put the second *UET* in node v .
 - An agent can reuse the picked up *UET* (one token) in u to continue the exploration with *CWWT*. Hence, no extra token is needed.

Hence, using two co-located agents, five (5) tokens in total are sufficient to locate the BH in a labeled torus.

□

Lemma 40 $O(n)$ moves is sufficient using algorithm *Cross Rings*.

Proof: In procedure “Find a *Base* ring”, a *Base* ring is decided after agent a_1 explored the *north-south* ring that includes the \mathcal{HB} or a_2 explored the *east-west* ring that includes the \mathcal{HB} . 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 are required. Hence, $O(n)$ moves is sufficient.

In procedure “*Bypass* on 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 A_{i1} steps before it executes procedure “Back to the *Dangerous* Ring — Torus”. For every “Back to the *Dangerous* Ring — Torus”, a_1 uses A_{i2} steps before it walks back to the *dangerous* ring through 1 link. A maximum of y such links are going to be traversed given there is maximum of y nodes on each *east-west* ring. So, it takes an agent $O(n)$ moves, even when using *CWWT*.

The idea of the *bypass* technique is to let an agent visit a node on a *dangerous* ring only through the links on a *safe* ring. Let u and v denote two nodes on the *dangerous* ring. a_1 will either traverse i links on the *dangerous* ring in order to go to from u to v , or a_1 will go through the link that connects to the *safe* ring, then traverse i links on the *safe* ring, then go to v on the *dangerous* ring through 1 link that connects from the *safe* ring to the *dangerous* ring. See Figure 5.12 (The two ways for an agent to go from node u to node v are: 1) Through the links and nodes that are on the *dangerous* ring; 2) Through the links and nodes that are on the *safe* ring.). 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 procedure “*Bypass on Torus*” and “*Back to the Dangerous Ring — Torus*”.

Hence, $O(n)$ moves in total are sufficient for both agents to locate the BH.

□

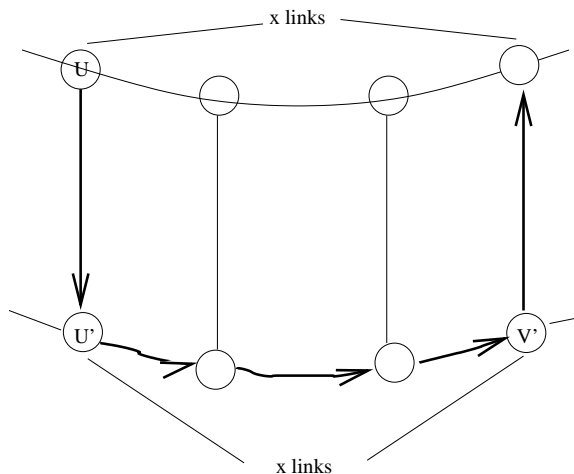


Figure 5.12: The two ways for an agent to go from node u to node v

According to the lemmas proved above, and following the lower bound from the whiteboard model presented in [42], we can conclude:

Theorem 15 *Using two (2) co-located agents and five (5) tokens in total, the BH can be successfully located with $\Theta(n)$ moves in a labeled torus with n nodes.*

5.3 Algorithm Modified ‘Cross Rings’ — The Case of 3 Scattered Agents

5.3.1 Assumptions, Basic Ideas, Observation and General Description

In this section, we study the BHS problem in a labeled torus (labeled the same as described in Subsection 5.2.1) with 3 scattered mobile agents. Here, we will prove that 3 is the minimum team size. The number of nodes n and the dimensions of the torus $x \times y$ are assumed known to all the scattered agents. Again as assumed in Subsection 5.2.1, we do not require that all the links and nodes obey the FIFO rule; but the *CWWT* technique is used throughout this section.

As we explained in Section 5.2, Subsection 5.2.1, it is impossible for an agent to traverse a torus (which, clearly, is necessary for locating the BH), if this agent only goes through the *north* or *south* ports or only goes through the *east* or *west* ports. Also, it is obvious that one agent is not enough to locate the BH given that as soon as this agent wakes up, it can wonder into the BH directly. Our immediate question is: are two agents enough to locate the BH in a labeled torus? We know that 2 agents are enough to solve the BHS problem if they are co-located, according to the investigation presented in the previous section. But in this section, we need to study the BHS problem in a labeled torus with scattered mobile agents. As mentioned in the previous chapters, it is a fact that before scattered agents *meet* in the same node, they can neither agree on the same sense of direction nor divide their common task into sub-tasks (i.e., cooperate). So, if we have 2 scattered agents, if they do not *meet* each other fast enough, it is possible that one dies on a *north* or *south* port and the other agent dies on an *east* or *west* port while they try to achieve the task separately.

Hence we can get the following lemma:

Lemma 41 *Two scattered agents are not enough to locate the BH in a labeled torus.*

We prove later in this section that 3 scattered agents are enough to locate the BH with 7 tokens per agent.

As we know, algorithm *Cross Rings* solves BHS problem in a labeled torus with 2 co-located agents. If we can somehow gather two agents into a pair in one node, then they can start locating the BH with algorithm *Cross Rings* from the node in which they form the pair. Following this idea, we develop Algorithm *Modified ‘Cross Rings’* that allows us to form a pair with two agents and let them locate the BH according to Algorithm *Cross Rings*, to which some modification is required.

In algorithm *Modified ‘Cross Rings’*, an agent starts as a *single* agent. A single agent tries to explore a *north-south* ring and all the *east-west* rings that start from the nodes that are on the *north-south* ring, if it does not die in the BH or become *Passive* before succeeding. Sooner or later, an agent will either explore the entire torus except for the BH (when the BH is located in the node that is to the *south-west* side of this agent) or die in the BH or will *meet* (i.e., see the token of another agent, as we explained in the previous chapter) another agent or become *Passive*. Once two agents *meet*, they form a pair immediately. A paired agent will execute some of the procedures we modify in Algorithm *Cross Rings* and eventually locate the BH. We will describe all the necessary modifications to the algorithm *Cross Rings* in the coming subsections.

From this point on, we assume that the *Base* ring is a *north-south* ring. All the descriptions and procedures are based on this “assumption”. Importantly, if in fact the *Base* ring is an *east-west* ring, the description and procedures can be obtained by exchanging all the key words according to Table 5.1.









	A Paired agent is walking with <i>CWWT</i> to the <i>east</i>
	A Paired agent is walking with <i>CWWT</i> to the <i>north</i>
	A Single agent is walking with <i>CWWT</i> to the <i>east</i>
	A Single agent is walking with <i>CWWT</i> to the <i>north</i>
	An Single agent found another Single agent and formed a pair with it
	<i>Bypass</i> is executing by two agents ----always use the ring to the <i>south</i>
	<i>Base Ring Progress Sign</i> ---- the <i>east-west</i> ring that starts from this node is under exploration. Can be reused as the token for `` <i>Bypass</i> '' procedure
	A <i>Base ring</i> is found

Figure 5.13: Token positions and their meanings.

The meaning of token(s) at different locations can be found in Figure 5.13.

The details of each procedure are given in the following subsections.

5.3.2 Procedures “Initialization” and “Single Agent Explores a *north-south* Ring”

Algorithm Description

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 initializes $xCount$, which is used to record the size of the explored region on a *north-south* ring. Then it explores the *north-south* ring starting from node u where it woke up (we called it a *homebase*(\mathcal{HB}) as we explained in the previous chapters), with $CWWT$ (two tokens on the port). While a_1 explores on the *north-south* ring, each time it explores one more node, it increases $xCount$ by one. As Figure 5.14 shows, during the exploration there are eight possible Cases/Senarios that can happen:

- a_1 goes into a node with one token in the middle of a node. According to Figure 5.13, it is a *UET* (defined in Subsection 5.2.1). This indicates that a pair is formed and the paired agents are executing algorithm *Modified ‘Cross Rings’* as paired agents. So, a_1 becomes *Passive* immediately.
- a_1 goes into a node with two tokens on the *east* port. This means that a single agent is exploring an *east-west* ring to the *east*. So, a_1 moves one token to the *north* port, one token to the middle of the node. Most importantly, it leaves one extra token in the middle of the node. Then it executes “Paired agent finds a *Base* ring” to the *north*.
- a_1 goes into a node with two tokens on the *north* port. This means that a single

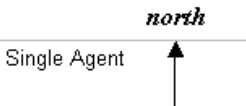





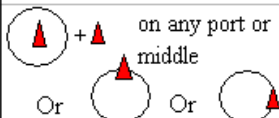
Single agent going to the north	Case 1	Meaning: it is a <i>UET</i> , this indicates that a pair is
		Action: become <i>Passive</i> immediately.
Single Agent	Case 2	Meaning: it meets a Single agent, with which it wants to form a pair
		Action: moves one token to the <i>north</i> port, one token to the middle of the node. Executes "Paired agent finds a <i>Base ring</i> " to the <i>north</i> .
	Case 3	Meaning: it meets a Single agent, with which it wants to form a pair
		Action: moves one token to the <i>east</i> port, one token to the middle of the node. Executes "Paired agent finds a <i>Base ring</i> " to the <i>east</i> .
	Case 4	Meaning: another agent wants to form a pair with
CWWT token is changed into:		
		Action: executes "Paired agent finds a <i>Base ring</i> " to the <i>north</i> .
	Case 5	Meaning: a pair has been formed already
Either of the three situations:		
		Action: becomes <i>Passive</i> immediately
	Case 6	Meaning: it goes back to the node where it started
	"xCount=x"	Action: executes "Single agent exploring a <i>east-west ring</i> ".
	Case 7	Meaning: a pair was formed and one of the paired agent eliminated this single agent.
CWWT tokens are stolen		Action: becomes <i>Passive</i> immediately
	Case 8	Meaning: it goes back to the node where it started
	"nCount" is not empty and none of Case1~6 happens	Action: increase "nCount" and executes "Single agent exploring a <i>east-west ring</i> ".

Figure 5.14: Token configurations and their resulting actions in procedure "Single Agent Explores a north-south Ring".

agent is exploring a *north-south* ring to the *north*. So, a_1 moves one token to the *east* port, one token to the middle of the node. And again, it is very important that a_1 leaves one extra token in the middle of the node. It then executes “Paired agent finds a *Base* ring” to the *east*.

- When a_1 comes back to the node where it left its *CWWT* tokens (two tokens on the *north* port), if there are two tokens in the middle and at least one token on the *east* port of the node, it means that another single agent a_2 encountered a_1 , and it formed a pair with a_1 . When there are two tokens on the *east* port, that means a_2 is exploring the node to the *east* of the current node through the *east* port. a_1 then executes “Paired agent finds a *Base* ring” to the *north*.
- 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:
 1. there is at least one token in the middle of the node (there may be also token(s) on a port of that node).
 2. there is a token on the *north* port.
 3. there is a token on the *east* port.
- When $xCount = x$ happens, while a_1 explores the *north-south* ring, it means that a_1 finished exploring the *north-south* ring. a_1 initializes *nCount* in order to keep track of the number of nodes explored, then executes procedure “Single Agent Explores an *east-west* Ring”.
- When a_1 comes back to the node where it left its *CWWT* tokens (two tokens on the *north* port), if all the *CWWT* tokens are stolen, it means that a paired agent eliminated a_1 . Hence, a_1 becomes *Passive*.

- When $nCount$ is non zero, and none of the previous cases occur while a_1 explores the *north-south* ring, it means that a_1 is intending to explore 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 and increases $nCount$ by one.

Pseudo Code

The pseudo code of procedures “Initialization” and “Single Agent Explores a *north-south* Ring” are in Algorithm 33.

5.3.3 Procedure “Single Agent Explores an *east-west* Ring”

Algorithm Description

As a single agent, there are only two scenarios (Cases 6 and 8 in Figure 5.14) that can trigger a_1 to execute procedure “Single Agent Explores an *east-west* Ring”. In this procedure, a_1 sets $yCount$ to zero and then keeps exploring on an *east-west* ring starting from a node that is on the only *north-south* ring that a_1 finished exploring. While a_1 explores the *east-west* ring, it walks with *CWWT*. Each time a_1 explores one more node, it increases $yCount$ and $nCount$ by one.

As Figure 5.15 shows: during the exploration there are eight possible Cases/Senarios that may occur. Cases 1, 2, 3, 5, 7 are the same as Cases 1, 2, 3, 5, 7 in procedure “Single Agent Explores a *north-south* Ring”. Consequently we now only explain Cases 4, 6 and 8, which are different from the cases in procedure “Single Agent Explores a *north-south* Ring”:

- When a_1 comes back to the node where it left its *CWWT* tokens (two tokens on the *east* port), if there are two tokens in the middle and at least one token on

Algorithm 33 Algorithm *Modified ‘Cross Rings’* — Procedure “Initialization” and “Single Agent Explores a *north-south* Ring”

```

1: INITIALIZATION:(upon initial wake-up in the  $\mathcal{HB}$ )
2:  $xCount = 0$ , execute SINGLE AGENT EXPLORES A north-south RING( $nCount$ )

3: procedure SINGLE AGENT EXPLORES A north-south RING( $nCount$ )
4:   repeat
5:     keep exploring to north with CWWT and increases  $xCount$ 
6:   until A or B or C or D or E or F or G or H happens
7:   if A: into a node with one token in the middle of a node then
8:     become Passive immediately
9:   else if B: into a node with two tokens on the east port then
10:    move one token to the north port, one token to the middle of the node
11:    add one more token to the middle of the node
12:    execute PAIRED AGENT FINDS A Base RING(north)
13:   else if C: into a node with two tokens on the north port then
14:    move one token to the east port, one token to the middle of the node
15:    add one more token to the middle of the node
16:    execute PAIRED AGENT FINDS A Base RING(east)
17:   else if D: go back to the node where you left your CWWT tokens and notice that
    there is a token in the middle and at least one token on the east port of the node then
18:    execute PAIRED AGENT FIND A Base RING(north)
19:   else if E: into a node that either one token in the middle of the node and there are
    also token(s) on a port of that node, or there is only one token on a port then
20:    become Passive immediately
21:   else if F:  $xCount = x$  then
22:      $nCount = 0$ 
23:     execute SINGLE AGENT EXPLORES AN east-west RING( $nCount$ )
24:   else if G: noticing CWWT tokens are stolen then
25:     become Passive
26:   else if H:  $nCount \neq null$  and none of A to G is true then
27:     execute SINGLE AGENT EXPLORES AN east-west RING( $nCount$ )
28:      $nCount = nCount + 1$ 
29:   end if
30: end procedure

```



Single agent going to the <i>east</i>	Case 1	The same as "Single agent exploring a <i>north-south</i> ring" Case 1
Single Agent \rightarrow <i>east</i>	Case 2	The same as "Single agent exploring a <i>north-south</i> ring" Case 2
	Case 3	The same as "Single agent exploring a <i>north-south</i> ring" Case 3
	Case 4	Meaning: another agent wants to form a pair with it
	CWWT tokens are changed into:	
		Action: executes "Paired agent finds a <i>Base</i> ring" to the <i>east</i> .
	Case 5	The same as "Single agent exploring a <i>north-south</i> ring" Case 5
	Case 6	Meaning: it goes back to the node where it started
	"yCount=y"	Action: executes procedure "Single agent exploring a <i>north-south</i> ring".
	Case 7	The same as "Single agent exploring a <i>north-south</i> ring" Case 7
	Case 8	Meaning: it goes back to the node where it started
	"nCount=n-1"	Action: becomes <i>Done</i> . Algorithm terminates, the Black Hole is located.

Figure 5.15: Token configurations and their resulting actions in procedure "Single Agent Explores an east-west Ring".

the *north* port of the node, it means that another single agent a_2 encountered a_1 , and it formed a pair with a_1 . When there are two tokens on the *north* port, that means a_2 is exploring the node to the *north* of the current node through the *north* port. a_1 then executes “Paired agent find a *Base* ring” to the *east*.

- When a_1 notices $nCount = n$, it means that the only node that a_1 did not explore is the BH. a_1 then terminates the algorithm.
- When a_1 comes back to the node where it left its *CWWT* tokens, if all the *CWWT* tokens are stolen, it means that a paired agent eliminated a_1 . Hence, a_1 becomes *Passive*.
- When $yCount = y$ happens, while a_1 explores the *east-west* ring, it means that a_1 finished exploring this *east-west* ring. a_1 then executes procedure “Single agent exploring a *north-south* ring” for one node. If $nCount$ is not empty and none of Cases 1, 2, 3, 4, 5, 6, 7 happen, then a_1 executes Case 8 in procedure “Single agent exploring a *north-south* ring”.

Pseudo Code

The pseudo code of procedure “Single Agent Explores an *east-west* Ring” is in Algorithm 34.

5.3.4 Procedure “Paired Agent Finds a *Base* Ring”

Procedure Description

In this subsection we explain procedure “Paired agent finds a *Base* ring”. The basic idea and general description of this procedure is the same as procedure “Find a *Base* Ring” in algorithm *Cross Rings*. There are two differences between the two procedures:

Algorithm 34 Algorithm *Modified ‘Cross Rings’* — Procedure “Single Agent Explores an *east-west* Ring”

```

1: procedure SINGLE AGENT EXPLORES AN east-west RING( $nCount$ )
2:   repeat
3:     keep exploring to east with CWWT and increase  $yCount$  and  $nCount$ 
4:   until A or B or C or E or I or J or K or L happens
5:   if A: into a node with one token in the middle of a node then
6:     become Passive immediately
7:   else if B: into a node with two tokens on the east port then
8:     move one token to the north port, one token to the middle of the node
9:     add one more token to the middle of the node
10:    execute PAIRED AGENT FINDS A Base RING(north)
11:   else if C: into a node with two tokens on the north port then
12:     move one token to the east port, one token to the middle of the node
13:     add one more token to the middle of the node
14:     execute PAIRED AGENT FINDS A Base RING(east)
15:   else if I: back to the node where you left your CWWT tokens and notice that there
    is a token in the middle and at least one token on the north port of the node then
16:     execute PAIRED AGENT FIND A Base RING(east)
17:   else if E: into a node that either one token in the middle of the node and there are
    also token(s) on a port of that node, or there is only one token on a port then
18:     become Passive immediately
19:   else if J:  $yCount = y$  then
20:     execute SINGLE AGENT EXPLORES A north-south RING( $nCount$ )
21:   else if K: noticing CWWT tokens are stolen then
22:     become Passive
23:   else if L:  $nCount = n - 1$  then
24:     become DONE
25:   end if
26: end procedure

```

- The token(s) representation when the procedure starts are different. All the differences between the two procedures are shown in Figure 5.16.
- Any time during this procedure, as long as an agent sees that there are two tokens on the *east* port or *north* port of a node, it picks up the tokens. This is because two tokens on a port represents a single agent. Hence, we let a paired agent eliminate a single agent by stealing its *CWWT* tokens.

Once an agent 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{HB} 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 procedure “Paired Agent Finds a *Base* Ring” to the *north* (the description and pseudo code for executing the procedure to the *east* can be converted according to Table 5.1 in Subsection 5.4.1) a paired agent a_1 keeps walking to the *north* with *CWWT*, until it goes back to the \mathcal{HB} of this pair. It is possible to have the following two token configurations in this node:

- there is 1 token on the *north* port and two tokens in the middle of their \mathcal{HB} (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)
- there are 2/3 (2 or 3) tokens in the middle of the node.

		First	Agent:	A ₁ (to the north)	Second	Agent:	A ₂
		Colocated	Scattered	Agents	Colocated	Scattered	Agents
		Agents	Case 1	Case 2	Agents	Case 1	Case 2
	Trigger						
	Meaning	I initiate "form pair"	I initiate "form pair"	I initiate "form pair"			
	Action						
	Trigger						
	Meaning	north-south ring is the Base ring	north-south ring is the Base ring	east-west ring is the Base ring			
A ₁ is back before A ₂ wakes up/back	Action						
	Trigger						
	Meaning				the north-south ring is the Base ring	the north-south ring is the Base ring	the east-west ring is the Base ring
	Action:				explores the east-west ring	explores the east-west ring	explores the north-south ring
A ₂ wakes up/back before A ₁ finishes the north-south ring	Trigger						
	Meaning				the east-west ring is the Base ring	the east-west ring is the Base ring	the north-south ring is the Base ring
	Action						
	Trigger						
	Meaning	the east-west ring is the Base ring	the east-west ring is the Base ring	the north-south ring is the Base ring			
	Action	walks until sees the UET, executes procedure "Advance on the Base ring"	walks until sees the UET, executes procedure "paired agent advances on the Base ring"	walks until sees the UET, executes procedure "paired agent advances on the Base ring"			

Figure 5.16: The comparison table of procedures "Find a Base Ring" in algorithm Cross Rings and procedures "Paired Agent Finds a Base Ring" in algorithm Modified 'Cross Rings'.

In the former 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. It is very important to know that a_1 is going to reuse this token as a *UET* in the procedure it then executes: “Paired Agent Explores the *east-west* Rings”. In the latter case, 2 tokens in the middle of the \mathcal{HB} 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. a_1 then keeps walking to the *east* until it sees 1 token (the *UET* as defined earlier) in the middle of a node. It then executes procedure “Paired Agent Advances in the *Base* ring” to the *east* port. If there are 3 tokens in the middle, the third token in the middle is a *UET*. This means a_2 is exploring the first *east-west* ring as a paired agent. Hence, a_1 executes procedure “Paired Agent Advances in the *Base* ring” to the *east* port immediately.

When the second agent a_2 walks back to the \mathcal{HB} of this paired agent after exploring the *east-west* ring, there are either 2 tokens in the middle of the \mathcal{HB} or 3 tokens in the middle of the \mathcal{HB} or 1 token on the *north* port and 2 tokens in the middle of their \mathcal{HB} . In the first case, a_1 informed a_2 that the *north-south* ring is the *Base* ring. a_2 keeps walking to the *north* until it arrives in the node with a *UET* in the middle. It then executes procedure “Paired Agent Advances in the *Base* Ring” to the *north*. In the second case, a_2 sees 3 tokens in the middle. 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 procedure “Paired Agent Advances in the *Base* Ring” to the *north*. 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{HB} . a_2 then executes procedure “Paired Agent Explores the *east-west* Rings” to the *east*.

During the execution of this procedure, there are two other possible scenarios:

- as soon as a_1 or a_2 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 then continue.
- as soon as a_1 or a_2 notice its *CWWT* token is moved, it will execute procedure “Paired Agent *Bypasses*”.

Pseudo Code

The pseudocode of procedure “Paired Agent Finds a *Base* Ring” in algorithm *Modified ‘Cross Rings’* is in Algorithm 35.

5.3.5 The Rest of the Algorithm

As we mentioned in Subsection 5.4.1, the idea of solving the BHS problem with scattered agents is to let two of the three agents to form pairs, then let the paired agents execute algorithm *Cross Rings* starting from the node (their \mathcal{HB}) where they formed a pair. In the previous three subsections, we explained how do the agents form a pair and how a pair of agents finds a *Base* ring. From this point on, the procedures: “Paired Agent Explores the *north-south/east-west* Rings”, “Paired Agent Advances In the *Base* Ring”, “Paired Agent *Bypasses* on Torus” and “Paired Agent Goes back to the *Dangerous* Ring — Torus” used by this pair of agents are almost the same as the procedures: “Explore the *north-south/east-west* Rings”, “Advance In the *Base* Ring”, “*Bypass* on Torus” and “Back to the *Dangerous* Ring — Torus” of algorithm *Cross Rings*.

Beyond the scenarios already discussed in these four procedures in algorithm *Cross Rings*, there is one extra scenario that occurs in each of these four procedures in Algorithm *Modified ‘Cross Rings’*:

- as soon as an agent goes into a node with 2 tokens on any of a port (the

Algorithm 35 Algorithm Modified ‘Cross Rings’ — Procedure “Paired Agent Finds a Base Ring”

```

1: procedure PAIRED AGENT FINDS A Base RING(ForS)
2:   if ForS = F then
3:     repeat
4:       keep walking to the north with CWWT
5:       if there are 2 tokens on a port then
6:         pick up the two tokens
7:       end if
8:     until M or N or P happens
9:     if M: there is 1 token on the north port and 2 tokens in the middle then
10:      // north-south ring becomes the Base ring
11:      pick up the token on the north port
12:      execute PAIRED AGENT EXPLORES THE east-west RINGS(x,yCount) to the
13:      east
14:     else if N: there are 2 tokens in the middle of the node then // east-west ring
15:      becomes the Base ring
16:      keep walking to the east until going into the node with a UET
17:      execute PAIRED AGENT ADVANCES IN THE Base RING(east)
18:     else if P: there are 3 tokens in the middle then
19:      execute PAIRED AGENT ADVANCES IN THE Base RING(east)
20:     end if
21:   else if
22:     then
23:       repeat
24:         keep walking to the east with CWWT
25:         if there are 2 tokens on a port then
26:           pick up the two tokens
27:         end if
28:       until M or N or P happens
29:       if M: there is 1 token on the north port and 2 tokens in the middle then
30:        // east-west ring becomes the Base ring
31:        pick up the token on the north port
32:        execute PAIRED AGENT EXPLORES THE north-south RINGS(y,xCount) to
33:        the north
34:       else if N: there are 2 tokens in the middle of the node then // north-south
35:        ring becomes the Base ring
36:       keep walking to the north until going into the node with a UET
37:       execute PAIRED AGENT ADVANCES IN THE Base RING(north)
38:       else if P: there are 3 tokens in the middle then
39:        execute PAIRED AGENT ADVANCES IN THE Base RING(north)
40:       end if
41:     end if
42:   end procedure

```

indication of a single agent), it will pick up all the tokens and then continue.

Beyond the scenario mentioned above and the scenarios in procedure “*Bypass on Torus*” and “Back to the *Dangerous* Ring — Torus” in algorithm *Cross Rings*, we emphasize one little detail in procedures “Paired Agent *Bypasses on Torus*” and “Paired Agent Goes back to the *Dangerous* Ring — Torus” in algorithm *Modified ‘Cross Rings’*:

- Each time one of the paired agent initiates a *Bypass*, one extra token will be put in the middle of the node in which a *CWWT* token also needs to be moved.

We obtain the four procedures: “Paired Agent Explores the *north-south/east-west Rings*”, “Paired Agent Advances In the *Base* Ring”, “Paired Agent *Bypasses on Torus*” and “Paired Agent Goes back to the *Dangerous* Ring — Torus” by

- First, adding the following lines into each of the four procedures mentioned above in algorithm *Cross Rings*:

Algorithm 36 Algorithm *Modified ‘Cross Rings’* — Extra Lines

```
1: if there are 2 tokens on a port then
2:   pick up the two tokens
3: end if
```

- Second, adding the following line after lines 5 and 10, in the pseudo code of procedure “*Bypass on Torus*”, and after line 25, in the pseudo code of procedure “Back to the *Dangerous* Ring — Torus” in algorithm *Cross Rings*: Algorithm 31:

Algorithm 37 Algorithm *Modified ‘Cross Rings’* — Another Extra Line

```
1: put 1 token in the middle of this node
```

5.3.6 Correctness and Complexity Analysis

Correctness

Lemma 42 *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:

1. dying in the BH; or
2. 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
3. being eliminated by a paired agent (having its *CWWT* tokens stolen).

this single agent 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 the following facts:

1. All three (minimum team size) agents execute the same algorithm.
2. All single agents walk with *CWWT*.
3. 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.
4. Any single agent only leaves through the *north* and/or *east* ports of a node, then, a single agent 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 a 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.

□

According to Lemma 42 that we proved in Subsubsection 5.2.5, we can obtain the following lemma:

Lemma 43 *At least one agent will find a Base ring in the torus.*

Proof: As we explained in Subsubsection 5.3.4, there are only two small changes being made to procedure “Find a *Base* Ring” in order to adapt it to the BHS with scattered agents. The comparison between the two procedures “Paired Agent Finds a *Base* Ring” and “Find a *Base* Ring” is given in detail in Figure 5.16. We can see from the columns: “co-located agent” and “scattered agent, case 1”, that once we give the same meaning to the token configurations of procedures “Paired Agent Finds a *Base* Ring” as those of procedure “Find a *Base* Ring”, then the triggers for the same meaning will get the same action.

Beyond all the functions in procedure “Find a *Base* Ring”, procedure “Paired Agent Finds a *Base* Ring” makes a paired agent pick up all the *CWWT* tokens of a single agent. According to the token configurations and their meanings explained in Figure 5.13, 2 tokens on a port has one and only one meaning, namely, the *CWWT* tokens of a single agent. So, a paired agent will pick up the tokens without causing any problem.

Finally according to Lemma 42, at least one agent will find a *Base* ring in the torus using Algorithm *Cross Rings*. Hence, at least one agent will find a *Base* ring in the torus using algorithm *Modified ‘Cross Rings’*.

□

Lemma 44 *A single agent will not interfere with the progress of any paired agent.*

Proof: In procedure “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 the port.

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

In all the procedures that a paired agent executes, we added “eliminate single agent” steps, as shown in Algorithm 37 in Subsection 5.3.5. 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 45 *The Bypass technique can be correctly executed by a pair of agents.*

Proof: The only modification we did to procedure “*Bypass* on 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 in the previous section, 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 46 *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 chapter 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 37 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 agent die in the BH during Algorithm *Modified 'Cross Rings'*.

□

Lemma 47 *Within finite time one agent will determine the location of the BH.*

Proof:

As we proved in Lemma 43, 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 38 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'*.

□

Complexity Analysis

Lemma 48 *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 Lemma 46 and 47, 3 agents are sufficient to locate the BH.

We now prove that maximum 7 tokens are sufficient in order for three agents to locate the BH.

We have to make a difference between the *CWWT* tokens of a single agent and to a paired agent, because of the following two facts:

- 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, see Lemma 41.
- Algorithm *Cross Rings* locates the BH correctly with 2 co-located agents.

We can either use 2 tokens on port as the *CWWT* tokens of a single agent and 1 token on the port as the *CWWT* token of a paired agent, or vice versa. After analyzing these two choices, we conclude that both use the same number of tokens in total (7 or less). We arbitrarily decided to choose the first of these two alternatives for our algorithm. Now we are going to prove that 7 tokens in total is sufficient.

- 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 ($CWWT$ 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 39). Given a_1 has an extra token and leaves it in the middle of the new \mathcal{HB} , 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 *bypassing* 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 *bypassing*.

Hence, seven (7) tokens in total are sufficient to locate the BH in a labeled torus with scattered agent.

□

Lemma 49 $O(n)$ moves is sufficient using algorithm Modified ‘Cross Rings’.

Proof: We proved in Lemma 42 that within finite time there will be one pair 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 agents. 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’* affects the number of moves. And we know that once an agent becomes a paired agent, $O(n)$ moves is sufficient to locate the BH, according to Theorem 15 in algorithm *Cross Rings*.

Hence, $O(n)$ moves in total is sufficient for the three agents to locate the BH.

□

According to the lemmas proved above, and following the lower bound from the whiteboard model presented in [42], we can conclude:

Theorem 16 *Using three (3) scattered agents and seven (7) tokens in total, the BH can be successfully located using $\Theta(n)$ moves in a labeled torus with n nodes.*

5.4 Algorithm *Single Forward* — The Case of k Scattered Agents

5.4.1 Assumptions

In this section, we study the BHS problem in a labeled torus (labeled the same as described in Subsection 5.2.1) with k ($k > 3$) scattered mobile agents. Here, k is not known to any of the agents. The number of nodes n and the dimension of the torus $x \times y$ in this torus are known to all the scattered agents. In this section, we do require that all the links and nodes obey the FIFO rule. Also, the *CWWT* technique is used through out this section.

We develop a simple algorithm *Single Forward* that can locate the BH with 1 token per agent and $O(k^2n^2)$ moves in total using k scattered agents. The general idea is explained in the following subsection.

5.4.2 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.

Once an agent wakes up, it is a *single* agent. A *single* agent a_s becomes a *forward* agent, when it finds a token on a port which a_s wants to go and it has to go through another *unsafe* port (*east* port). In other words, when an agent arrives to a node, if further progress is blocked (at least one of the *unsafe north/east* ports is *blocked (with token)*), that port becomes a *Check Point* for that agent. If there is no other *unsafe north* or *east* port available (*without token*), a *single* agent remains its state and wait in the node after putting one token in the middle. a_s continues as a *single* agent if the port it wants to go 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 of these two *Check Points*. 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:

- *single* agent — has no *Check Point*, explores and becomes forward if blocked;
- forward agent — has one *Check Points*, explores and becomes checker if blocked

again;

- checking agent — has two *Check Points*, sits at one *Check Point* and upon any change checks the other *Check Point*. Becomes a *forward* agent when one of the *Check Points* unblocks *without token*, a *single* agent if both of them unblocked.

Beyond this high-level description, there are some details for each procedure. Procedures “Single Agent” and “Forward Agent” are described in the following two subsections.

5.4.3 Procedures “Initialization” and “Single Agent”

Detailed Description

Upon waking up, an agent a_1 executes procedure “Single Agent” of the algorithm. In procedure “Single Agent” a_1 explores the *north-south* ring starting from its \mathcal{HB} , using *CWWT* with one token on a port. a_1 always goes for an unexplored node reachable from its current location using only *north* and *east* links. If we call ‘first choice’ the port that a_1 attempts to go through during its normal exploration, we will call ‘second choice’ the other port that eventually leads to another unexplored node. During the exploration, a_1 keeps counting the number of nodes it explored so that a_1 can terminate the algorithm once it explored $n - 1$ nodes.

The exploration continues until a_1 goes into a node with one token on the ‘first choice’ port. If there is also a token on the ‘second choice’ port, then a_1 waits in the node as a *single* agent. Once the ‘first choice’ port becomes *without token*, then a_1 continues its exploration through that port as a *single* agent. Once the ‘first choice’ port becomes *without token*, then a_1 puts a token on that port and becomes a *forward* agent.

Pseudo Code

The pseudo code of procedures “Initialization” and “Single Agent” is given in Algorithm 38.

Algorithm 38 Algorithm *Single Forward* — Procedure “Initialization” and “Single Agent”

```

1: INITIALIZATION:(upon initial wake-up in the  $\mathcal{HB}$ )
2: execute SINGLE AGENT( $nCount$ )

3: procedure SINGLE AGENT( $nCount$ )
4:   repeat
5:     keep exploring the next unexplored node reachable from its current location using
       only north and east links, keep counting the nodes explored in  $nCount$ 
6:   until A: arrive in a node with the ‘first choice’ port blocked; or B:  $nCount = n - 1$ 
       happens
7:   if A happens then
8:     if there is also a token on the ‘second choice’ port then
9:       repeatwait in the node
10:      until either the ‘first choice’ or ‘second choice’ port becomes unblocked
11:      if the ‘first choice’ port becomes unblocked then
12:        execute SINGLE AGENT( $nCount$ )
13:      else
14:        puts a token on that port, remember it as a Check Point and execute
        FORWARD AGENT( $nCount$ )
15:      end if
16:    else
17:      puts a token on that port, remember it as a Check Point and execute FOR-
        WARD AGENT( $nCount$ )
18:    end if
19:  else if B happens then
20:    become DONE, the last node without being explored is the BH
21:  end if
22: end procedure

```

5.4.4 Procedures “Forward Agent” and “Checking Agent”

Algorithm Description

A *forward* agent a_f continues exploring the next unexplored node reachable from its current location using only *north* and *east* links until it goes into its next *Check Point* u . If the ‘second choice’ port is *without token*, a_f remembers this second *Check Point*, then put a token on the ‘second choice’ port and continues its exploration through this port as a *forward* agent. Otherwise, if both *north* and *east* ports are blocked, a_f becomes a *checking* agent immediately.

If a *checking* agent a_c notices that there is no token in the middle of u , then a_c waits in the node until either the ‘first choice’ or the ‘second choice’ port become *without token* or the number of tokens in the middle changes. When a ‘Choice’ port becomes *without token*, then a_c continues accordingly. When there is a change in the number of tokens in the middle of the node, then a_c goes back to the previous *Check Point* to check the availability. If there is at least one token in the middle of u , it then immediately goes back to its previous *Check Point* v . If v is empty, then a_c becomes a *single* agent if v is the first *Check Point*. Otherwise, a_c continues as a *forward* agent. In either case, it keeps exploring the torus accordingly. If v is not empty, then a_c leaves a token in the middle of the node and goes back to u . If now u still has at least one token in the middle, then a_c goes back to v and waits there. But if there is no token in the middle of u any more, then a_c goes back to v to pick up its token from the middle and return to u . Once both *Check Point* become unblocked, a_c will become a *single* agent immediately and continue the exploration. If only one of the *Check Point* is unblocked, a_c will continue the exploration as a *forward* agent.

Pseudo Code

The pseudo code of procedures “Forward Agent” and “Checking Agent” is in Algorithm 39 and 40.

Algorithm 39 Algorithm *Single Forward* — Procedure “Forward Agent”

```

1: procedure FORWARD AGENT( $nCount$ )
2:   repeat
3:     keep exploring to the next unexplored node from the current node through either
       the north port of the east with CWWT and increase  $nCount$ 
4:   until see another Check Point  $v$ 
5:   if the ‘second choice’ port is not blocked then
6:     put a token on that port, remember this Check Point, then execute FORWARD
       AGENT( $nCount$ )
7:   else if the ‘second choice’ port is also blocked then
8:     if there is no token in the middle of the node then
9:       repeat
10:        wait in the node
11:       until either E or F happens
12:       if E: there is a change in the number of tokens in the middle then
13:         execute CHECKING AGENT( $nCount$ )
14:       else if F: a port is unblocked then
15:         remember this node as another Check Point; execute FORWARD
       AGENT( $nCount$ )
16:       end if
17:     else
18:       execute CHECKING AGENT( $nCount$ )
19:     end if
20:   end if
21: end procedure

```

5.4.5 Correctness and Complexity Analysis

Lemma 50 *The algorithm progresses correctly.*

Proof: According to the definition of *single* agent and *forward* agent, an agent is a *single* agent as long as it continues its exploration without meeting any other agent in its normal exploration route. Once a *single* agent has to give up the current exploration route and choose another unexplored node, it becomes a forward agent.

Algorithm 40 Algorithm *Single Forward* — Procedure “Checking Agent”

```

1: procedure CHECKING AGENT( $nCount$ )
2:   go back to your previous Check Point  $v$ 
3:   if there is a token on the ‘first choice’ port in  $v$  then
4:     leave a token in the middle of the node and go back to  $u$ 
5:     if  $u$  still has at least one token in the middle then
6:       go back to  $v$  and waits there
7:     else if there is no token in the middle of  $u$  then
8:       if both north and east ports are with token then
9:         goes back to  $v$  to pick up your token from the middle and return to  $u$ 
10:        execute FORWARD AGENT( $nCount$ )
11:       else if the ‘first choice’ port is without token then
12:         execute FORWARD AGENT( $nCount$ )
13:       else
14:         remember this node as another Check Point; execute FORWARD
AGENT( $nCount$ )
15:       end if
16:     end if
17:   else if there is no token in  $v$  then
18:     execute SINGLE AGENT( $nCount$ )
19:   end if
20: end procedure

```

This change of route is caused by being blocked (i.e., stops exploring and after seeing a token on the port it wants to take) by another agent. A *forward* agent will either wait in a blocked (*with token*) node or continue its exploration when a ‘choice’ port becomes *without token*.

From procedure “Check the Previous *Check Point*”, we can see that it is impossible to have all the forward agents being blocked in one node. When there are two or more agents being blocked, at least two Check Points must each have a forward agent in it. Given there is only one BH, either *Check Point* will eventually be empty. The agent in it will be able to continue its exploration. In finite time, at least one node will be explored by one agent. Hence, the algorithm progresses correctly.

□

Lemma 51 *Within finite time at least one agent will survive and determine the location of the BH.*

Proof:

Lemma 50 proves that, in finite time, at least one node will be explored by one agent. According to our algorithm, such exploration continues until one agent explores $n - 1$ nodes. Given there is only one BH, and an agent can only go through a *north* or *east unsafe* port, so, at most 2 agents will die in the BH. Also, we assumed that there are k ($k > 3$) agents in this torus. Hence at least one agent survives. The surviving agent keeps exploring until one of the surviving agent explored $n - 1$ nodes. This agent then terminates the algorithm immediately. The only node left without being explored is the BH.

□

Lemma 52 *One token per agent suffices using algorithm Single Forward to locate the BH.*

Proof: In algorithm *Single Forward*, there is no communication (message exchanging) between the agents. The only use for tokens is to complete the *CWWT*. There is no need to differentiate a *single* agent and a *forward* agent. So, one token per agent allows the agents to finish the task correctly.

□

Lemma 53 *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 (i.e., whether it still has tokens in it) of that node. 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 preceding 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.

□

According to the above lemmas, the following theorem follows:

Theorem 17 *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.*

Chapter 6

Black Hole Search in Complete Networks

6.1 Topological Characteristics

A complete graph is an undirected graph with an edge between every pair of vertices. This means every node has a direct connection (i.e., there is only 1 link between the two nodes) to any other node in the graph. Let \mathcal{K}_n denote the complete network with n nodes. Then \mathcal{K}_n has $n(n-1)/2$ edges and is a regular graph of degree $\Delta = n-1$. A complete graph/network is maximally connected because the only vertex cut that disconnects the graph is the complete set of vertices. The positive consequence of this observation is that an agent can traverse the whole graph using a very simple algorithm with only $\Theta(n)$ moves (see the figure on the left of Figure 6.1).

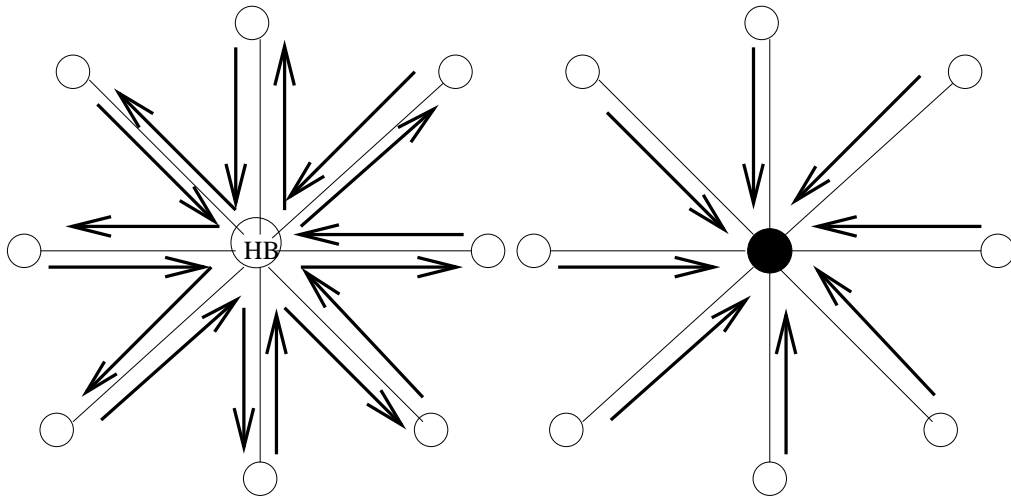


Figure 6.1: Left: The simplest way for an agent to traverse a unlabeled complete network. Right: $n-1$ agents wander into the BH once they wake up.

Unfortunately, if the agents are scattered, even if there are $n - 1$ agents, one in each non-BH node, it is possible that they all walk to the BH as soon as they wake up (see the figure on the right of Figure 6.1). According to Observation 1, at least n scattered mobile agents are needed to locate the BH.

6.2 Assumption, Basic Technique and Observations

In this chapter, we are going to study the BHS problem in a complete network. A team of anonymous co-located agents is used to solve the problem in Section 6.3. A team of anonymous scattered agents is used to solve the problem in Section 6.4. In both cases, the complete network is anonymous and the FIFO rule is not required on links or on nodes. We will assume n is known to all the agents. Unlike most of the algorithms in this dissertation, the two algorithms we develop in this chapter do not use the *CWWT* introduced in Chapter 3.

We remark that a complete graph is a loop graph [105]. Any complete graph with more than two vertices is Hamiltonian. We know we can locate the BH with $O(n \log n)$ moves, according to the BHS with tokens on the ring topology. Since a complete network has full connectivity, clearly it has a chordal ring⁸ and thus that we can solve BHS in $\Theta(n)$ moves with an algorithm using 2 agents and one token in total if the agents are co-located. Indeed, we develop a very simple algorithm called *Take Turn* to locate the BH using co-located agents in a complete network. Algorithm *Take Turn* is presented in Section 6.3. For the scattered agents case, we prove later that n or more scattered agents can locate the BH in an unoriented complete network with 1 token per agent and maximum of n^2 moves in total.

⁸A chordal ring is an augmented ring, or a circulant graph with a chord of length 1 [6].

6.3 BHS in a Complete Network with Co-located Agents

In this section, we are going to study the BHS problem in a un-oriented complete network, that is, one in which the links are assumed to be undirected. However, it is important to note that, even without orientation, co-located agents may agree on an order of traversal for the links of their \mathcal{HB} . More precisely, when agents are co-located, they share a common reference (e.g., indexing) mechanism for the $n - 1$ links of their \mathcal{HB} and thus can share a common order of traversal of these links. For simplicity, we will say the links are traverse '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 co-located agents.). A team of two anonymous 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{HB} of a pair of co-located agents) in the middle, see Figure 6.2.

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{HB} . a_1 then explores the node reachable from this port. When a_1 comes back (we assume that a_1 is able to remember the port that it just visited.) to its \mathcal{HB} 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 finds the port with one token, moves this token to the next port clockwise (we assumed that the orientation can be agreed between all the co-located agents) and starts exploring another node through this port. During this process, an agent keeps counting (using variable $nCount$) the number of ports it visited (i.e.,

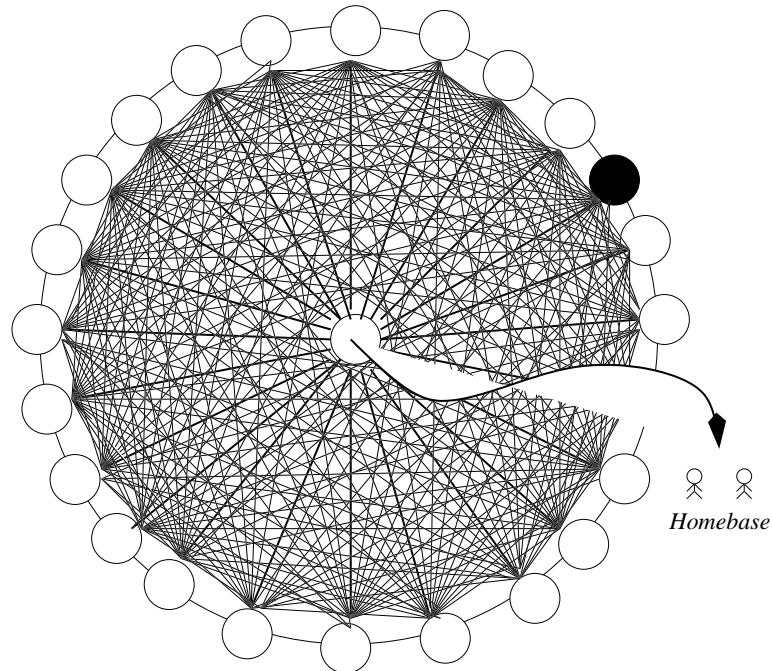


Figure 6.2: *The Star Shape Complete Network of 24 nodes with the \mathcal{HB} of a pair of co-located agents in the middle*

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 another variable $bhCount$ to record the location of the BH.

When an agent a_i ($i=1$ or 2) takes over the token from the other agent a_j (when $i = 1, j = 2$, when $i = 2, j = 1$) that is exploring a new node, it is not clear whether a_j died in the BH or is just exploring a node through a slow link. So, each time an agent a_i moves the token used by partner a_j to the next port, a_i resets the variable $bhCount$ to 0, then keeps increasing it by one each time it explores a new node. Recall that 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 $bhCount$ indicates the location of the BH: the $bhCount^{th}$ port counter clockwise leads to the BH.

6.3.1 Algorithm *Take Turn*

Algorithm 41 Algorithm *Take Turn* — Procedure “Initialization” and “Take Turn”

```

1: procedure INITIALIZATION
2:   wake up,  $bhCount = 0$ ,  $nCount = 0$ 
3:   execute TAKE TURN( $bhCount, nCount$ ) clockwise
4: end procedure
5: procedure TAKE TURN( $bhCount, nCount$ )
6:   if there is no token on any port in the node then
7:     choose a port randomly, put a token on it
8:   else
9:     move the token to the next port clockwise,  $bhCount ++$ ,  $nCount ++$ 
10:  end if
11:  repeat
12:    go to the next node via this port then come back through the same link
13:    if there is no token on that port then
14:       $bhCount = 0$ 
15:      count the number of ports clockwise and increase  $nCount$  until you see a
token
16:      move this token to the next port clockwise,  $bhCount ++$ ,  $nCount ++$ 
17:      execute TAKE TURN( $bhCount, nCount$ ) clockwise
18:    else
19:      move the token to the next port clockwise,  $nCount ++$ 
20:       $bhCount ++$  if  $bhCount \neq 0$ 
21:      execute TAKE TURN( $nCount, bhCount$ ) clockwise
22:    end if
23:  until  $nCount = n - 1$ 
24:  become DONE, the  $bhCount^{th}$  port counter clockwise leads to the BH.
25: end procedure

```

6.3.2 Correctness and Complexity

The left figure in Figure 6.1 shows that any complete network has a subgraph [69] that allows one node connection to all the other nodes in a complete network. Figure 6.3 shows that there are $n - 1$ links connected to the \mathcal{HB} of these 2 co-located agents. Let \mathcal{S}_n denote such a subgraph of complete graph \mathcal{K}_n .

Lemma 54 *Each link in \mathcal{S}_n will be traversed once and by one and only one agent.*

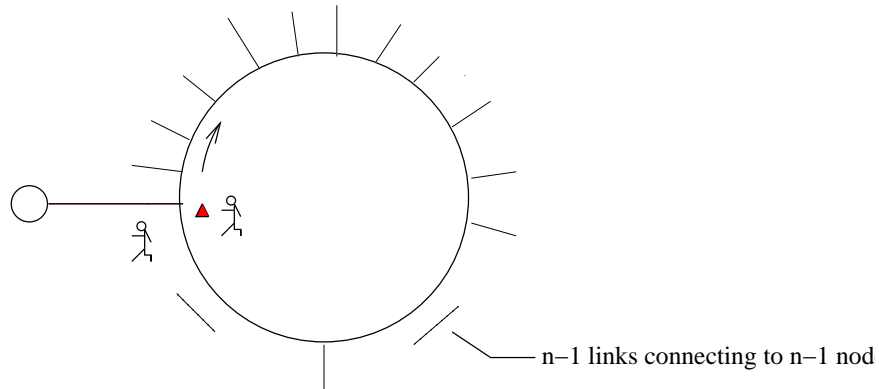


Figure 6.3: The Star Shape Complete Network with the \mathcal{HB} of a pair of co-located agents in the middle

Proof:

From algorithm *Take Turn* we observe:

- An agent does not leave a token unless there is no token in its \mathcal{HB} .
- An agent does not go to explore a node before it leaves a token on the port leading to that node.
- An agent only moves a token to another port if there is a token in the node and the token is on a port.
- An agent increases variable $nCount$ by one after visiting a port or passing every port between the last visited port and the port with a token clockwise.
- An agent terminates the algorithm as soon as the variable $nCount$ reaches $n-1$.

So, no link will be traversed more than once. Hence, each link in \mathcal{S}_n will be traversed once by one and only one agent.

□

Lemma 54 and the fact that there is only one link in this subgraph that leads to a BH, clearly leads to Lemma 55, which is trivial to prove.

Lemma 55 *There is at most one agent that dies in the BH using Algorithm Take Turn.*

Lemma 56 *The surviving agent will locate the BH correctly.*

Proof: From Lemma 56, we know that one agent survives. Once an agent a_1 dies in the BH, the other agent a_2 will move the token to the next port clockwise. Variable $bhCount$ is reset to 0 and is used by a_2 to record the location of the port that leads to the BH. Each time a_2 moves the token to another port, it increases both $nCount$ and $bhCount$ by one. a_2 continues until $nCount$ reaches $n - 1$. a_2 terminates the algorithm immediately. The $bhCount^{th}$ port counter clockwise is the port that leads to the BH. Hence, the surviving agent will locate the BH correctly. □

Lemma 57 *Two agents can locate the BH within n moves.*

Proof: Given the size of the complete network is n , there are $n - 1$ nodes (including the BH) connecting to the \mathcal{HB} of the two co-located agents. According to Lemma 54, each link will be explored once and only once. So, at most $2(n - 1)$ moves are required to locate the BH. Hence $O(n)$ moves suffice for the two agents to locate the BH. □

According to the above Lemmas, and following the lower bound from the white-board model presented in [42], we conclude:

Theorem 18 *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.*

6.4 BHS in a Complete Network with Scattered Agents

6.4.1 Some Basic Observations, Assumptions and Conclusions

According to Observation 1, in this section we use:

- a team of n scattered agents for BHs in a complete network.

Our immediate question is: are n scattered agents enough to locate the BH? If so, how many tokens do they need?

We remark that, in order to locate the BH, there is a simple solution:

Upon one agent waking up, it leaves a token in the middle of its \mathcal{HB} and waits. This agent start executing algorithm *Take Turn* as soon as its token is moved to a port of its \mathcal{HB} . 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{HB} , it becomes *Passive* immediately. Eventually, maximum of $n/2$ pair 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 19 *Using n or more 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 .*

Proof: According to the above description, only the agents that have at least two agents will execute algorithm *Take Turn*. We know that an agent that executes algorithm *Take Turn* will go back to its \mathcal{HB} as soon as it explores a non-BH. So, whether a node is *with* or *without* token(s) will not affect the behavior of this agent. It will executing algorithm *Take Turn*, ignoring the tokens in any other nodes but its \mathcal{HB} . Eventually at least one agent will locate the BH correctly. Given a pair of agent execute n moves during the algorithm, and there are maximum $n/2$ such pairs,

$O(n^2)$ moves are executed during the entire algorithm. Hence after $O(n^2)$ moves, n scattered agents with one token each will locate the BH in a unoriented complete network.

□

Chapter 7

Conclusion

7.1 Recapitulation

A Black Hole is a highly harmful host that disposes of visiting agents upon their arrival. It is known that it is possible to locate a BH with co-located agents in an anonymous network using a whiteboard in each node. Also, in the same model (using whiteboards) a solution that uses scattered agents has been presented only for the ring topology.

In this dissertation, we have developed a set of token-based algorithms for locating a BH in four kinds of networks: ring, hypercube, torus and complete network. For these four topologies we obtained solutions not only for co-located agents, but also for scattered ones as well. We believe that this set of algorithms constitute a significant improvement to the state of the art for this problem. Let us elaborate:

First, we observe that it is rather unrealistic to have at least $O(\log n)$ bits of local storage available all the time to agents to access through fair mutual exclusion, but this is the basic assumption of whiteboard model. Also, several such algorithms rely on face-to-face recognition, which is not easily realizable in practice. Consequently, obtaining a solution to the BH search problem that requires neither local storage nor face-to-face recognition constitutes, in our opinion, a significant improvement to the state of the art for this problem. We present such a solution, which is based on the token model. It is important to note that this token model imposes more constraints on the BH search problem than the whiteboard model does, since it requires that

both local storage and face-to-face recognition be avoided.

Our second contribution is two-fold. On the one hand, we claim that, costwise (in terms of memory and number of moves), the algorithms we developed show the token model to be as efficient as the whiteboard model [45, 46, 49]. This is important given the fact that the token model is more constrained than the whiteboard one. On the other hand, we contend that investigating the token model separately for each of the four-abovementioned topologies results in algorithms that are more efficient than a general, topology-independent, one.

Indeed, we proved that the algorithms we designed for co-located agents achieve a better number of moves than:

1. the generic (i.e., topology-independent) algorithm for an arbitrary network using a whiteboard model, published in [46]. The following results were presented in [46]: with topological ignorance $\Delta + 1$ agents are needed and suffice, and the cost is Θn^2 , where Δ is the maximal degree of a node and n is the number of nodes in the network; with topological ignorance but in presence of sense of direction, only two agents suffice and the cost is Θn^2 ; and with complete topological knowledge only two agents suffice and the cost is $\Theta n \log n$. All the upper-bound proofs are constructive.
2. a general solution for an unknown graph using the token model presented in [44] (A general solution for an unknown graph with $\Delta + 1$ mobile agents, $O(\Delta^2 M^2 n^7)$ moves is presented (where, M is the number of edges in the graph, n is the number of nodes in the graph, and Δ is the maximum number of degree of the graph) .

In other words, we conclude that, for the BHS problem, topology-specific solutions outperform a topology-independent one. Thus, our second contribution is to suggest

that, for BHS, topology-specific token-based algorithms are as powerful (i.e., able to solve the problem) and efficient as those based on a whiteboard model.

Third, we generalize our results by considering two flavors of the BH search problem, namely: using either co-located or scattered agents. In the latter case, since scattered agents start from different *homebases*, they may not agree on a same sense of direction when the network is unoriented. Also, contrary to co-located agents, the scattered agents cannot immediately participate in an extensive exploration of the topology at hand (since they first must be paired). This greatly increases the difficulty of the BHS problem. Indeed, even if using a whiteboard model, BHS with scattered agents has only been considered in the ring topology [44]. Thus solving the BH search problem in several topologies using scattered agents constitutes, in our opinion, a significant contribution.

Last, we have studied several factors that affect performance (namely: team size, token cost, knowledge of team size, sense of direction and connectivity of the network topology) and we have briefly discussed the trade-offs between them. Further investigation of these performance factors should considerably help the researcher to choose the best strategy to solve the BHS problem under different environment constraints.

In this dissertation we believe our contributions directly addressed three open problems:

1. solving the BH search problem while avoiding whiteboards.
2. establishing whether known bounds for the BHS problem can be improved by considering specific network topologies (as opposed to not making assumptions about topology).
3. solving the BHS problem with scattered mobile agents.

We now present a more detailed summary of our contributions.

In the co-located agents case, BHS is indeed solvable in the token model [44, 49]. In particular, in [49] we showed that a team of two or more co-located agents can solve BHS with $O(n \log n)$ moves and two (2) tokens per agent in a *ring* network.

Without requiring the FIFO rule, we proved that:

- using two (2) co-located agents, two (2) tokens in total and $\Theta(n)$ moves, the BH can be successfully located in a labeled hypercube.
- using two (2) co-located agents and five (5) tokens in total, the BH can be successfully located with $\Theta(n)$ moves in a labeled torus.
- using two (2) co-located agents and one (1) token in total, the BH can be successfully located in a complete network without sense of direction, with $\Theta(n)$ moves in total.

The problem became considerably more difficult when the agents are *scattered*. In particular, with scattered agents, the presence (or lack) of orientation in the network topology and knowledge of the team size were observed to be important factors. Here, an “oriented” network topology is taken to be one in which all the agents are able to agree on a common sense of direction. Conversely, an “unoriented network topology” means the agents may not be able to agree on a common sense of direction. In the ring topology, the following results were obtained: in [52], we showed that a team of two or more scattered agents can solve BHS with $O(n \log n)$ moves and five (5) tokens per agent in a *ring* network when the orientation is known. Furthermore, in [53], we showed that a team of three or more scattered agents can solve BHS with $O(n^2)$ moves and four (4) tokens per agent in a *ring* network when the orientation is unknown. But given one more agent (4 scattered agents in total), BHS can be solved with $O(n \log n)$ moves and four (4) tokens per agent when the orientation is still unknown. For the three other network topologies, still without requiring the FIFO rule, we proved that:

- using three (3) scattered agents and seven (7) tokens in total, the BH can be successfully located using $\Theta(n)$ moves in a labeled torus.
- using k scattered agents and 1 token per agent, the BH can be successfully located using $O(k^2n^2)$ moves in a labeled torus.
- using n scattered agents and one (1) token per agent, the BH can be successfully located in a complete network with $O(n^2)$ moves without requiring the sense of direction.

7.2 Comparative Evaluation

We have showed that by taking into account the specific properties of each topology in a token model, the complexity of the general algorithm (for an arbitrary network) can be considerably improved in an algorithm designed for a specific network topology. Here, we will compare the similarities and differences between the results obtained for the specific topologies and analyze the impact of topology and other performance factors on the BHS problem.

In Figure 7.1 and 7.2, we listed the fourteen algorithms presented in this dissertation. From these two figures we can make the following observations:

1. when we use co-located agents to solve BHS problem:
 - minimum team size (2 agents) is achieved on all four topologies.
 - the token cost is inversely proportional to the connectivity of the topology.
 - 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.
 - as 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							
		The case of Co-located Agents			The Case of Scattered Agents				
		Divide with Token	Divide with Token +	Divide with Token -	Gather Divide	Pair Elimination	Shadow Check without CWWT	Shadow Check without CWWT	Modified Shadow Check without CWWT
Orientation		No	No	No	Yes	Yes	No	No	No
FIFO		Yes	Yes	Yes	Yes	Yes	Yes	No	No
uses CWWT		Yes	Yes	Yes	Yes	Yes	Yes	No	No
Knows k		No	No	No	Yes	No	No	No	No
Team Size		$2 +$	$2 +$	$2 +$	$2 +$	$2 +$	$3 +$	$3 +$	$4 +$
Token Cost		$\log n$	$5/\text{agent}$	3 when $k = 2$ or $2/\text{agent}$ if $k > 2$	$1/\text{agent}$	$4/\text{agent}$	$1/\text{agent}$	$5/\text{agent}$	$5/\text{agent}$
Move Cost		$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$O(kn + n \log n)$	$\Theta(n \log n)$	$O(n^2)$	$O(n^2)$	$\Theta(n \log n)$

Figure 7.1: Comparative evaluation table — Ring.

	Complete Network		Hypercube	Torus	
	Co-located Agents	Scattered Agents	Co-located Agents	Co-located Agents	Scattered Agents
	Take Turn	Modified 'Take Turn'	Two Rings	Cross Rings	Modified 'Cross Rings'
					Single Forward
Orientation	No	No	No	No	Yes
FIFO	No	No	No	No	Yes
uses <i>CWWT</i>	No	No	Yes	Yes	Yes
Knows k	No	No	No	No	No
Team Size	2	n	2	2	3 or more
Token Cost	1 in total	1 per agent	1 per agent	5 in total	7 in total
Move Cost	$\Theta(n)$	$O(n^2)$	$\Theta(n)$	$\Theta(n)$	$O(k^2 n^2)$

Figure 7.2: Comparative evaluation table — Complete Network, Hypercube and Torus.

ring topology still has the highest cost (both for number of tokens and number of moves).

2. when we use scattered agents to solve the BHS problem:
 - under the same assumptions (see Figure 7.1 columns 5 and 6), the move cost is improved with sense of direction and bigger token cost.
 - under the same assumptions (see Figure 7.1 columns 8 and 9), the move cost is improved with a larger team size.
 - under the same assumptions (see Figure 7.1 columns 7 and 8), the FIFO constraint can be removed at the cost of using more tokens.
 - the token cost is proportional to the connectivity among the following three interconnected topologies: Torus, Hypercube and Complete Network.
3. within one topology, using scattered agents always requires more agents than using co-located agents.
4. for co-located agents, the cost of solving the BHS problem in the Ring topology is the worst.
5. unlike using co-located agents, for scattered agents agents, the cost of solving the BHS problem become even worth, then the connectivity is stronger.

7.3 Future Work

Before we sketch out a research plan, it is essential that we emphasize the difficulties inherent to complex distributed algorithms. For the purpose of illustration, let us now describe briefly what could appear to be an improved solution to the case of scattered agents for torus. Despite an apparent possible correctness, we want to focus on a specific situation in which this algorithm does not work correctly.

The basic idea is to have agents in one of two basic states: “Single” or “Forward”. Each agent tries to explore the whole torus on its own, with occasional help from the agents it *bypasses*. An agent explores the torus using *CWWT*. It explores a *north-south* ring going *north*, then it moves one step *east* and repeats this process. Once an agent wakes up, it is a *single*, and each *single* agent knows that all the nodes from its \mathcal{HB} to its current location are *safe*. A single agent a_1 becomes a *forward* agent when it finds a token of another *single* agent a_2 (which, according to *CWWT*, blocks a_1 's further progress). In such a case, a_1 leaves a message for a_2 indicating that a_2 now has an associated *forward* agent, and a_1 continues to explore the rest of the *north-south* rings after *bypassing* a_2 (and the possibly *dangerous* node it is visiting).

This message from a_1 to a_2 essentially forms a virtual bond (or partnership) between these agents. The goal of a_1 , as a *forward* agent, is to explore as much as possible, while having one and only one ‘partner’ a_2 (which was exploring a possibly *dangerous* node). If a_1 finds that the next edge it wants to use is blocked (i.e., *with token*) by another agent a_3 , then a_1 will wait until either the edge becomes unblocked (i.e., *without token*), or a_2 comes to inform a_1 that the node a_1 *bypassed* was not the BH after all. Agent a_1 remains a *forward* unless either of these situations occurs. Also, here, a_3 can be just a *single* agent, or a *single* agent that has an associated *forward* agent, or a *forward* agent.

Given this high level description, two immediate questions were raised:

1. how does a *single* agent a_2 , that has an associated *forward* agent a_1 , know how to find a_1 ?
2. how to ensure that the agent that a_2 finds is indeed the agent a_1 that *bypassed* a_2 (as opposed to some other agent)?

In order to address these questions, we invented a technique to ensure that no

other agent can ‘cut in’ on a segment between a pair of agents. But upon verification, we found that, while this technique could handle most scenarios, one infrequent situation created a problem. Without going in details, because pairs could be created anywhere in the torus, interactions between such pairs could alter the token protocol and invalidate our would-be solution. As is often the case in complex distributed algorithms (especially using tokens), it is the coordination between co-occurrent token protocols that presents a significant challenge.

Consequently, the point to be grasped is that the verification of complex distributed algorithms largely remains, in our opinion, an open problem. It is this observation that is at the root of the future research we will now propose.

Our most immediate and obvious goal is to find a solution to the use of scattered agents in a hypercube (if any is possible). At this point in time we have in fact explored several possible solutions. But we have found that all conceal hard to find special cases that invalidate our current ideas. Regardless, we hope to find a solution in the medium term. We also want to improve our solution for scattered agents for the torus. Assuming these two goals are achievable, we could then conduct an in-depth analysis of the similarities and differences between this ‘family’ of algorithms. Once, and only once such an analysis has been carried out, we would like to address eventually the following issues:

- First, the FIFO requirement epitomizes the difficulty of simulating complex distributed algorithms. It would have been highly desirable to include several simulations as a verification mechanism in this dissertation. However, the more complex cases (e.g., in which enforcing FIFO is essential) would require considerable effort to create. Put another way, placing a set of concurrent agents each in a specific state is highly challenging in any topology. More generally, it is widely accepted that controllability and observability remain challenges for the

verification of distributed algorithms. (Hence in this dissertation, our verification method has consisted in proofs based on the numerous pencil and paper examples we have developed.) We intend to tackle the challenge of simulating our algorithms in the medium term (though we expect this will first require the selection of an appropriate simulation environment, as well as the development of some method for generating tests for our proposed algorithms).

- Second, we studied lower bounds only for the ring topology. Clearly, the lower bounds of our algorithms for Complete Network, Hypercube and Torus need to be eventually researched. This should be feasible in the short term.
- Finally, in this dissertation we have considered the BHS problem 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 the BHS in the case of multiple black holes appears to depend directly on the 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 (as a solution will likely depend on a deep understanding of the result of this dissertation).

Bibliography

- [1] S. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Trans. Comp.*, 38(4):555–66, 1989.
- [2] S. Albers and M. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29:1164–1188, 2000.
- [3] S. Alpern, V. Baston, and S. Essegaiier. Rendezvous search on a graph. *Journal of Applied Probability*, 36(1):223–231, 1999.
- [4] S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous*. Kluwer, 2003.
- [5] B. Alspach, J. Bermond, and D. Sotteau. *Decomposition into cycles I: Hamilton decompositions in Cycles and Rays*. Kluwer Academic, 1990.
- [6] B. Arden and H. Lee. Analysis of chordal ring network. *IEEE Transaction on Computers.*, c-30(4):291–295, 1981.
- [7] J. Aubert and B. Schneider. Decomposition of $km + kn$ into hamiltonian cycles. *Discrete Mathematics*, 37:19–27, 1981.
- [8] I. Averbakh and O. Berman. A heuristic with worst-case analysis for minimax routing of two traveling salesmen on a tree. *Discr. Appl. Mathematics*, 68:17–32, 1996.
- [9] B. Awerbuch, M. Betke, and M. Singh. Piecemeal graph learning by a mobile robot. *Information and Computation*, 152:155–172, 1999.
- [10] L. Barriere, P. Flocchini, P. Fraigniaud, and N.Santoro. Capture of an intruder by mobile agents. In *Proc. of 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA'02)*, pages 200–209, 2002.
- [11] L. Barriere, P. Flocchini, P. Fraigniaud, and N.Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems*, page to appear, 2007.
- [12] V. Baston and S. Gal. Rendezvous search when marks are left at the starting points. *Naval Research Logistics*, 38:469–494, 1991.
- [13] M. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *Proc. of 30th ACM Symp. on Theory of Computing (STOC'98)*, pages 269–287. IEEE Computer Society Press, 1998.
- [14] M. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proc. of 35th Symp. on Foundations of Computer Science (FOCS'94)*, pages 75–85, 1994.
- [15] S. Bhatt and I. Ipsen. How to imbed trees in hypercubes. Technical Report Res. Rep. 443, Dep. Comput. Sci., Yale Univ, 1985.
- [16] L. Bhuyan and D. Agrawal. Generalized hypercube and hyperbus structures for a computer network. In *IEEE Trans. Comput.*, vol. C-33, pages 323–333, 1984.
- [17] P. E. Black. Gray code. in *Dictionary of Algorithms and Data Structures* (U.S. National Institute of Standards and Technology), <http://www.nist.gov/dads/HTML/graycode.html>, January 2007.
- [18] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. of 19th Symposium on Foundations of Computer Science (FOCS'78)*, pages 132–142, 1978.

- [19] M. Blum and W. Sakoda. On the capability of finite automata in 2 and 3 dimensional space. In *Proc. of 18th Symposium on Foundations of Computer Science (FOCS'77)*, pages 147–161, 1977.
- [20] B. Bodlaender. Distributed computing on transitive networks: The torus. In *Proc. of 6th Annual Symposium on Theoretical Aspects of Computer Science STACS'89*, pages 294–303, 1989.
- [21] L. Budach. On the solution of the labyrinth problem for finite automata. *Elektronische Informationsverarbeitung und Kybernetik*, 11:661–672, 1975.
- [22] L. Budach. Environments, labyrinths and automata. In *Proc. of 1977 International Foundations Computational Theory (FCT'77)*, pages 54–64, 1977.
- [23] L. Budach. Automata and labyrinths. *Math. Nachrichten*, pages 195–282, 1978.
- [24] D. Chess. Security issues in mobile code systems. In *Proc. of 1998 Conf. on Mobile Agent Security (MAS'98)*, LNCS 1419, pages 1–14, 1998.
- [25] P. Chong-Dae and C. Kyung-Yong. Hamiltonian properties on the class of hypercube-like networks. *Information processing letters (Inf. process. lett.)*, 91(1):11–17, 2004.
- [26] E. Cockayne and A. Thomason. Optimal multi-message broadcasting in complete graphs. In *Utilitas Mathematica*, volume 18, pages 181–199, 1980.
- [27] C. Cooper, R. Klasing, and T. Radzik. Searching for black-hole faults in a network using multiple agents. In *Proc. of 10th Int. Conf. on Principles of Distributed Systems (OPODIS'06)*, pages 320–332, 2006.
- [28] W. Coy. Automata in labyrinths. In *Proc. of 1977 International Foundations Computational Theory (FCT'77)*, pages 65–71, 1977.
- [29] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in tree networks. In *Proc. of 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, pages 35–45, 2004.
- [30] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. *Fundamenta Informatica.*, 71(2-3):229–242, 2006.
- [31] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability and Computing*, page to appear, 2007.
- [32] S. Das, P. Flocchini, A. Nayak, and N. Santoro. Distributed exploration of an unknown graph. In *Proc. of 12th International Coll. on Structural Information and Communication Complexity (SIROCCO'05)*, pages 99–114, 2005.
- [33] J. Demers and P. Labute. Distributed computing using moe. <http://www.chemcomp.com/journal/moesmp.htm>.
- [34] X. Deng and C. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.
- [35] D. Deserale. The arrowhead torus: a cayley graph on the 6-valent grid. Technical Report 2814, Institut national de recherche en informatique et en automatique, March 1996.
- [36] A. Dessmark, P. Fraigniaud, and A. Pelc. Deterministic rendezvous in graphs. In *Proc. of 11th European Symposium on Algorithms (ESA'03)*, pages 184–195, 2003.
- [37] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. In *Proc. of 10th European Symposium on Algorithms (ESA'02)*, pages 374–386, 2002.
- [38] D. Deugo. Mobile agents for electing a leader. In *Proc. of 4th Int. Symposium on Autonomous Decentralized System (SADS'99)*, pages 324–324, 1999.

- [39] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51:38–63, 2004.
- [40] K. Diks, E. Kranakis, and A. Pelc. Broadcasting in unlabeled tori. *Parallel Processing Letters*, 8(2):177–188, 1998.
- [41] S. Dobrev. Communication-efficient broadcasting in complete networks with dynamic faults,. *Theory Comput Syst.*, 36(6):695–709, 2003.
- [42] S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, and N. Santoro. Optimal search for a black hole in common interconnection networks. *Networks*, 47:61–71, 2006.
- [43] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Multiple agents rendezvous in a ring in spite of a black hole. In *Proc. of 7th International Conference on Principles of Distributed Systems (OPODIS'03)*, pages 34–46, 2003.
- [44] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Exploring a dangerous unknown graph using tokens. In *Proc. of 5th IFIP International Conference on Theoretical Computer Science (TCS'06)*, pages 169–180, 2006.
- [45] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, to appear, 2007.
- [46] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agent protocols. *Distributed Computing*, to appear, 2007.
- [47] S. Dobrev, P. Flocchini, and N. Santoro. Improved bounds for optimal black hole search in a network with a map. In *Proc. of 10th International Coll. on Structural Information and Communication Complexity (SIROCCO'04)*, pages 111–122, 2004.
- [48] S. Dobrev, P. Flocchini, and N. Santoro. Cycling through a dangerous network: a simple efficient strategy for black hole search. In *Proc. of 26th International Conference on Distributed Computing Systems (ICDCS'06)*, page 57, 2006.
- [49] S. Dobrev, R. Kralovic, N. Santoro, and W. Shi. Black hole search in asynchronous rings using tokens. In *Proc. of 6th Conference on Algorithms and Complexity (CIAC'06)*, pages 139–150, 2006.
- [50] S. Dobrev and P. Ruzicka. Linear broadcasting and $n \log \log n$ election in unoriented hypercubes. In *Proc. of 4th Coll. on Structural Information and Communication Complexity (SIROCCO'97)*, pages 53–68, 1997.
- [51] S. Dobrev and P. Ruzicka. Broadcasting on anonymous unoriented tori. In *Proc. of 24th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'98)*, pages 50–62, 1998.
- [52] S. Dobrev, N. Santoro, and W. Shi. Locating a black hole in a ring using tokens: The case of scattered agents. In *Proc. of 13th International Euro-Par Conference, European Conference on Parallel and Distributed Computing (Euro-Par'07)*, page to appear, 2007.
- [53] S. Dobrev, N. Santoro, and W. Shi. Scattered black hole search in an oriented ring using tokens. In *Proc. of 9th Workshop on Advances in Parallel and Distributed Computational Models (APDCM'07)*, page to appear, 2007.
- [54] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *Transactions on Robotics and Automation*, 7(6):859–865, 1991.
- [55] O. Esparza, M. Soriano, J. Munoz, and J. Forne. Host revocation authority: A way of protecting mobile agents from malicious hosts. In *Proc. of Int. Conf. on Web Engineering (ICWE'03)*, LNCS 2722, pages 289–292, 2003.

- [56] P. Flocchini, M. Huang, and F. Luccio. Contiguous search in the hypercube for capturing an intruder. In *Proc. of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, pages 62–62, 2005.
- [57] P. Flocchini, M. Huang, and F. Luccio. Decontamination of chordal rings and tori. In *Proc. of 8th Workshop on Advances in Parallel and Distributed Computational Models (APDCM'06)*, page 8, 2006.
- [58] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Multiple mobile agent rendezvous in a ring. In *Proc. of 6th Latin American Theoretical Informatics Symp (LATIN'04)*, pages 599–608, 2004.
- [59] P. Flocchini and B. Mans. Optimal election in labeled hypercubes. *Journal of Parallel and Distributed Computing*, 33(1):76–83, 1996.
- [60] P. Flocchini, B. Mans, and N. Sanatory. Sense of direction: definition, properties and classes. *Networks*, 32(3):165–180, 1998.
- [61] P. Flocchini, B. Mans, and N. Santoro. Sense of direction in distributed computing. *Theoretical Computer Science*, (291):29–53, 2003.
- [62] P. Flocchini, A. Roncato, and N. Santoro. Backward consistency and sense of direction in advanced distributed systems. *SIAM J. Computing*, 32(2):281–306, 2003.
- [63] P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. In *Proc. of 6th Latin American Theoretical Informatics Symp (LATIN'04)*, pages 141–151, 2004.
- [64] P. Fraigniaud and D. Ilcinkas. Digraph exploration with little memory. In *Proc. of 21st Symp. on Theoretical Aspects of Computer Science (STACS'04)*, pages 246–257, 2004.
- [65] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theor. Comput. Sci.*, 345(2-3):331–344, 2005.
- [66] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM J. on Computing*, 7:178–193, 1978.
- [67] F. Gray. Pulse code communication. (U. S. Patent 2 632 058), March 17, 1953.
- [68] M. Greenberg, J. Byington, and D. G. Harper. Mobile agents and security. *IEEE Commun. Mag.*, 36(7):76–85, 1998.
- [69] E. Gyuri. On division of graphs to connected subgraphs. In *Proc. of 5th Hungarian Combinatorial Colloquium (HCC'76)*, pages 485–494, 1976.
- [70] B. Hanane and P. Flocchini. Optimal map construction of an unknown torus. *International Journal of Foundation of Computer Science*, page to appear, 2007.
- [71] F. Heath. Origins of the binary code. *Scientific American*, 227(2):76–83, 1972.
- [72] D. Hillis. *The Connection Machine*. MIT. Press, 1985.
- [73] F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *Proc. of 1981 Foundations Computational Theory (FCT'81)*, pages 433–444, 1981.
- [74] F. Hohl. A model of attacks of malicious hosts against mobile agents. In *Proc. of 4th Workshop on Mobile Object Systems and the ECOOP Workshop on Distributed Object Security (WMOS'98)*, LNCS 1603, pages 105–120, 1998.
- [75] F. Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. *Lecture Notes in Computer Science*, 1419:92–113, 1998.
- [76] F. Hohl. A framework to protect mobile agents by using reference states. In *Proc. of the 20th Int. Conf. on Distr. Computing Systems (ICDCS'00)*, pages 410–417, 2000.

- [77] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. *24(4):223–232*, 1991.
- [78] S. Kekkonen-Moneta. Torus orientation. *Distrib. Comput.*, 15(1):39–48, 2002.
- [79] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Structural Information and Communication Complexity*, 3499:200–215, 2005.
- [80] E. Korach, S. Moran, and S. Zaks. Optimal lower bounds for some distributed algorithms for a complete network of processors. *Theoretical Comput. Sci.*, 64(1):125–132, 1989.
- [81] A. Kotzig. Every cartesian product of two circuits is decomposable into two hamiltonian circuits. Technical report, Centre de Recherche Mathematiques, Montreal, 1973.
- [82] D. Kozen. Automata and planar graphs. In *Proc. of 1979 Foundations Computational Theory (FCT'79)*, pages 243–254, 1979.
- [83] D. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, 1992.
- [84] E. Kranakis and D. Krizanc. Distributed computing on anonymous hypercube networks. *J. Algorithms*, 23(1):722–729, 1997.
- [85] E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Mobile agent rendezvous in a ring. In *Proc. of Int. Conf. on Distributed Computing Systems (ICDCS'03)*, pages 592–599, 2003.
- [86] T. Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes*. M.I.T. Press, 1992.
- [87] B. Mans. Optimal distributed algorithms in unlabelled tori and chordal rings. Technical Report JCU-CS-96/4, January 1996.
- [88] B. Mans and N. Santoro. Optimal fault-tolerant leader election in chordal rings. In *Proc. of 24th International Symposium on Fault Tolerant Computing (FTCS'94)*, pages 392–403, Austin, Texas, 1994. IEEE Computer Society Press.
- [89] B. Mans and N. Santoro. Optimal elections in faulty loop networks and applications. *IEEE Transactions on Computers*, 47(3):286–297, 1998.
- [90] J. Mir and J. Borrell. Protecting mobile agent itineraries. In *Proc. of 5th Conf. on Mobile Agents for Telecommunication Applications (MATA'03)*, volume 2881 of Lecture Notes in Computer Science, pages 275–285. Springer Verlag, 2003.
- [91] H. Muller. Automata catching labyrinths with at most three components. *Elektronische Informationsverarbeitung und Kybernetik*, 15(1):3–9, 1979.
- [92] S. Ng and K. Cheung. Protecting mobile agents against malicious hosts by intention spreading. In *Proc. of 1999 Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*.
- [93] R. Oppliger. Security issues related to mobile code and agent-based systems. *Computer Communications*, 22(12):1165–1170, 1999.
- [94] P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33:281–295, 1999.
- [95] M. Rabin. Maze threading automata. Technical report, University of California at Berkeley, Seminar Talk, October 1967.
- [96] J. Rattner. The new age of supercomputation. *Distributed Memory Computing, Lecture Notes in Computer Science*, 487:1–6, 1991.
- [97] H. Rollik. Automaten in planaren graphen. *Acta Informatica*, 13:287–298, 1980.

- [98] Y. Saad and M. Schultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, 37(7):867–872, 1988.
- [99] T. Sander and C. Tschudin. Protecting mobile agents against malicious hosts. *Lecture Notes in Computer Science*, 1419:44–60, 1998.
- [100] N. Santoro. *Design and Analysis of Distributed Algorithms*. Wiley, 2006.
- [101] C. Shannon. Presentation of a maze-solving machine. In *Proc. of 8th Conf. of the Josiah Macy Jr. Found (Cybernetics)*, pages 173–180, 1951.
- [102] W. Shi and J.-P. Corriveau. An executable model for a family of election algorithms. In *Proc. of 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, pages 178–186, 2004.
- [103] W. Shi and J.-P. Corriveau. System family engineering in leader election in the ring topology. In *Proc. of 16th International Association of Science and Technology for Development (IASTED'04)*, pages 398–403, 2004.
- [104] G. Singh. Leader election in complete networks. In *Proc. of 11th Annual ACM symposium on Principles of distributed computing (PODC'92)*, pages 179–190, 1992.
- [105] S. Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. MA: Addison-Wesley, 1990. Section 5.3.4 Hamiltonian Cycles, pages 196–198.
- [106] G. Tel. Linear election for oriented hypercube. Technical Report TR-RUU-CS-93-39, Utrecht University, Department of Computer Science, The Netherlands, 1993.
- [107] S. Unger. A computer oriented toward spatial problems. In *Proc. of 1958 Innovating Regions Europe (IRE'58)*, pages 1744–1750, 1958.
- [108] J. Villadangos, A. Cordoba, F. Farina, and M. Prieto. Efficient leader election in complete networks. In *Proc. of 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'05)*, pages 136–143, 2005.
- [109] J. Vitek and G. Castagna. Mobile computations and hostile hosts. *Mobile Objects*, pages 241–261, 1999. University of Geneva.
- [110] E. Weisstein. Hamiltonian circuit. *From MathWorld — A Wolfram Web Resource*, <http://mathworld.wolfram.com/HamiltonianCircuit.html>.
- [111] A. Wu. Interconnection networks for high-performance parallel computers. *IEEE Computer Society Press*, pages 532–543, 1994.
- [112] X. Yu and M. Yung. Agent rendezvous: A dynamic symmetry-breaking problem. In *Proc. of 23th Int. Coll. on Automata Languages and Programming (ICALP'96)*, pages 610–621, 1996.