EFFICIENT MECHANISMS FOR EXPLORATION OF DANGEROUS GRAPHS AND FOR INTER-AGENT COMMUNICATION

Balasingham Balamohan

A thesis submitted to
the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for a doctoral degree in Computer Scienece

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abstract

This thesis deals with the problems of exploration and map construction of a dangerous network by mobile agents, and it introduces new general mechanisms for inter-agent communication, which could be applied to other mobile agents' problems.

A dangerous network contains a harmful process called *Black Hole* that destroys all agents entering the node where it resides, without leaving any observable trace. The task for the agents, which are moving asynchronously, is to construct a map of the network with edges incident on the black hole unambiguously identified. Two types of communication mechanisms are considered: *whiteboards* and *tokens*. In the whiteboard model every node provides a shared memory on which agents can read and write. When communication occurs through tokens, instead, the agents have some pebbles that can be placed on and picked up from the nodes. Four different costs for comparing the efficiency of the protocols are taken into account: the number of agents required, the number of moves performed, the size of the whiteboard (or the token capacity at a node), and time. The black hole search problem is considered first in ring networks with whiteboards, and optimal exact time and move complexities are established improving all existing results. The same problem is then studied in arbitrary unknown graphs and it is solved in the token model by using a constant number of tokens in total. The protocol improves on existing results and is based on a novel technique for communicating using tokens. Finally, the new method of communicating using tokens described in the context of black hole search is generalized to propose a novel communication mechanism among the agents that could possibly be employed for any distributed algorithm by mobile agents.

Acknowledgements

My first thanks goes to my supervisor Prof. Paola Flocchini and co-supervisor Prof. Nicola Santoro for all their advice and guidance. I also express my thanks to Student Academic Success Service of University of Ottawa for their guidance all through my stay at University of Ottawa.

Chapter 1

Introduction

The thesis deals with distributed computing by mobile entities on a network, and in particular with the problem of deploying a team of agents to discover a dangerous node. The main focus of the thesis is on mechanisms for inter-agent communication for this task and, in general, for any distributed task.

More specifically, the problem of exploration and map construction of dangerous networks is investigated using both classical communication techniques and a novel communication method. This new method is then generalized to design a general communication mechanism based on tokens.

In this chapter some motivations for the problem are provided followed by a summary of the contributions and by a brief description of the contents of the thesis.

1.1 Motivation

A distributed computing system is a collection of computational entities that can communicate among them. Computer networks, Internet, and grids are some examples of distributed systems. A distributed algorithm or protocol is a well defined set of behavioral rules that are followed by the entities in a distributed system. Usually distributed protocols are designed so that the collection of computational entities can cooperatively perform tasks. Cooperation is achieved through communication. Two of the most common models of communication used in the distributed algorithms community are shared memory and message passing model. In the shared memory model entities in the network communicate by writing and reading from shared memory areas. In the message passing model, messages are passed along the links of the networks i.e., communication capable physical connections.

A relatively recent addition to the distributed system models is the mobile agent model. A mobile agent is a computational entity than can migrate from place to

place in the environment supporting it. For example, cookies are mobile agents that migrate between the computers on the Internet. Distributed computing using mobile agents is sometimes termed *distributed mobile computing*, and has recently attracted extensive research as a computing paradigm, as well as a system supported programming platform. In distributed mobile computing the nodes of the networks are hosts; local processes are stationary agents; mobile agents are the processes that move from host to neighboring host through the links of the network, and perform computations at each host, according to a predefined set of identical behavioral rules called a protocol.

One of the principal concerns in these systems is security [19,60,73,79]. Among severe security threats is the problem of harmful host (that is, the presence at a network node of a malicious stationary process). How to protect agents from the presence of a malicious host has been intensively studied from a programming perspective (e.g., see [16, 60–62, 64, 72, 77] in a general setting and [24, 75, 82] in specific environments). The *Black hole* is a type of a harmful host: a host that destroys all visiting agents upon their arrival, leaving no observable trace of such a destruction [34, 38]. The task of identifying a black hole has been studied using an algorithmic approach (e.g., see [20, 22, 31, 32, 34, 38, 39, 50, 59, 63, 81]). After the identification of the location of such harmful host an action to remedy the situation can be carried out. The task of unambiguously determining and reporting the location of the black hole is called *black hole search*(Bhs). This problem can model a network where one of the nodes has completely failed or is under a malicious attack preventing one of the node from communicating with the rest of the network.

In a network a black hols could represent an IP address not assigned to a host or a disconnected router. Data packets sent to such an IP address (or to the disconnected router) would be discarded and lost. More generally, black holes model non detectable crash failures.

Hence, efficient solutions to the black hole search is crucial.

The natural algorithmic interest in the Bhs problem is amplified by the fact that this problem opens the more general research question of exploration, search, and map construction of *dangerous* graphs. The algorithmic goal when solving the black

hole search problem is to minimize the *size* of the team deployed (i.e., the number of agents), the *cost* of the search (i.e., the number of moves performed by the agents), and possibly, the *time* spent in the search and resources for inter-agent communication.

Traditionally, three types of inter-agent communication mechanisms have been used. They are as follows:

1. **Whiteboard**: In this model, agents communicate by writing on and reading from shared memory areas at each node.

2. **Tokens**: Agents communicate using pebbles they can pick up, carry and/or drop off on nodes.

3. **Face-to-Face**: In this mechanism, agents can recognize each other when on the same node and communicate among them when on the same node.

Clearly there is a correspondence between whiteboard mechanism and token mechanism. A token capacity of C at node i.e., maximum number of tokens that can placed on a node correspond to whiteboard of size $\lceil \log_2(C+1) \rceil$ bits. One of the goals of the mobile agents' algorithms is to limit the resources needed for the inter-agent communication. It is crucial since on networks there could be several team of agents performing different tasks.

1.2 Contributions

The first part of the thesis focuses on Black Hole Search (BHS). The black hole search problem is considered first in the ring with the classical whiteboard communication mechanism, and then in an arbitrary topology using a new communication mechanism based on tokens.

For BHS in a *ring* with *whiteboards*, the focus is on two complexity measures: *time* and number of *moves*. The optimal existing solution for a ring of size n in terms of worst case time complexity employs $n-1$ agents, $2(n-2)$ time units (both in the worst and in the average case), and $O(n^2)$ moves. In the thesis, the time bounds are improved as follows:

- It is shown that $n-1$ agents can solve the problem in average time $\frac{7}{4}n+O(1)$ and almost optimal worst case time (2 units from optimal), improving the existing bound of $2(n-2)$.

- A lower bound of $\frac{3}{2}n - O(1)$ average time is derived, showing that any asynchronous black hole search algorithm for rings requires that much regardless of the number $k > 1$ of agents. It is also shown that, with $2(n-1)$ agents, this average case complexity can be achieved maintaining an optimal worst case time complexity.

- It is proven that it is possible to locate the black hole in asymptotically optimal (worst and average) $\Theta(n)$ time with just $k = 2$ agents. In fact, an algorithm is described that spends $8n + O(1)$ time in the worst case and $\frac{15}{2}n + O(1)$ on the average, employing an optimal team of 2 agents.

Still in the context of the *ring* with *whiteboards*, the best existing solution in terms of worst case move complexity employs 2 agents, in $O(n \log_2 n)$ time (both in the worst and in the average case), and $O(n \log_2 n)$ moves. More precisely, the exact complexity of that solution is $2n \log_2 n + O(n)$. In the thesis, the exact optimal move complexity is derived improving the existing bound. More specifically:

- It is shown that $3n\frac{\log_2 n}{\log_2(3)} - O(n)$ moves are necessary in the worst case. A novel algorithm is then described that allows two agents to locate the black hole with at most $3n\frac{\log_2 n}{\log_2(3)} + O(n)$ moves, improving the existing upper bounds, and matching the lower bound up to the constant of proportionality.

- It is shown how to modify the proposed protocol so to achieve also asymptotically optimal time complexity $\Theta(n)$, still with $3n\frac{\log_2 n}{log_2(3)} + O(n)$ moves and only two agents; this improves upon all existing time-optimal protocols, which require $O(n^2)$ moves and sometimes more than two agents.

The thesis then considers Black hole search in an *arbitrary unknown network* where the agents communicate by manipulating *tokens*. BHS in arbitrary unknown network has been studied in the whiteboard model and in the enhanced token model. In the

whiteboard model, the best known algorithm solves BHS in $O(n^2)$ moves and uses whiteboards of size $O(\log \delta)$ bits at nodes of degree δ. In the enhanced token model, the best known algorithm solves BHS with $O(\Delta^2 m^2 n^7)$ moves for a network with n nodes, m links and maximum degree Δ. In this thesis for networks with n nodes, m links, and maximum degree Δ :

- It is shown that when only a constant number of tokens is available, at least $\Delta + 2$ agents are necessary for solving the BHS.
- When only 2 tokens are available then BHS is unsolvable regardless of the team size.
- BHS can be solved using $\Delta + 2$ agents and 3 tokens with $O(nm)$ moves. This shows for the first time how to solve the problem in an unknown topology in the token model with a *constant* number of tokens.

The last chapter of the thesis deals with the design of general communication mechanisms introducing a novel technique of communication among agents, which is a generalization of the one employed for the location of the black hole in an arbitrary network. This technique is based on the idea that the agents use dedicated nodes (also called *rooms*) for communicating. Communication occurs by placing tokens on those nodes and by picking them up. Several communication protocols are presented, first for the particular cases of two communicating agents, and then for the case of broadcast among any number of agents. In particular, in the case of communication between two agents it has been shown that:

- When only 1 token is available, regardless of the number of rooms, any communication using a bounded number of moves is impossible.
- When 2 tokens and only 1 room are available, any bidirectional communication using a bounded number of moves is impossible.
- In all the other cases, bidirectional communication is possible and the communication time of the protocols depends on the amount of rooms and tokens available.

In the case of broadcast, similar results have been obtained:

- When only 1 token is available, regardless of the number of rooms, or when there are 2 tokens in 1 room, any communication using a bounded number of moves is impossible.

- In all the other cases, bidirectional communication is possible and the communication time of the protocols depends on the amount of rooms and tokens available.

1.3 Organization of the Thesis

The thesis is organized as follows:

Chapter 2 reviews the existing literature on the graph exploration, map construction, and the BHS in networks under various communication mechanisms.

In Chapter 3 the main definitions and the terminology used in the rest of the thesis are presented, the various agent models are described, and the basic limitations and techniques in the existing literature on black hole search are discussed.

Chapter 4 and Chapter 5 deal with the ring network in the whiteboard model: Chapter 4 focusing on time complexity and Chapter 5 on move complexity. In Chapter 4 several algorithms are designed to improve the existing time complexities bounds. The results of the Chapter 4 have been published in [7, 9].

In Chapter 5 an exact and tight move complexity bound is established improving the current bounds. The best Black Hole Search algorithm for the ring is described at the end of Chapter 5, where the proposed solution simultaneously achieves exact move complexity, team size, and asymptotically optimal time complexity. The results of the Chapter 5 have appeared in preliminary form in [8].

Chapter 6 considers again the problem of BHS, but in an arbitrary topology. Various bounds are derived on the number of tokens and on the number of agents necessary and sufficient to solve the problem. The solution employs as primitives some new token-based communication protocols. In fact the agents make use of two special nodes (the home base and the waiting room) where they perform appropriate

token manipulations to exchange information and coordinate their actions. The messages communicated using tokens are all of polynomial length. Most of the results of Chapter 6 have appeared in preliminary form in [6] and are submitted to a Special Issue of the Journal *Theoretical Computer Science*.

Chapter 7 introduces novel methods by which mobile entities can communicate by picking up and dropping tokens on nodes of the network. This method generalizes the one used in the Chapter 6 by employing several nodes (rooms) and several tokens for communicating general information. The Chapter contains various communication protocols depending on the number of available rooms and tokens for communication between two and more agents.

In Chapter 8, we discuss future work and conclude the thesis.

Chapter 2

Related Work

In this chapter, the literature on the black hole search and the related problems and communication techniques used to solve them are reviewed. We begin by presenting an overview of the research on graph exploration and map construction. Then we conclude this chapter with a discussion of the literature on black hole search.

2.1 Graph Exploration and Map Construction

There are two main variants of the exploration. They are the computational entity (a) must traverse all the edges in the networks (edge variant) and (b) must visit all the nodes of the network (node variant). Earlier works focused on the maze and the planar graph exploration. Mazes are two dimensional structures, that can be modeled by subset of planar graphs, described as follows:

1. A vertex is a point with integer coordinates. An edge is a line connecting adjacent vertices. Observe vertex and edge referred to are that of the structure in two-dimensional plane and not that of the underlying graph.

2. The cells are the squares of unit area enclosed by edges. Any two cell sharing an edge are said to be adjacent.

3. There is a function f on the cells that can take two values. For a cell c, $f(c)$ is either "black" or "white". The function f satisfies that any pair of cells c_1 and c_n with $f(c_1)$ and $f(c_n)$ both "white" there is at least a sequence of cells $c_1, c_2, \cdots c_n$ such that for all $1 \leq i < n$

   (a) $f(c_i)$ is "white".

   (b) c_i and c_{i+1} are adjacent.

More recent works focus on more specific topologies such as trees, rings as well as more general directed and undirected graphs. Algorithmic solutions are distinguished as offline or online. The online algorithms solve the problem based on the local knowledge and the offline algorithms solve the problem based on the centralized perspective. There are three distinctions made on the terminating conditions for the graph exploration. They are

1. Perpetual exploration: The computational entity must explore perpetually.

2. Exploration with stop: This requires that the computational entity must stop the exploration within finite time after completion of the exploration.

3. Exploration with return: This is in addition to satisfying the condition for "Exploration with stop", the computational entity must return to where it started within finite time.

Usually the measure of efficiency used is number of edge traversals or measures based on that such as the competitive ratio. The competitive ratio is the ratio of the cost of the online algorithm being studied to the cost of an optimal offline algorithm.

2.1.1 Maze Exploration

Initial investigation on exploration focused on maze exploration. Shannon in [80] describes a machine that can solve the maze problem. In [15], Blum and Kozen consider the problem of searching the maze. The task is to visit every cell of the maze. The solution presented is for finite automata. The authors show that with additional tools of counter or pebbles or as a team of two, finite automata can explore every maze. Without additional tools maze exploration can be done in log-space and a map of the maze can be constructed in linear space. The authors of [15] also study the problem of exploring planar graphs. They show that even though mazes and planar graphs are related the results become impossible for the planar graphs. In particular it is shown no three automata can explore all cubic planar graphs. Their technique is subsequently formalized as the concept of sense of direction in [53, 78].

2.1.2 Directed Graph Exploration and Map Construction

Deng and Papadimitriou in [28] study the strongly connected directed graph exploration by robots that can recognize visited nodes and know how many unexplored edges emanate from those nodes. A single robot explores the network. The problem studied is the exploration with stop of the edge variant of the problem. They show that complexity is related to the deficiency of the graph where the deficiency is defined as the minimum number of edges needed to be added to the graph to make the graph Eulerian. The authors establish that graph with deficiency $d > 1$ can be explored with $O(d^{d\log(d)}m)$ moves and $d = 1$ can be explored with $4m$ moves where m is the number of edges. Albers and Henzinger in [1] present algorithm which is more efficient. Their algorithm with the complexity of $m(d+1)^6 d^{2\log d}$ achieves sub-exponential complexity. They also show a lowerbound $d^{\Omega(\log d)}$.

In [13], exploration of strongly connected graph with a team of two robots is studied. The algorithm uses homing sequence that is learnt by actively wandering through the graph. Their algorithm has a complexity of $O(d^2n^5)$ where d is the diameter and n is the number of nodes of the network,

Bender et al. in [12] focus on exploration of a strongly connected graph by a single robot endowed with tokens that it can place on and pick up from the nodes. They show that if an upperbound $\hat{n}$ on the number nodes n is known then a single token suffice. They also show that by searching on possible upperbound $\hat{n}$ for the number nodes n that robot can solve the problem using $O(\log(\log(n)))$ tokens when there is no information about the size of the network. The complexities of algorithms are $O(\hat{n}^2n^6d^2)$ in both cases where d is the diameter.

Fraiginaud and Ilcinkas in [56] investigate the exploration of strongly connected directed graphs of n nodes with out-degree bounded by Δ. They study two models of computational entities. The first one consists of a finite automaton with access to pebbles (called robot); the second model has constant memory and access to a whiteboard (agent). For their robot model they show that robots must be equipped with at least $\Omega(n\log(\Delta))$ memory for the perpetual exploration of the directed graph. They present two algorithms for the robot model. First algorithm is executable by robots with memory of size $O(n\Delta\log(n))$ and endowed with a pebble. But the number

of moves made is exponential. Second algorithm achieves polynomial complexity on the number of moves. It is executable by the robots endowed with $O(n^2\Delta\log(n))$ memory and $O(\log(\log(n)))$ pebbles. In their agent model they show that constant memory is necessary and sufficient. Their algorithm uses $O(\log(\Delta))$ size whiteboards at each node of out degree Δ.

Flocchini, Mans, and Santoro in [54] consider for the first time dynamic graph exploration. They focus on periodically varying (PV) graphs. Theses graphs are modeled by bus system where buses synchronously and periodically move from nodes to nodes. Agents are viewed as riding on the buses. They distinguish whether route lengths are homogeneous or not. In the homogeneous model all routes are of equal length. Then the graphs are distinguished as simple, circular or arbitrary. In simple graphs there are no self-loops. The circular graphs are such that at every route a link appears at most once. For the homogeneous graphs solutions are provided for simple, circular and arbitrary networks with tight complexities $\Theta(kn^2)$, $\Theta(kn)$ and $\Theta(kp)$ and for the heterogeneous case solution algorithms are presented with tight complexities $\Theta(kn^4)$, $\Theta(kn^2)$ and $\Theta(kp^2)$ for simple, circular and arbitrary networks where n is the number nodes, k is the number of routes, and p is the length of a longest route.

2.1.3 Undirected Graph Exploration and Map Construction

Existing work on the undirected graph exploration focuses on specific topology of trees as well as general topologies. First we present the work on trees and rings followed by those focusing on general topologies.

Trees

Averbakh and Berman in [4] look at the p traveling salesmen problem on trees. The objective is to minimize the maximum of distance traveled by salesmen. They show that the problem is NP-hard. They give $\frac{2p}{p+1}$-approximation algorithm when p salesmen begin in different nodes. In [3] traveling salesman problem with two salesmen is investigated. A $\frac{3}{2}$ and $\frac{4}{3}$ approximation algorithms costing $O(n)$ moves for co-located and scattered salesmen are presented.

Fraigniaud et al. in [55] consider the collective exploration of tree networks by

a team of k robots. The robots or agents move synchronously and the goal is to explore as quickly as possible. The problem is NP-hard. They study the impact of communication. In the first model of communication robots can communicate arbitrary information at each time step. For this model a lowerbound of $2 - \frac{1}{k}$ on competitive ratio is presented. In the second model of communication robots can communicate only through whiteboards. The authors of [55] obtain a solution with $O(\frac{k}{\log k})$ competitive ratio for both models. The protocol's time complexity is $O(d + \frac{k}{\log k})$ where d is the diameter of the tree. For the model in which no communication is present a lowerbound of k on competitive ratio is presented.

Diks et al. in [30] focus on memory needed by robots for different types of exploration of trees. They present algorithms for robots with memory $O(\log(\Delta))$ and $O((\log(n))^2)$ memory for perpetual exploration and exploration with return, where Δ is maximum degree and n is the number of nodes. The solution for latter case is improved by Ambühl et al. in [2] who show an algorithm for robots with memory $O(\log(n))$. Some lowerbounds are also established. They are $\Omega(\log(\log(\log(n))))$ and $\Omega(\log(n))$ for exploration with stop and return respectively.

Flocchini et al. in [46] and [47] consider the problem of exploration of rings and trees respectively by asynchronous oblivious robots that can see other. The authors in [47] show trees of size n with maximum degree $\Delta \geq 4$ require $\Omega(n)$ robots to solve exploration. But for the trees of maximum degree 3, authors establish that number agents required is $\Theta(\frac{\log(n)}{\log(\log(n))})$.

General Topologies

Dudek et al. in [42] investigate the problems of robotic exploration and map construction of connected undirected graph. In their model, edges at a node are not labelled. But any robot entering a node through an edge can recognize a cyclic order of edges in relation to edge through which it entered. They show that single robot endowed with k markers can solve the problem of map construction using $O(\frac{mn}{k})$ moves for small k where n is the number of nodes and m is the number of edges.

Panaite and Pelc in [74] consider the problem when robots can recognize previously visited nodes and number of unexplored edges incident on those nodes. They show

that problem is solvable using $m + O(n)$ moves. In [5] piecemeal exploration i.e., exploration of the network with periodically returning to starting point is studied in the same model. They solve the problem using $O(m+n^{1+o(1)})$ where m is the number edges and n is the number nodes.

Dessmark and Pelc [29] investigate the impact of topological knowledge on graph exploration. The cost is measured as competitive ratio against cost of an optimal algorithm. Th authors of [29] show that robot with no topological knowledge can solve with competitive ratio of 2; with an anchored map i.e., starting node identified can solve with competitive ratios $\frac{3}{2}$, $\frac{7}{5}$ and 2 for trees, line graphs and general graphs; with an unanchored map can solve for trees, line graphs and general graphs with competitive ratios less than 2, $\sqrt{3}$ and 2 respectively.

Das et al. in [25] solve the problem of the labelled map construction of an unknown network. They show that the problem is solvable if $\gcd(n, k) = 1$ where n is the number of nodes and k is the number of agents. The authors [25] of present an algorithm costing $O(m \log(k))$ moves when both n and k is known and $O(km)$ solution when only one of k and m is known.

Fraiginaud et al. in [57] look at the effect of maximum degree Δ on the size of memory of automaton needed to solve the exploration problem. They establish tight bound of $\Theta(D \log(\Delta))$. The authors of [43] establish a lowerbound $\Omega(\frac{\log(k)}{\log(\log(k))})$ on the competitive ratio.

2.2 Black Hole Search

In this section we review the existing research on black hole search problem and related exploration and map construction of dangerous graphs. The *black hole search*(BHS) problem has been originally studied for the ring networks in [38] and has been extensively investigated in various settings since then. The main distinctions made are the following:

1. **Synchronous vs asynchronous execution**: In synchronous execution every move of any agent costs single unit of time; Computation and access to tokens or shared memory area take constant units of time. In asynchronous execution every action of any agent takes finite but unbounded time.

2. **Inter-agent communication**: In distributed mobile computing three models of inter-agent communication are used. They are

   (a) Whiteboard: Whiteboard is a shared memory area at the center of nodes that is accessed fairly by the agents i.e., if an agent requests access then it will be granted within finite time. In the case of asynchronous execution access is granted mutually exclusively.

   (b) (Pure) tokens: Pure tokens are pebbles that can placed on and picked up from nodes. Access to pure tokens are granted exactly like the access to whiteboard.

   (c) Enhanced tokens: Enhanced tokens are pebbles that can be placed on and picked up from center or *ports of the networks*. Access to enhanced tokens are granted exactly like the access to whiteboard.

3. **Extent of topological knowledge**: The problem of black hole search has been studied under various degrees of topological knowledge. It has shown in [34, 35] that with out the knowledge of number of nodes then the black hole problem can not be solved in the asynchronous model. The extent of knowledge of topology as follows:

   (a) Topology of the network is known and specific

   (b) Topology of the network is completely known and arbitrary

   (c) Sense of Direction is known

   (d) No topological knowledge except for the number of nodes and number of links

   (e) No topological knowledge except for the number of nodes and an upper-bound on the maximum degree

   (f) (Synchronous only) No topological knowledge at all. Observe that in the asynchronous model number of nodes of network must be known to agents. But in the synchronous model with unitary time delays, time-out can be used to deduce location of black hole and hence knowledge of number nodes is not necessary.

4. **Directed vs Undirected**: In undirected network agents can move along links on either direction. In directed network agents can move along only one direction.

5. **Static vs dynamic networks**: In static networks the set of links does not change. In dynamic networks set of links may change over time.

6. **Co-located or scattered agents:** Agents are said to be co-located if they begin their execution from a single node. Otherwise agents are said to be scattered.

2.2.1 Synchronous Networks

In [23] a variant of synchronous network is considered. In this model, it is assumed that each move takes bounded time. Tight bounds on the number of moves have been established for some classes of trees. It is shown that finding optimal strategy is NP-hard for tree. An $\frac{5}{3}$-approximation algorithm is presented.

Kosowski et al. in [65] study the black hole search in arbitrary synchronous networks. In the case of general networks finding the optimal strategy is shown to be NP-hard for general graph. A $\frac{27}{8}$- approximation algorithm is presented. A lowerbound of $\frac{3}{2}$ is presented for approximation ratio. In [66] the problem when the map of the network is known is studied. A 6-approximation algorithm is presented.

Recent investigations [17, 18] have dealt with scattered agents searching for a black hole in rings and tori. They show that finite automata can solve the problems in optimal time endowed with several mechanisms to communicate such as pebbles, ability mark a link permanently, and detect the presence of other agents. For the model with movable tokens 3 agents with 1 token each necessary and sufficient in oriented or unoriented rings; For the model with unmovable tokens 4 agents with 2 tokens each and 5 agents with 2 tokens each are necessary and sufficient for oriented and unoriented rings. For the oriented tori, [17] presents time optimal algorithms using 3 agents each endowed with two tokens.

Multiple Black Holes

The case of multiple black holes has been investigated in [20] where a lower bound on the cost and a close upper bounds are given. The mobile agents are co-located at the homebase when the execution of the protocol begins. Let b be the number of black holes and D_b the diameter of connected component containing the homebase after removing the black holes. The authors establish a lowerbound of $O(\frac{n}{k} + D_b)$ when $b \leq k - 2$. Two algorithmic solutions are provided when $b \leq \frac{k}{2}$. The first solution has a time complexity of $O(\frac{n \log(n)}{k(\log(\log(n))+bD_b)})$. The second solution has a complexity $O(\frac{n}{k})$ with $k = O(\sqrt(n))$ and $bD_b = O(\sqrt(n))$. In [21], a variant of black hole search is studied. In this model when an agent enters the black hole it both repairs the fault and gets destroyed. An algorithm is given that solves the problem with time complexity $O(\frac{n/k+D\log(f)}{\log(\log(f))})$ where D is the diameter of the network, $f = \min(\frac{n}{k}, \frac{n}{D})$, and $b \leq \frac{k}{2}$.It is shown that this cost is optimal.

Kosowski et al. in [68] and [67] study the black hole search in synchronous directed networks. Let m_b be the number of edges leading into black holes. They show that $O(m_b 2^{m_b})$ agents are sufficient to locate the black hole.

2.2.2 Asynchronous Networks

In this section we describe the state of the research on black hole search in asynchronous networks. We first present the research in whiteboard model followed by token models.

Whiteboard Model

Dobrev et al. in [38] focus on black hole search in anonymous asynchronous rings. The authors focus on number of moves, team size, and ideal time as complexity measures. It is established if agents are co-located two agents are necessary and sufficient. In the case of scattered agents it is shown two and three agents are necessary and sufficient for oriented and unoriented rings. For the time complexity it is demonstrated that regardless of time size $2(n-2)$ time units are required. A lowerbound of $(n-1)\log(n-1) + O(n)$ move complexity is established when agents are initially co-located. An algorithm solving the black hole search with $2n\log(n) + O(n)$ moves using two agents

is presented when agents begin their execution from the same node. An algorithm solving the problem with optimal ideal time using $n-1$ agents is presented. For the case of scattered agents lowerbounds on number of move of $n\log(n-k)$ and $n\log(n)$ when k is known and not known respectively. Asymptotically matching upperbounds are presented.

The subject of the investigation of [31] is the black hole search in common interconnection networks. The authors show that it is possible for two agents to search hypercubes, cube-connected cycles, star graphs, wrapped butterflies, chordal rings, and restricted diameter multidimensional tori and meshes in $\Theta(n)$ moves constructively.

Dobrev, Flocchini, Prencipe, and Santoro in [34] look at black hole search in connected undirected graphs with varying degree of topological knowledge. They look at three situations: topological ignorance, topological ignorance but the network has sense of direction, and complete topological knowledge. For topological ignorance, the authors prove a lower bound of $\Omega(n^2)$ on the number of moves and Δ on the team size where $\Delta+1$ is the upperbound maximum degree known. They present a solution that uses $\Delta+1$ agents and an optimal $O(n^2)$ moves. For topological ignorance with sense of direction, a two-agent algorithm with optimal $O(n^2)$ algorithm is presented. For complete topological knowledge they show that two agents are necessary and sufficient; any solution algorithm must make at least $\Omega(n\log n)$ in the worst-case. They present algorithm achieving the lowerbounds in team size and number of moves. Dobrev, Flocchini, and Santoro in [36] present improved protocols for black hole search in connected undirected graphs when the agents have the complete map of the network. They give a solution that locates the black hole with two agents using just $O(n+D\log D)$ moves where D is the diameter of the network. Dobrev, Flocchini, and Santoro in [37] develop a preprocessing method based on cycles that allow the agents to find the black hole using only $O(n)$ moves for large classes of networks.

In [59] the effects of knowledge of incoming link on the optimal team size has been studied and lower bounds provided. A lowerbound of $\frac{\Delta^2+\Delta}{2}+1$ number of agents as well as an algorithm with matching number of agents is provided for the case in which agents can not know the link through which they arrived at a node.

Black hole search in directed graphs has been investigated in [22], where it is shown that the requirements in number of agents change considerably. The authors of [22] show that there are graphs with maximum degree Δ requiring $\Omega(2^{\Delta})$ agents. They investigate what other restrictions on topological properties will give improved bound. They show that for planar graphs with known planar embedding $2\Delta + 1$ agents are sufficient and 2Δ agents are necessary.

A variant of dangerous node behavior has been studied in [69], where the authors introduce black holes with Byzantine behavior (they do not always destroy a passing agent), consider the periodic ring exploration problem, and show that constant number of agents can solve the problem.

The BHS search in dynamic networks are considered for the first time in [51, 52]. The problem is studied in subway model where carriers move between nodes and agents ride on the carriers. Let n be the number of carriers and p be the length of longest route. Let γ be the number of stops made by carriers at each location. An algorithm solving the problem with $k = \gamma + 1$ agents costing $O(knp + np^2)$ and a matching lowerbound is presented.

Pure Token and Enhanced Token Model

Dobrev et al. in [39] look at black hole search in asynchronous ring networks where the inter-agent communication is by token. The agents are initially co-located. The authors give an optimal $\Theta(n \log n)$ solution for two agents that requires only $O(1)$ tokens.

Dobrev, Santoro, and Shi in [40] look at black hole search by mobile agents scattered in an unoriented asynchronous ring where the agents communicate using a constant number of tokens. The authors show that it is possible to locate the black hole with just 3 agents in $O(n^2)$ moves. They show when number of agents k is known then problem can be solved using $O(kn + n \log n)$ moves. In [41] it is shown that, when agents are scattered on oriented ring, black hole search can be solved using $O(n \log n)$ moves.

Shi in [81] looks at black hole search in some interconnection networks. The

author focus on hypercubes, tori, and complete asynchronous networks. The inter-agent communication model studied is token based model. Both the cases of co-located and scattered agents are studied. For co-located agents, author shows that two agents can find the black hole in all three topologies in an optimal $\Theta(n)$ moves $O(1)$ tokens. For scattered agents, authors present two algorithms. First one is for complete networks and uses n agents each with a single token costing $O(n^2)$ moves. The solution for oriented torus costs $O(k^2n^2)$ moves for $k > 3$ agents.

In [48,49] black hole search using tokens of networks of known topology is studied. The authors of [48, 49] show that tokens are as powerful as whiteboards when the topology is known. Their solution algorithm uses two agents each endowed with a single token can discover black hole with $O(n \log n)$ move complexity. For the algorithm single token capacity at a node is sufficient. A technique called "ping-pong" is introduced by the algorithm.

In [32] black hole search in unknown topology using tokens is considered. The authors of [32] introduce a token model where tokens can be placed on ports of a node or on the center of the node. The links are assumed to be FIFO. They present a solution using optimal number $\Delta+1$ of agents and $O(\Delta^2m^2n^7)$ where n is the number of nodes and m is the number of links.

Multiple Black Holes

Flocchini et al. in [50] consider the map construction problem in a simple, connected dangerous graph by a set of mobile agents that start from scattered locations throughout the graph. The graph contains black links and black holes. They present a protocol that solves the problem. The total cost of the algorithm is $O(n_sm)$ total number of moves, where m is the number of links in the network and n_s is the number of safe nodes. A leader is also elected whenever symmetry can be broken.

Chalopin et al. in [63] investigate the problem of rendezvous when black links are present. Let n be the number of nodes, k be the number of agents and m be the number of links. If there are τ dangerous links in the graph, G and k agents are initially present, then at most $k - \tau$ agents can rendezvous in G. A bound B such that $n \leq B \leq 2n$ on number of nodes is required; extended-view of the network

must be covering minimal. An algorithm is presented with a matching lowerbound, establishing the complexity as $\Theta(m(m+k))$.

2.3 Inter-Agent Communication

Rabin first proposed endowing automata with pebbles in [76]. Since then a large number of algorithms using automata endowed with tokens or pebbles on graphs and networks both safe and dangerous were published in [12, 39, 41, 42, 49, 56]. Enhanced tokens were introduced in [32]. Several problems were studied in whiteboard model. Rendezvous of agents [70, 83], intruder capture [10, 14], network decontamination [44,45], and leader election [11,26,27] are some problems other than black hole search studied under whiteboard model.

2.4 Summary

In this chapter we reviewed the existing works on the black hole search and related problems, and mechanisms for inter-agent communication. In the next chapter we present necessary definitions, basic limitations and techniques for the black hole search in asynchronous networks.

Chapter 3

Definitions, Models, and Limitations

In this chapter the main definitions are introduced, the models are described, limitations of the model and a basic technique are discussed.

3.1 Network Environment, Mobile Agents, and Black Hole

The network environment is a simple undirected edge-labelled graph $G = (V, E)$ of $|V| = n$ nodes and $|E| = m$ edges. The network is *anonymous*; that is, the nodes have no distinct identifiers that can be used in the computation. At each node $x \in V$ there is a distinct label (called port number) associated to each of its incident links (or ports); without loss of generality, we assume that the labels at $x \in V$ are the consecutive integers #1, #2, ..., $\#d(x)$, where $d(x)$ denotes the degree of x; $\Delta(G)$ (or simply Δ) denotes the maximum node degree in G. If $(x, y) \in E$ then x and y are said to be neighbours.

Operating in G is a team $\mathcal{A}$ of anonymous (i.e., identical) agents. The agents obey the identical set of behavioral rules (called *algorithm* or protocol). Initially, they are all in the identical state and enter the network from the identical node h, called *homebase*, but not necessarily at the same time. The agents can move from node to neighboring node in G, perform computation, and interact with the inter-agent communication medium. Links are not necessarily FIFO i.e. for any link e and for any two agents A and B that have entered e from one of the port P in the execution and both have not arrived via other port P' of the link at some time t, first agent to arrive via the P' could be any one of A and B. For the computing capabilities, the agents are regarded as Turing machines. The next section describes the inter-agent communication mechanisms investigated in this thesis. The network contains a *black hole*, a node $\text{BH} \in V$ that destroys any incoming agent without leaving any trace of that destruction. The goal of a *black hole search* algorithm $\mathcal{P}$ is to identify the

location of BH; that is, at least one agent survives and all the surviving agents within finite time must know the edges leading to the black hole. Depending on available topological knowledge a subset L of links that contains as subset links leading to BH is identified. Knowledge of the location of the black hole is *exact* if all and only the links leading to the black hole is determined; it is *partial* if a subset L of links that contains as proper subset all the links leading to the black hole is determined (i.e., some links might be incorrectly suspected to lead to the black hole). Termination of a solution protocol is *explicit* if within finite time all surviving agents enter a terminal state and have the same information on the location of the black hole in the graph.

3.2 Inter-Agent Communication

In this thesis, two models of inter-agent communication are used. They are the *whiteboard* and the *pure token* models. In the existing literature, in addition to these models, the enhanced token model is also used. Agents can see whiteboard or tokens placed on the node they are currently visiting, but not other agents currently there; i.e., agents are *invisible* to each other.

3.2.1 Whiteboard Model

In this model, each node is equipped with a limited amount of storage, called *whiteboard*, which is used by the agents to communicate and coordinate by writing on and reading from. When active at a node, an agent may attempt to access the whiteboard at that node (to read or write); attempts by different agents at the same node are resolved in fair mutual exclusion. In other words, every agent attempting an access, will be granted one within finite time and when an agent A gets access to the whiteboard at a node, no other agent is given access to the whiteboard at that node to read or write until the agent A gives up its access to whiteboard at that node. The accesses are *asynchronous* in the sense that every action they perform (reading and writing on whiteboards) takes an unpredictable (but finite) amount of time; similarly, the interval between activities is finite but unpredictable.

3.2.2 Pure Tokens

For agents operating in pure *token model*: the only mechanism available to the agents for coordination, control and communication is by means of identical *pebbles* that each agent is capable to pick up from a node, hold, carry while moving, and drop in the center of a node. Let t denotes the *token load*, that is the total number of pebbles available in the system. Initially, the tokens can be either held by (some of) the agents or located on the homebase. When active at a node, an agent may attempt to access the pebbles at that node (to determine the number, to drop pebbles, or to pick up pebbles); attempts by different agents at the same node are resolved in fair mutual exclusion. In other words, every agent attempting an access, will be granted one within finite time and when an agent A gets access to the tokens on a node, it gets access to all the tokens on the node simultaneously and no other agent is given access to the tokens on the node to read, pick up, or drop tokens until A gives up its access. The accesses are *asynchronous* in the sense that every action they perform (pebble drop and pick up) takes an unpredictable (but finite) amount of time; similarly, the interval between activities is finite, but unpredictable.

3.2.3 Enhanced Tokens

The *enhanced token model* is similar to the pure token model; the only difference is that each agent is capable to drop and pick up tokens, not only from the center of a node, but also in correspondence to each port. Like for the case of the pure token model, initially, the tokens can be either held by (some of) the agents or located on the homebase. When active at a node, an agent may attempt to access the pebbles at that node (to determine whether the center contains a pebble and which subset of ports contains pebbles, to drop pebbles on locations,or to pick up pebbles from locations); attempts by different agents at the same node are resolved in fair mutual exclusion.

3.2.4 Relationships between Whiteboard, Pure Token, Enhanced Token Models

The whiteboard, the pure token and the enhanced token model are obviously related in terms of computability and complexity, and there are simple mechanisms to transfer solution techniques and complexity results between the models.

It is easy to see that any protocol with a pure token capacity of C (i.e., at most C pebbles can be placed on a node) can be directly implemented using whiteboards of size at most of $\lceil \log(C+1) \rceil$ bits. Hence solutions and complexity results can be immediately transferred from the pure token model to the hybrid token and whiteboard model.

The transfer works also in the other direction: any algorithm to solve a problem $\mathcal{P}$ with whiteboards of size s bits can be easily transformed into an algorithm to solve the problem $\mathcal{P}$ in the token model with a pure token capacity $C = 2^s$ and pure token load $T = n2^s$. Intuitively, the content of a whiteboard will be represented in unary notation by the number of pure tokens placed there; thus, the changing the content of a whiteboard to the sequence of bits $w \leq 2^s$ is implemented by placing w pebbles in the node. Note that any algorithm that uses H token capacity in the hybrid model can be implemented in the pure token model with a pure token capacity of $C = H\Delta$ pebbles. Hence solutions and complexity results can be transferred from the hybrid token and whiteboard models to the pure token model.

Importantly, all these transformations preserve important costs such as time and total number of moves by agents.

3.3 Measures of Efficiency

In this thesis four measures of efficiency are used. They are as follows:

1. **Team size:** Number of agents used by the solution algorithm.

2. **Move complexity:** The total number of moves performed by all the agents in worst case.

3. **Ideal Time:** The asynchrony of the computational entities means that the

algorithm must work regardless of the time required for each computation or movement, which is finite but a priori unknown (i.e., determined by an adversary); however, the time complexity of the algorithm is measured assuming unitary time delays (i.e., determined by a synchronous scheduler), is called *ideal time* and is often employed in distributed computing (e.g., [38, 58, 71]).

4. **Memory:** Size of the whiteboard or number tokens and token capacity of nodes.

3.4 Basic Limitations

Some basic limitations are well known [34, 35]:

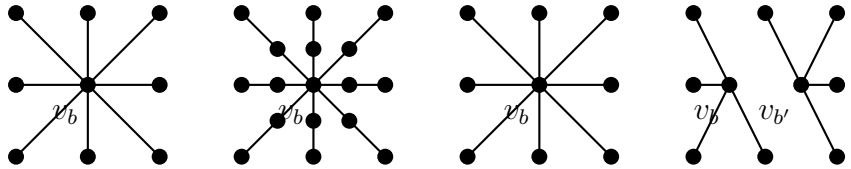
Lemma 3.4.1. *(1) It is impossible to decide whether or not G contains a* Bh*. That is given G which may or may not contain a* Bh*, there is no algorithm that outputs within finite time* 1 *if G contains a* Bh *and* 0 *otherwise (2) If G has a cut vertex different from the homebase, the node h from which all agents begin their execution, the* Bhs *problem is unsolvable. (3) In the absence of other topological knowledge, if n is not known, the* Bhs *problem is unsolvable in asynchronous environments.*

As a consequence, it is assumed that G remains connected once Bh is removed, and that n is known to the agents. Knowledge of n implies that an algorithm could terminate as soon as $n - 1$ safe nodes have been visited. However, due to asynchrony, explicit termination (even with just partial knowledge) may be impossible, as expressed by the following simple observations mostly appeared in existing literature, for example [34, 35]:

Property 3.4.1. *If there are two or more unexplored nodes, then the* Bhs *problem is not solved.*

Lemma 3.4.2. *In absence of any other topological knowledge, explicit termination is not possible*
(1) if only n and an upper bound $\Delta' \geq \Delta$ are known.
(2) if only m is known;

Figure 3.1: Minimum requirement



On the other hand

Lemma 3.4.3. *If n and m are known, explicit termination with exact knowledge is possible.*

In all the algorithms in this thesis, it is assumed that number of links m and number of nodes n are known. The algorithms can be easily modified for the cases in which m is unknown.

3.5 A Basic Technique: Cautious Walk

Following describes *cautious walk* technique introduced in [38]. At any time during the search for the black hole, the ports (corresponding to the incident links) of a node can be classified as follows:

1. *unexplored*: if no agent has moved across this port.
2. *safe*: if an agent arrived via this port.
3. *active*: if an agent departed via this port, but no agent has arrived via it.

Clearly, both unexplored and active links are dangerous in that they might lead to the black hole; however, active links are being explored, so there is no need for another agent to go there unless it is declared safe. Cautious walk is defined by the following two rules:

1. when an agent moves from node u to v via an unexplored port (turning it into active), if it does not disappear (i.e., v is not the black hole) the agent immediately returns to u (making the port safe), and only then resumes its execution;

2. no agent leaves via an active port.

3.6 Summary

In this chapter we discussed the models, limitations and basic techniques. In the next chapter we turn our attention to the black hole search in the ring networks.

Chapter 4

Black Hole Search in the Ring: Improving Time

In this chapter the problem of locating a black hole is considered in a ring of n nodes using a team of *asynchronous* agents communicating by means of whiteboards. The focus of this chapter is on *time complexity*, which is measured in terms of *ideal time* [38, 58, 71], assuming unitary time delays. Several new algorithms are designed improving the existing bounds. A summary of the contribution in relation with the existing results is given below.

It has been shown in [38] that any asynchronous black hole search algorithm for rings requires $2(n-2)$ time in the worst case regardless of the number $k > 1$ of agents. The best algorithm achieves this bound with a team of $k = n-1$ agents with an average time cost $2(n-2)$, equal to the worst case is presented in [38]. Observe here the authors in [38] focus only on worst case complexity. Their technique is that every agent visits all but one node. Hence all but one agent get destroyed and the node excluded by the safely returning agent is the black hole. We focus on worst-case as well as average time. In this chapter, improvements are made on those bounds by showing how the same number of agents can solve the problem requiring only $\frac{7}{4}n + O(1)$ time in average, and 2 extra time units from optimal in the worst case. The average time we consider is ideal time averaged over the position of black hole assuming each node other than home base is equally likely to be the black hole. It is also shown that any asynchronous black hole search algorithm for rings requires $\frac{3}{2}n - O(1)$ time *in average* regardless of the number $k > 1$ of agents, and then prove that, with $2(n-1)$ agents, the optimal average case complexity $\frac{3}{2}n - O(1)$ can be achieved *without increasing the worst case*. Finally, observing that all considered protocols achieve (worst and average) $\Theta(n)$ time using $O(n)$ agents, it is proved that it is possible to locate a black hole in asymptotically optimal (worst and average) $\Theta(n)$ time with just $k = 2$ agents. In fact, we design an algorithm that uses $8n + O(1)$

time in the worst case and $\frac{15}{2}n + O(1)$ on the average, employing an optimal team of 2 agents thus improving the earlier result that employed $n-1$ agents. These results are summarized in Table 4.1. The costs in terms of moves of all these algorithms is $O(n^2)$, the same as that of the algorithm in [38] they improve upon.

Algorithm	Agents	Time Complexity		Move Complexity
		Average	Worst	Worst
[38]	$n-1$	$2(n-2)$	$2(n-2)$ $(\star)$	$O(n^2)$
GROUP	$n-1$	$\frac{7}{4}\ n + O(1)$	$2(n-1)$	$O(n^2)$
OPTAVGTIME	$2(n-1)$	$\frac{3}{2}\ n - O(1)$ $(\star)$	$2(n-2)$ $(\star)$	$O(n^2)$
OPTTEAMSIZE	2 $(\star)$	$\frac{15}{2}n + O(1)$	$8n + O(1)$	$O(n^2)$

$(\star)$ indicates an optimal exact bound.

Table 4.1: Summary of the results of Chapter 4.

The results of this chapter have been published in [7, 9].

4.1 Basic Technique

Since our agents are initially co-located, they can be assigned distinct identifiers. The first agent A_1 to access the whiteboard at the *homebase* reads nothing and hence understands that it is the first agent to access the whiteboard at the *homebase*. Then agent A_1 assumes the identifier 1 and writes on the whiteboard at the *homebase* that identifier 1 is assigned. Similarly the i^{th} agent A_i to access the whiteboard on the *homebase* reads the message that identifier $i-1$ is assigned and identifier i is not assigned. Agent A_i writes on the whiteboard that identifier i is assigned and assumes identifier i. W.l.g, in our all algorithms we assume that set of distinct identifiers can be arbitrary.

4.2 Improving the Average Time Complexity

In this section, improvements are on the average time complexity of [38]. An algorithm that uses only $n-1$ agents, as in [38], but only $\frac{7}{4}n+O(1)$ time on the average (instead of $2(n-2)$), is described. The worst case complexity is $2(n-1)$ (instead of $2(n-2)$).

The idea is to determine the location of the black hole on some node by having a particular pair of agents (witnessing pair) returning successfully to the homebase after exploring a subset of nodes that does not include the black hole. The way subsets of nodes are associated to agents is complicated by the objective of reducing the number of agents entering the black hole.

Recall that node 0 indicates the homebase and the other nodes are indicated as $1, 2, \ldots, n-1$ in clockwise direction. For simplicity, assume that n is of the form: $n = 4q + 1$, for some $q > 1$. The $4q$ agents are divided into four groups: *Left*, *Right*, *Middle* and *TieBreakers*.

The groups *Left* and *Right* contain q agents each. The *Middle* group consists of $q + 1$ agents and the *TieBreakers* group consists of $q - 1$ agents.

The witnessing pairs are chosen in a different way depending on the potential location of the black hole. For a node v with witnessing pairs A_v and B_v are such that v is the unique node that is not visited by both A_v and B_v. Hence, if A_v and B_v returns safely, then the location of BH is v. If the black hole is far from the homebase, it will be witnessed by a pair from *Left* and *Middle* or from *Right* and *Middle*. If instead, the black hole is closer to the homebase the witnessing pair will belong to *TieBreakers* and *Right* or to *TieBreakers* and *Left*.

More precisely, the idea is that the agents of the *Left*, *Right* and *Middle* groups explore each a region of size $3q$ appropriately chosen in such a way that the $2q$ nodes farthest from the homebase are endpoints of complements of explored areas of some agents. In other words, the presence of the black hole in one of those $2q$ nodes would be witnessed by a pair of agents being able to successfully return after their exploration.

The agents of the *Tiebreakers* group are instead used to pair themselves either with an agent from *Right* or with one from $Left$ to locate the black hole when it is within q nodes from the homebase. The details are given in Algorithm 4.1.

Algorithm 4.1 Algorithm GROUP

1. The Left group consists of $left_i$ for $1 \leq i \leq q$. An agent $left_i$ in this group explores all node except the nodes $\{i, i+1, i+2, \cdots, i+q-1\}$. It moves left first, then right and then returns to homebase. (See Figure 4.1).

2. The Right group consists of $right_i$ for $1 \leq i \leq q$. An agent $right_i$ in this group explores all node except the nodes $\{n-i-q+1, \cdots, n-i\}$. It moves right first, then left and then returns to homebase.

3. The Middle group consists of $middle_i$ for $1 \leq i \leq q+1$. An agent $middle_i$ in this group explores all node except the nodes $\{q+i-1, q+i, \cdots, 2q+i-1\}$. It moves left first, then right and then returns to homebase. (See Figure 4.2).

4. The Tiebreaker group consists of $tiebreaker_i$ for $1 \leq i \leq q-1$. An agent $tiebreaker_i$ in this group explores all nodes except nodes $\{i, i-1, i-2, \cdots, 1\}$ at the left of the homebase (or nodes $\{n-1, n-2, \cdots, n-i\}$ at the right of the homebase) starting as soon as either $right_{i+1}$ (or $left_{i+1}$) passes through the homebase. (See Figure 4.3).

5. The black hole is located on node $i+q-1$,$n-i-q-1$,j or $n-j$ iff the witnessing pairs return safely ($left_i$,$middle_i$), ($right_i, middle_{q-i}$), ($tiebreaker_j$,$left_j$), or ($tiebreaker_j$,$right_j$) respectively where $1 \leq i \leq q$ and $1 \leq j < q$.

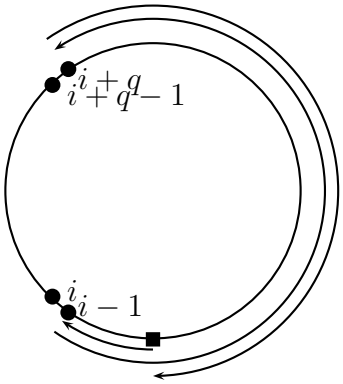


Figure 4.1: Algorithm GROUP: Protocol for Left Group

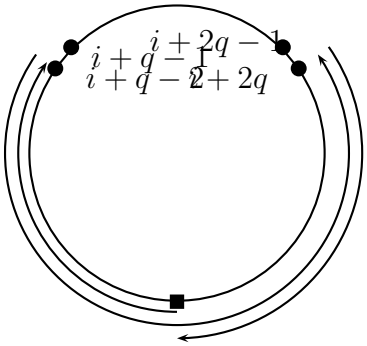


Figure 4.2: Algorithm GROUP: Protocol for Middle Group

For the cases $n = 4q+a$ with $a \neq 1$, the algorithm above can be adapted as follows. The *Left* and *Right* groups are composed of q agents each, while the *Tiebreaker* group is composed of $q-1$ agents and the *Middle* group is composed by $q+a$ agents. As before, agent $middle_i$ in the Middle group explores all nodes except nodes $\{q+i-1, q+i, \cdots, 2q+i-1\}$ for $1 \leq i \leq q+a$.

Following holds:

Theorem 4.2.1. *Algorithm* GROUP *correctly solves the black hole search problem.*

Proof. Assume, for simplicity, that $n = 4q+1$. If the black hole is one of nodes $\{q, q+1, q+2, \cdots, 2q\}$ then, by construction, it is the unique node in the intersection of the excluded segments of a pair of agents $left_i$ and $middle_i$ from the *Left* and the *Middle* groups. (for example, $q+1$ is the only node not explored by the pair $left_2$, $middle_2$). In such a case the black hole is located by the return of such a witnessing pair and the location takes $\frac{3}{2}(n-2)$ time units as both agents $left_i$ and $middle_i$ explore

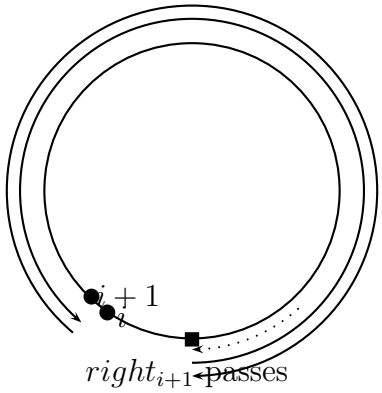


Figure 4.3: Algorithm GROUP: Protocol for Tie-Breaker Group

all but $q = \frac{1}{4}(n-1)$ agents. If the black hole is node i with $i < q$ then agent $right_{i+1}$, after exploring the i nodes on the right of the homebase, passes back through the homebase. At this moment, by the rules of the algorithm, agent $tiebreaker_i$ starts exploring all the nodes (except node i and the $i-1$ nodes between node i and homebase) moving to the right of the homebase. If also agent $left_i$ returns, it means that node i contains the black hole because it is the only unexplored node. Hence, the return of the pair $tiebreaker_i$,$left_i$ (or $tiebreaker_i$,$right_i$) signals the presence of the black hole in node i. The case when $n \neq 4q+1$ follows a very similar reasoning. □

In the following the time complexity of the algorithm is established.

Theorem 4.2.2. *Algorithm* GROUP *locates the black hole with average time* $\frac{7n}{4}+O(1)$ *and worst case time* $2(n-1)$.

Proof. Assume that $n \equiv 1 \pmod 4$ (it is easy to see that this constitutes the worst possible case for time complexity). Agent $tiebreaker_i$ begins the execution after $2i$ time units, and it explores all but i nodes. Hence it returns back to the homebase $2(n-1)$ time units after the start of the execution of the first agent. So, when the black hole is one of the $q-1$ nodes closest to the homebase, it will be located within $2(n-1)$ units. A similar argument apply when the blackhole is symmetrically placed on the other (right) half of the ring, and the worst case result follows.

As for the average time complexity, consider two situations: when the black hole is within q nodes of the homebase, the time for locating it is $2(n-1)$; otherwise, it is $\frac{3}{2}(n-2)$ time units. Hence, the average ideal time complexity of Algorithm GROUP for the case $n \equiv 1 \pmod 4$ is:

$$\frac{2(q-1)(2(n-1))+2(q+1)(\frac{3(n-2)}{2})}{4q} = \frac{7(n-1)}{4} - O(1).$$

It is clear that the average complexity when $n \neq 1 \pmod 4$ cannot be higher than $\frac{7(n-1)}{4} + O(1)$. □

4.3 Optimal Average Time

In this section it will be shown that, by doubling the number of agents, it is possible to achieve simultaneously optimal time both in the average and in the worst case, establishing a lower bound on the average time complexity of black hole search.

The idea of the algorithm is similar to the one of Algorithm GROUP, simplified by the availability of more agents. The idea is to identify pairs of agents ($left_i, right_i$, $i \leq 1 \leq n-1$) among the $2(n-1)$ available, and to assign each pair to "check" a node of the ring. To check node i, an agent of the pair would move to node $i-1$ clockwise (thus exploring nodes $1, 2, \ldots, i-1$) and the other would move to node $i+1$ counterclockwise (thus exploring nodes $i+1, i+2, \ldots n-1$). The presence of the black hole in the ring insures that only one pair will come back to the homebase intact while one agents of each of the other pairs will disappear in the black hole. Once the successful pair returns, the black hole is located.

Algorithm 4.2 Algorithm OPTAVGTIME

$2(n-1)$ co-located agent $left_i, right_i$, $i \leq 1 \leq n-1$ at homebase node 0.

1. Agent $left_i$ explores nodes $(0, 1, 2, \cdots, i-1)$ and returns.
2. Agent $right_i$ explores nodes $(n-1, n-2 \cdots, i+2, i+1)$ and returns.
3. Let ($left_j$, $right_j$) be the only full pair safely returning. The black hole is node j.

Theorem 4.3.1. *Algorithm* OPTAVGTIME *solves the black hole location problem. in average ideal time complexity* $\frac{3}{2}n+O(1)$ *and worst case ideal time complexity* $2(n-2)$. *Both complexities are optimal.*

Proof. Correctness follows from the fact that for each node i there are two agents, namely $left_i$ and $right_i$ such that the singleton set $\{i\}$ is the intersection of the areas that they do not explore.

Now consider worst case time complexity: The time spent by $left_i$ and $right_i$ to reach their destinations and come back is $2Max\{i-1, n-i\}$; the worst case clearly occurs when the black hole is located on node 1 (or $n-2$) and the corresponding time complexity is $2(n-2)$, which is optimal [38].

As for the average time. The presence of the black hole at a node i is witnessed by agents reaching nodes $i-1$ and $n-i-1$. Hence, the ideal time delay for the algorithm when the black hole is located at node i is $2Max\{i-1, n-1-i\}$. $2(i-1)$, which is greater than or equal to $2(n-1-i)$ whenever $i \geq \frac{n}{2}$. Since all nodes other than the homebase are equally likely to contain the black hole, the average time complexity is:

$$\frac{\sum_{i=1}^{\frac{n}{2}-1} 2(n-1-i) + \sum_{i=\frac{n}{2}}^{n-1} 2(i-1)}{n-1}$$

$$= \frac{(n-1)(n-2) + \sum_{i=1}^{\frac{n}{2}-1} -2i + \sum_{\frac{n}{2}}^{n-1} 2i}{n-1}$$

$$= \frac{(n-1)(n-2) - (\frac{n}{2}-1)n + (n-1)n}{n-1} = \frac{3}{2}n + O(1)$$

Notice that nodes on either side of the black hole have necessarily to be reached by some agents and their visit reported back. Hence the time when the black hole is located at node i must be greater than or equal to both $2(i-1)$ and $2(n-1-i)$, which precisely corresponds to the time complexity of the algorithm for node i. Hence it can be concluded that the bound is optimal.

□

4.4 Optimal Team Size

The algorithm of Dobrev et al [38] as well as the improvements presented here have optimal time complexities both in the worst and in the average case; however they all use $O(n)$ agents, which is order of magnitude larger that than the optimal team size $k = 2$. One might think that this large number of agents used by the time-optimal

solutions is necessary. This is however not true, as will be shown in this section.

In the following, an algorithm, that allows $k = 2$ agents to locate the black hole with asymptotically optimal time in both the worst and the average case, is presented. The cost in terms of number of moves of this algorithm is $O(n^2)$, the same as the one of [38] and all the others considered here.

4.4.1 The Algorithm

The idea of the algorithm, called OptTeamSize, is as follows. At each point in time the nodes of the ring are partitioned into an *Explored* area and an *Unexplored* one. The explored area has been already visited by some agent and it is known to be safe, the unexplored area is still to be visited and contains the black hole. Moreover, during the algorithm, the unexplored area is partitioned between the two agents. More precisely, it is always divided into two disjoint areas of different sizes to which agents are assigned: one part containing a single node and the other containing all other unexplored nodes. In each step of the algorithm one of the agents (called *small*) is given the task to explore the area containing a single node, while the other (called *big*) has to explore the other area. The exploration proceeds with cautious walk. Since the two areas are disjoint, one of the agents will certainly succeed in its exploration. If the *big* agent succeeds, the blackhole is obviously located and the algorithm terminates. On the other hand, if the *small* agent returns successfully, it further divides the remaining unexplored area and notifies the *big* agent of the update by leaving a message on the whiteboard of the last node successfully visited by the other agent. The way the update of the unexplored area is performed is such that an agent stays *small* for *two consecutive steps* before switching role. A stage of the algorithm consists of these two consecutive steps and the algorithm is a sequence of stages which terminates when $n - 1$ nodes are known to be safe.

This division process is preceded by a preprocessing phase where the two agents divide the ring in two disjoint parts of almost equal size: only when one of the two returns to the homebase the asymmetric workload division starts to take place.

In the following when it is said that an agent acts as *big* it is meant that it cautiously explores *all but the last* nodes of the unexplored area. When an agent acts

as *small* it cautiously explores the *first* node of the unexplored area. The location of the homebase in the various steps of the algorithms is variable and it is always the central node of the current explored area. An *update* message contains the update information about the current unexplored area and the current location of the homebase. The details of the rules are given in Algorithm 4.3.

4.4.2 Correctness and Complexity

In the following, it will be proved that the algorithm terminates correctly and complexity is established.

Theorem 4.4.1. *Algorithm* OptTeamSize *correctly solves the black hole search using 2 agents and performing* $O(n^2)$ *moves.*

Proof. Observe that since agents explore disjoint sets of nodes during the preprocessing stage at most one agent is destroyed. After the end of the preprocessing phase at least one agent, say *right*, survives and returns. Consider now the execution of the rest of the algorithm. Since the segments of the ring explored by each agent are always disjoint, at least one agent survives every stage. If the *big* agent survives, the algorithm terminates correctly with the black hole being the only node under exploration by agent *small*. If instead the *small* agent survives, the size of the unexplored area decreases by one and the algorithm correctly moves to the next stage (or it terminates if the new size is equal to one). In both cases the size of the unexplored area has decreased and there has been progress towards the solution; hence, one of the agents eventually discovers the location of the black hole.

To prove that the algorithm has $\Theta(n)$ time complexity, First observe that when the exploration phase of the algorithm begins the explored area is at least of size $\frac{n-1}{2}$. Since ideal time is being considered, it can be assumed that the agents start simultaneously, the execution is synchronized, and it takes exactly one time unit for each movement. While the *big* agent, say r, is exploring all but one nodes on its side, the other agent (if it did not disappear in the blackhole before) performs two steps as *small* making at least $\frac{3}{2}(n-1)$ moves (and spending the same amount of time). By that time, under the ideal time assumption, agent r would have either i) returned

Algorithm 4.3 Algorithm OptTeamSize

Two co-located agents l and r . $E = \{v_h\}$. $U = V - E$.

1. Preprocessing Phase: Agent l (resp. r) explores cautiously the leftmost (resp. rightmost) $\lfloor |U|/2 \rfloor$ nodes of the unexplored area and when finished returns to the homebase v_h.

2. Exploration Phase: One of the agents (say l) arrives at the homebase and becomes *small*. Agent l moves to the last explored node on agent r's side, it leaves an update message to r indicating to act as *big*. Agent l then moves to its side and act as *small*. Stage 1 of Phase 2 begins.

3. Stage i of Phase 2:

   (a) If the *big* agent (say r) returns to the homebase (or the *small* agent returns and the size of the unexplored area is one) then the blackhole is located and the algorithm terminates.

   (b) Otherwise, the *small* agent l returns, it moves to the last explored node on agent r's side, it leaves an update message for r indicating to maintain the same role *big*.

   (c) Agent l moves back to its side and it acts as *small*.

   (d) If r returns, then the blackhole is located and the algorithm terminates.

   (e) Otherwise agent l returns, it moves to the last explored node on agent r's side, it leaves an update message for r instructing to reverse role. Agent l then moves to the other side and it changes role acting as *big*.

   (f) If l returns the blackhole is located and the algorithm terminates.

   (g) Otherwise Agent r returns and becomes *small*; it moves to agent l's side, it leaves an update message for agent l instructing it to act as *big*. Agent r then moves back to its side and acts as *small*.

   (h) If l returns then the blackhole is located and the algorithm terminates.

   (i) Otherwise agent r returns, it changes role becoming *big*, it moves to agent l's side and it leaves a message to agent l at the last explored node updating the unexplored area and instructing to reverse roles. Agent l moves to its side and acts as *big*.

   (j) Stage $i + 1$ starts.

safely determining the location of the black hole, or *ii*) died in the black hole. In the first case, obviously the time complexity is $O(n)$. In the other case, when agent l switches role becoming *big* and moves to explore all but one nodes, it necessarily completes its task locating the black hole, again with an overall time complexity of $O(n)$.

To show that the worst case move complexity is $O(n^2)$, it suffices to notice that in the worst possible asynchronous execution it is always the *small* agent that completes a step, while the big agent is slow on a link. Since the *small* agent manages to explore a single node in each step, and the size of the unexplored area when this procedure starts is $\frac{n}{2}$, $O(n)$ steps are necessary to locate the black hole. In each step however $O(n)$ moves are performed by the *small* agent to explore and report the update on the other side of the explored area, for a total of $O(n^2)$ moves. □

Following discussions derive the *exact* average and worst case time complexities of the algorithm.

Theorem 4.4.2. *Algorithm* OptTeamSize *solves the black hole search in average ideal time* $\frac{15}{2}n + O(1)$ *and worst case* $8n + O(1)$.

Proof. By symmetry of the algorithm it may be assumed that the black hole is located on the right half of the ring (w.l.g let n be even). Then it suffice to calculate the ideal time delay when the black hole is located at node $i \leq \frac{n}{2}$ (i.e., $\frac{n}{2}$ nodes to the right of the homebase). Consider following different cases.

- *Case 1*: Node i is the border node of the partition between the right and the left agents. In this case the left agent returns after $\frac{3}{2}n + \frac{n}{2}$ time units to the homebase. In the sum, the first addend is for the cautious exploration and the second is for the time taken to return to the homebase. Now the left agents follows the path of the right agent. The right agent must have died at the last node of its partition. So in another n time units the left agent will reach the last safe node explored by the right agent and return. So, in this case the black hole is located in $3n$ total time units.

- *Case 2*: Node i is the neighbor of the border node of the partition between the right and the left agents. Similarly to the previous case, the left agent will take

$3n - O(1)$ time units to return to the homebase after exploring all nodes, except the black hole and its neighbor. Now the left agent in the role of *small* explores the last safe node and return in further $n + O(1)$ time units. In this case the black hole is then located in $4n + O(1)$ time units.

- *Case 3*: Node i is the third node from the border of the partition between the right and the left agents. In this case the left agent discovers the black hole after the end of the second round as *small*. The total ideal time delay in this case is $5n + O(1)$.

- *Case 4*: Node i is the fourth node or the node further from the border of the partition between the right and the left agents. In this case the left agent (l) performs two rounds as *small* and a round as *big*. Observe that under ideal conditions the right agent would die in the black hole and would not return, so it suffices to count the time taken by the left agent. Agent l explores $(n-i)+O(1)$ nodes in total and this cautious exploration costs in total $3(n-i)+O(1)$ time units. Next, the time, necessary for the other movements of the left agent, is computed. Agent l takes $\frac{n}{2}+i+O(1)$ time units for reaching the last safe node explored by the right agent. Moreover, agent l takes $4(\frac{n}{2}+i)+O(1)$ time unit for the two rounds as *small*; after becoming *big*, agent l reaches the last explored node on its side in $\frac{n}{2}+i$ time units. At this point it cautiously explores (the cost of the exploration has been already accounted for earlier). Finally, the agent returns to the homebase in $n-i+O(1)$ time units. Thus the total ideal time delay is $7n+2i+O(1)$ time units.

Hence the average ideal time delay is :

$$\frac{12n + \sum_{i=1}^{i=\frac{n}{2}}(7n+2i)}{\frac{n}{2}} = 7n + \frac{(\frac{n}{2})(\frac{n}{2}+1)}{\frac{n}{2}} = \frac{15}{2}n + O(1)$$

The worst case occurs in correspondence of Case 4, when $i = n/2 - O(1)$, which yields $8n + O(1)$.

□

4.5 Summary

In this chapter we have considered the black hole search problem in the ring focusing our attention on time complexity. We have described three different algorithms each one with different desirable features. Algorithm Group improves upon the existing solution of [38] in average time by slightly increasing the worst case time complexity and maintaining the same number of agents; algorithm OptAvgTime achieves both optimal average and worst time at the expense of doubling the number of agents; OptTeamSize allows to solve the problem with the optimal number of agents, with slight increase in the exact time complexities, which however remains linear. In the next chapter, we focus on move complexity for the black hole search problem on rings.

Chapter 5

Black Hole Search in the Ring: Improving The Number of Moves

This chapter considers the Black Hole Search in rings from the perspective of *move complexity*. The existing result is improved by showing a lower bound and a matching upper bound which are exact up to the constant of proportionality. The algorithm is then slightly modified and further improved to obtain the same bound on the number of moves and in optimal time (see Table 5.1 for a Summary of the results of this chapter).

The main results of this chapter have been published in [8].

Algorithm	Agents	Time Complexity		Move Complexity
		Average	Worst	Worst
[38]	2 $(\star)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$2n \log_2(n-1) + O(n)$
TRISECT	2 $(\star)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$3n\frac{\log_2(n)}{\log_2(3)} + O(n)$ $(\star)$
OPTIMAL	2 $(\star)$	$O(n)$	$O(n)$	$3n\frac{\log_2(n)}{\log_2(3)} + O(n)$ $(\star)$

$(\star)$ indicates an optimal exact bound.

Table 5.1: Summary of the results of Chapter 5.

5.1 Improved Lower Bound

In this section, the lower bound of [38] on the number of moves is improved. The new lower bound is established by describing a game between an adversary and any algorithm solving the black hole search, which will force an execution of the algorithm

in which the agents will perform at least the number of moves expressed by the bound. The new lower bound is established for algorithms using two agents. The result can be easily generalized to the case of more agents.

The adversary models the asynchronous scheduler; decides the location of black hole since a priori any node other than homebase can possibly be black hole. The adversary has the power to:

1. *block* a port for a finite time: the corresponding link becomes very slow, effectively blocking all the agents traversing it.

2. *unblock* a blocked port: the agents in transit on the corresponding link will then arrive to their destination.

3. choose which agents will move, if there is a choice;

4. decide where the black hole is located in the unexplored area.

Any agent exiting through or in transit on a blocked port will be said to be *blocked*. The adversary can block any port at any time; however, within finite time, it must unblock any blocked port not leading to the black hole.

Let E^t and U^t be the the sets of explored and unexplored nodes at time t respectively; for simplicity, when no ambiguity arises, the superscript t will be omitted. Since the topology is ring and agents are initially co-located E^t and U^t are both connected components of the ring. First notice that no algorithm can require both agents to go from E^t to U^t in the same direction or equivalently same link; otherwise, the adversary would have both agents enter the black hole. Thus, if both agents leave E^t, must do so in opposite directions.

The following property is from [38]:

Lemma 5.1.1 ([38])**.** *Let $|U^t| > 2$; then for any solution algorithm $\mathcal{P}$, within finite time the two agents will have left the explored area E^t in different directions.*

Consider the time $t' \geq t$ when the first of the two agents leave E^t; consider now blocking that agent. By Lemma 5.1.1, there is a time $t'' \geq t'$ when the other agent leaves E^t in the opposite direction; consider now blocking also this agent. This

situation, in which both agents are blocked at the brink of exploration, is called a *frozen* configuration and the time when this happens the *freezing* time; clearly within finite time at least one of the agents must become unblocked.

Regardless of time, denote by r and call *right agent* the one exploring U from the right (i.e., in the counter-clockwise) direction, and denote by l and call *left agent* the one exploring U from the left (i.e., in the clockwise) direction. Should the two agents switch role during an execution, their names are also switched accordingly in the moment they cross each other. Thus, in the following assume, without loss of generality, that the right agent r consistently explores the unexplored area from the right, while the left agent l explores from the left.

The following definition is from [38].

Definition 5.1.1. *A path* $< u = u_1, u_2, ..., u_d = v >$ *from node* u *to node* v *is a* causal chain *at time* t*, if there is an agent* a *and times* $t < t_1 < t'_1 < t_2 < t'_2 ... t_d < t'_d$ *such that* a *leaves* u_j *at time* t_j *and reaches* u_{j+1} *at time* t'_j *(*$0 < j < d$*); the agent* a *is said to* execute *the causal chain.*

In the following we define what it means for two causal chains executed by the same agent to *coincide at an edge* and to be *distinct*.

Definition 5.1.2. *Let an agent execute two causal chains* c_1 *at time* t *and* c_2 *at time* s*, composed, respectively, by nodes* $u_1, u_2, ..., u_d$ *at times* $t < t_1 < t'_1 < t_2 < t'_2 ... t_d < t'_d$*, and by nodes* $w_1, w_2, ..., w'_d$ *at times* $s < s_1 < s'_1 < s_2 < s'_2 ... s_{d'} < s'_{d'}$*. Let that* c_1 *and* c_2 coincide at edge (v, v') *if and only if there exists* $u_i = v, u_{i+1} = v'$ *and* $w_j = v, w_{j+1} = v'$ *such that* $t_i = s_j$ *and* $t'_i = s'_j$*. Two causal chains executed by an agent that do not coincide on any edges are called* distinct.

If two causal chains c_1 and c_2 by agent a coincide at an edge, they coincide in every edge in their intersection; their union is a causal chain executed by a, of which c_1 and c_2 are both fragments.

Next, the important notion of *overlapping* chains executed by different agents is defined.

Definition 5.1.3. *Two causal chains executed by different agents* overlap *if they have at least a node in common.*

Notice that, in this definition, the two agents visit the common nodes arriving from opposite directions. Further notice that there is no temporal condition on the two chains, which could occur in different executions (as will become clear later).

The importance of overlapping chains derives from their link to communication between agents. To make this link more precise, note that information from an agent to the other can be communicated directly (i.e. by leaving on a whiteboard a message that the other agent reads) or indirectly (by absence of a message on a whiteboard). Clearly, in an execution, the absence of a message at a node v can only be meaningful for an agent if there is another execution during which a message is written at v by the other agent. In other words, indirect communication is possible in an execution only if there is direct communication in another execution. Regardless of whether direct or indirect, for communication to take place, two overlapping causal chains must be executed, possibly in different executions. If the two overlapping causal chains take place in the same execution, then the communication is *explicit*; otherwise, communication is *implicit*. Indeed overlapping causal chains are a crucial element in the proof of all lower bounds for black hole search in rings.

Note that, in particular, a causal chain to the center of the explored area must be executed for communication to take place, as identified in [38]. The requirements on the moves that are necessary for the communication to take place in order to explore the ring constitute the basis for the calculation of the lower bound for the black hole search problem.

Let $\mathcal{P}$ be an arbitrary solution protocol, $|U^T| \geq 2$, and let both agents be in a frozen configuration at time T. Let $\mathcal{E}_l$ be the execution of $\mathcal{P}$ starting at time T, where agent r is kept blocked and l is unblocked and let to explore; let $\mathcal{E}_r$ be analogously defined.

The following lemma is a simple refinement and extension of Lemma 3 of [38].

Lemma 5.1.2. *For every node z in U^T, there is a node $w(z)$ in E^T such that both conditions hold:*

(*a*) *In execution $\mathcal{E}_r$, at some time $t_r > T$ before exploring z, agent r executes a causal chain to $w(z)$ from some newly explored node.*

(*b*) *In execution $\mathcal{E}_l$, at some time $t_l > T$ before exploring z, agent l executes a causal chain to $w(z)$ from some newly explored node.*

Proof. For any node $z \in U^T$, let $t_r(z) > T$ be the time when r moves to explore $z \in U$ in execution $\mathcal{E}_r$, and let $v_r(z)$ be the leftmost node visited by r between times T and $t_r(z)$ in $\mathcal{E}_r$; let $t_l(z) > T$ and $v_l(z)$ be analogously defined.

Assume by contradiction that for some node $z \in U^T$ there is no node $w(z) \in E^T$ satisfying conditions (a) and (b) of the Lemma. This means that there is no overlap between the causal chains of r in $\mathcal{E}_r$ and those of l in $\mathcal{E}_l$. This implies that, in the execution $\mathcal{E}$ starting at time T, in which neither agents are blocked and z is the black hole, each agent behaves in $\mathcal{E}$ as it did in the execution where the other agent was blocked; thus both agents enter z, the black hole, contradicting the assumed correctness of $\mathcal{P}$ (see Figure 5.1).

□

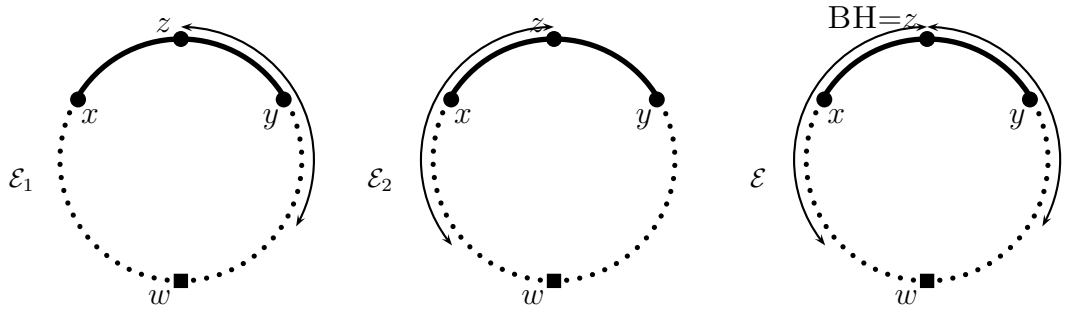


Figure 5.1: Both agents are unblocked in $\mathcal{E}$. The execution of each agent is independent of the other. The adversary sets node z as the black hole.

In other words, for any $z \in U^T$, after time T there is a causal chain $c_r(z)$ by r in $\mathcal{E}_r$ and a causal chain $c_l(z)$ by l in $\mathcal{E}_l$ overlapping in $w(z)$, and whose union covers the entire E^T.

A more useful formulation of this property is given by the following

Corollary 5.1.1. *Let $|U^T| \geq 2$, and let B_r and B_l be a partition of U^T into two non-empty contiguous segments, B_r clockwise closer to E^T. Then there exist nodes $w \in E^T$, $z_r \in B_r$, and $z_l \in B_l$ such that:*

(i) at some time $t_r > T$, r executes a causal chain c_r in $\mathcal{E}_r$ from z_r to w before exploring B_l, and

(ii) at some time $t_l > T$, l executes a causal chain c_l in $\mathcal{E}_l$ from z_l to w before exploring B_r.

Proof. By Lemma 5.1.2, choosing z to be the leftmost node of B_r (or the rightmost node of B_l). □

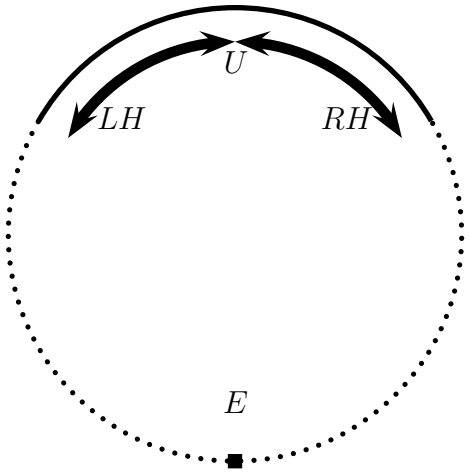


Figure 5.2: Partitions of the ring. The black square indicates the homebase, the bold line is the unexplored area U, the dotted line is the explored area E.

Any algorithm $\mathcal{P}$ solving the black hole search problem must obviously be able to solve the problem starting from any partition of the ring into two contiguous segments, E (safe area, including and possibly consisting just of the homebase) and U (the unexplored area containing the black hole), regardless of the size and choice of U.

Now, we can state the main theorem.

Theorem 5.1.1. *The cost $M_{\mathcal{P}}(s,n)$ of locating a black hole with algorithm $\mathcal{P}$ when $|U| = s$ and $|E| = n - s$, is $M_{\mathcal{P}}(s,n) \geq f(s,n) = 3n\frac{\log_2(s)}{\log_2(3)} - 6s - 6n$.*

Proof. The proof is by induction on the size s of the unexplored area. Let $P(s)$ be the predicate

$$P(s) \equiv \text{“}M_{\mathcal{P}}(s,n) \geq f(s,n)''$$

The basis of the induction is provided by the fact that $P(s)$ is true for $1 \leq s \leq 7$. In fact, for $1 \leq s \leq 7$, $3n\frac{\log_2(j)}{\log_2(3)} < 6n$; hence, $f(j,n) < 0$ and the theorem trivially holds.

Let $s \geq 8$, and let $P(j)$ hold for $1 \leq j < s$. Now we show that $P(s)$ holds. First of all, the adversary brings the agents to a frozen configuration; by definition $|U^T| = s$ and $|E^T| = n - s$.

Consider the partition of U^T into LH and RH, the left and right half of the unexplored area. By Corollary 5.1.1, in execution $\mathcal{E}_r$, before beginning to explore LH, r executes a causal chains c_1 at some time t_r^1; in $\mathcal{E}_l$, before beginning to explore RH, l executes a causal chains d_1 at some time t_l^1; the two chains overlap in at least one node w.

We now distinguish three scenarios (See Figure 5.3). They are:

1. **S1**: There is a node u in the explored area to which agent r in $\mathcal{E}_r$ executes a causal chains c_0 distinct from c_1 before beginning to explore LH, and to which agent l in $\mathcal{E}_l$ executes a causal chains d_0 distinct from d_1 before beginning to explore RH.

2. **S2**: The scenario S1 does not occur and one of the following situations occurs.

   – $S2(r)$: In $\mathcal{E}_r$, agent r executes a causal chain c_0 distinct from c_1 to node w, before beginning to explore LH; w.l.g, let c_0 is executed by the agent r, before executing c_1.

   – $S2(l)$: In $\mathcal{E}_l$, agent l executes a causal chain d_0 distinct from d_1 to node w, before beginning to explore RH; w.l.g, let d_0 is executed by the agent l, before executing d_1.

3. **S3**: Scenarios S1 and S2 do not occur.

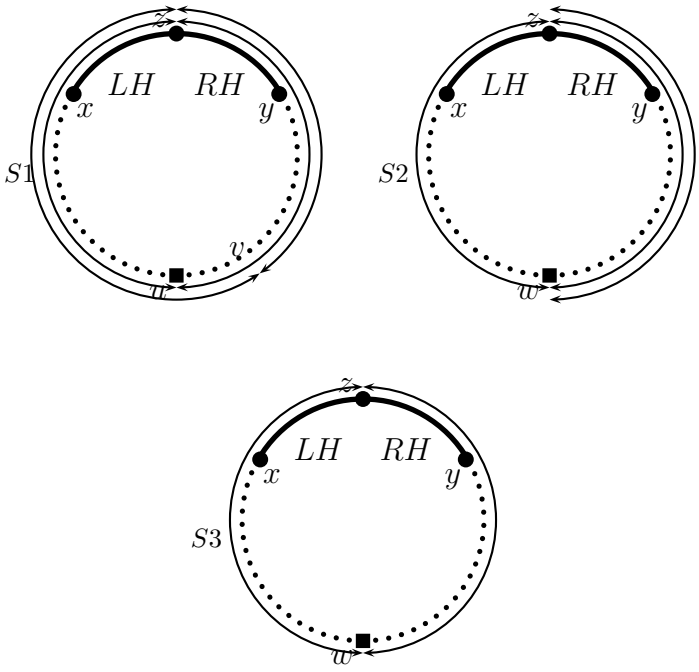


Figure 5.3: Different scenarios

Now, consider the three scenarios and show that all the three scenarios adversary can force the $f(s, n)$ moves. First, following the definition is made.

Definition 5.1.4. *For a causal chain c, we denote by $|c|$ the number of moves made in the causal chain c.*

Consider first scenario $S1$. By assumption, in execution $\mathcal{E}_r$, agent l is kept blocked and agent r executes the causal chains c_0 and c_1. Let the newly explored area be $A_r \subseteq RH$ of size a_r. In the execution $\mathcal{E}_l$, agent r is kept blocked and agent l executes the causal chains d_0 and d_1. Let the newly explored area be $A_l \subseteq LH$ of size a_l.

In execution $\mathcal{E}_r$, by inductive hypothesis, the adversary can force at least $X = 2|c_0|+2|c_1|+f(s-a_r, n)$ moves. In execution $\mathcal{E}_l$, by inductive hypothesis, the adversary can force at least $Y = 2|d_0| + 2|d_1| + f(s - a_l, n)$ moves.

The adversary, by choosing the costliest of the two executions, can force a number

of moves that is at least $\frac{Y+X}{2}$. Hence

$$M_{\mathcal{P}}(s,n)$$

$$\geq \frac{1}{2}(2|c_0| + 2|c_1| + f(s-a_r,n) + 2|d_0| + 2|d_1| + f(s-a_l,n))$$

$$\geq \frac{1}{2}((2|c_0| + 2|d_1|) + (2|d_0| + 2|c_1|)$$

$$+(3n\log_3(s-a_l) - 6(s-a_l) - 6n) + (3n\log_3(s-a_l) - 6(s-a_l) - 6n))$$

$$\geq \frac{1}{2}(4(n-s) + 3n\log_3(s^2/4) - 6s - 12n)$$

$$\geq 3n\frac{\log_2(s)}{\log_2(3)} - 6s - 6n$$

In other words, in scenario **S1** the theorem holds.

Let us now consider first scenarios **S2** and **S3**. Without loss of generality, for scenario **S2** we consider scenario $S2(r)$. We will be showing that, in both cases, the adversary can force three distinct pairs of overlapping causal chains.

Among all the causal chains executed in $\mathcal{E}_l$ by agent l before exploring RH, let h_1 be the one that reaches deepest inside the explored area, and let p be the node of this chain closest to the right border. Let $A_l \subseteq LH$ be the newly explored area in this chain (i.e., the portion of the chain in U^T).

Among all the causal chains executed in $\mathcal{E}_r$ by agent r before exploring LH, let g_1 be the *earliest* causal chain executed by agent r to p. Let $A_r \subseteq RH$ be the new area explored in this chain. Note that g_1 and c_1 are necessarily distinct because, the causal chain c_0 overlaps with at least one chain executed by l before beginning to explore RH and h_1 is the chain executed by l before beginning to explore RH that reaches deepest; hence, the causal chain c_0 overlaps with the causal chain h_1; but by assumption the causal chain c_0 is executed before the causal chain c_1.

Let us define, in execution $\mathcal{E}_r$, the following two other causal chains (which will be shown to exist), executed by r to some nodes in the explored area, before it begins to explore A_l: chain g_0, executed at time t_r^0 before completing the exploration of A_r; and chain g_2, executed at time t_r^2 before beginning the exploration of A_l. In the case there are several choices for the causal chains g_2 or g_0, the ones that were executed to

the closest to the left border are chosen. The times of executions of the causal chains of agent r are related by the relation $t_r^0 < t_r^1 \leq t_r^2$. Symmetrically, define the causal chains h_0 and h_2, and the times t_l^0 and t_l^2; the times of executions of causal chains of agent l are related by the relation $t_l^0 < t_l^1 \leq t_l^2$.

By definition we have:

Claim 5.1.1. *Chains g_0 and h_1 do not overlap.*

Claim 5.1.2. *Chains h_0 and g_1 do not overlap.*

Proof. Consider first Scenarios $S2(r)$. By contradiction, let the causal chains h_0 and g_1 overlap. By definition, h_1 is executed at least up to p. Again by definition, h_1 is the causal chain reaching closest to right border of the explored area. Moreover, by definition of c_1, c_1 overlaps with at least one causal chain executed by l before it begins to explore LH (d_1 is one of them). Hence, c_1 and h_1 overlap. As a consequence, (h_0, g_1) and (h_1, c_1) are distinct overlapping pairs This would however imply scenario $S1$; a contradiction.

Consider now scenario $S3$. By contradiction, let the causal chains h_0 and g_1 overlap; then h_0 and g_1 are executed up to some node q. By definition, h_1 is executed at least up to q. Hence, the two distinct causal chains h_0 and h_1 are executed by agent l to some node v and a causal chain g_1 is executed by agent r to q . This would however imply scenario $S2$; a contradiction. □

Claim 5.1.3. *Chains g_2 and h_0 overlap.*

Proof. Observe that h_0 does not overlap with g_1 (by Claim 5.1.2), nor with g_0 (which is by definition shorter than g_1). If, by contradiction, it does not overlap with g_2 as well, then the adversary can force the agents to explore a common node, the right most node of A_l, where it would place the BH. Consider in fact the execution $\mathcal{E}'$ coinciding with $\mathcal{E}_r$ until r executes g_0, g_1 and g_2; at that time, the adversary blocks r and unblocks l until it executes h_0. Since h_0 does not overlap with g_0, g_1 and g_2, l cannot distinguish between the current execution and $\mathcal{E}_l$; thus it returns to explore A_l and dies in the BH. At this time the adversary unblocks r; since, by definition, no chain performed by r before it explores A_l goes further than g_2, and g_2 does not

overlaps with h_0, r cannot distinguish between the current execution and $\mathcal{E}_r$; thus, it which proceeds to explore A_l and dies in the BH.

□

Since g_2 and h_0 overlap but g_1 and h_0 do not overlap, the causal chains g_2 and g_1 are distinct. Following a similar reasoning, the pair (g_0, h_2) overlap, and h_2 is distinct from h_1.

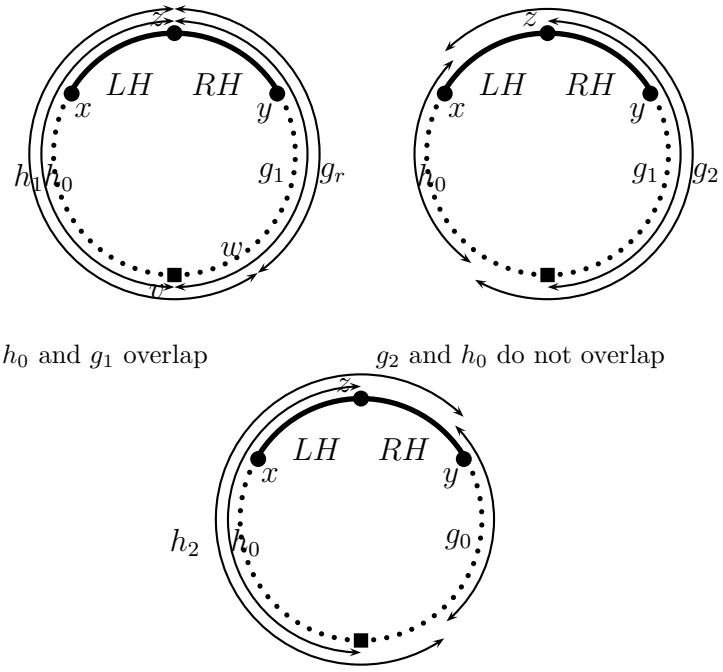
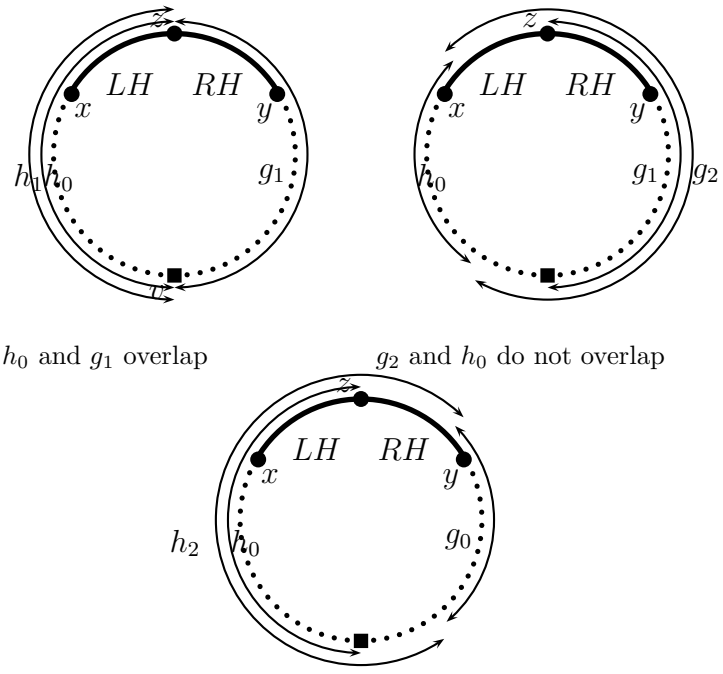


Figure 5.4: Scenario S2

Summarizing, in scenarios $S2$ and $S3$, agent r executes three causal chains g_0, g_1 and g_2 at times, $t_r^0 < t_r^1 < t_r^2$; agent l executes three causal chains h_0, h_1 and h_2 at times $t_l^0 < t_l^1 < t_l^2$. Moreover, the following pairs are overlapping chains: (g_0, h_2), (g_1, h_1) and (g_2, h_0). We are now going to show that $P(s)$ holds also for these two scenarios.

Let $|A_r| = a_r$ and $|A_l| = a_l$. For the rest of the proof, the main argument consists of several claims.

Claim 5.1.4. *If $a_r a_l \geq \frac{s^2}{9}$, then $P(s)$ holds.*



h_0 and g_1 overlap

g_2 and h_0 do not overlap

h_2 and g_0 do not overlap

Figure 5.5: Scenario S3

Proof. The adversary proceeds by choosing the costliest of $\mathcal{E}_r$ up to (and including) the execution of g_2, and of $\mathcal{E}_l$ up to (and including) the execution of h_2.

In $\mathcal{E}_r$, r executes the causal chains g_0, g_1 and g_2, while in $\mathcal{E}_l$, l executes the causal chains h_0, h_1, and h_2. The still unexplored area at t those times are of sizes at least a_l and a_r respectively. In both executions, the following pairs are overlapping: (g_0, h_2), (g_1, h_1) and (g_2, h_0). They altogether constitute $6(n-s)$ moves with the return. In the first execution, by inductive hypothesis, the adversary can force at least $2|g_0| + 2|g_1| + 2|g_2| + f(a_l, n)$ moves. In the second execution, by inductive hypothesis, the adversary can force at least $2|h_0| + 2|h_1| + 2|h_2| + f(a_r, n)$ moves.

The adversary can thus force a number of moves that is at least the average of the two executions. Hence

$$M_{\mathcal{P}}(s,n)$$

$$\geq \frac{1}{2}(2|g_0| + 2|g_1| + 2|g_2| + f(a_l,n) + 2|h_0| + 2|h_1| + 2|h_2 + f(a_r,n))$$

$$= \frac{1}{2}((2|g_0| + 2|h_2|) + (2|g_1| + 2|h_1|) + (2|g_2| + 2|h_0|)$$

$$+(3n \log_3 a_l - 6a_l - 6n) + (3n \log_3 a_l - 6a_l - 6n))$$

$$= \frac{1}{2}(6(n-s) + 3n \log_3(a_r a_l) - 6s + 12n) \geq 3n + \frac{3}{2} n \log_3(a_r a_l) - 6s - 6n$$

$$= 3n \frac{\log_2(s)}{\log_2(3)} - 6s - 6n$$

if $a_r a_l \geq \frac{s^2}{9}$.

□

Claim 5.1.5. *If $a_r + a_l \leq \frac{s}{2}$, $P(s)$ holds.*

Proof. Consider the execution in which the adversary unblocks both r and l but controls the timing in such a way that they meet when r is executing causal chain g_1, and l is executing causal chain h_1. Note that they do not meet before because g_0 does not overlap with h_0 and h_1, and analogously h_0 does not overlap with g_0 and g_1. After this happens, both agents must leave the explored area within finite time from different directions.

The unexplored area at this point is at least $s - a_l - a_r$. Hence, the cost is at least: $2|g_1| + 2|h_1| + f(s - a_r - a_l, n)$. Thus,

$$M_{\mathcal{P}}(s, n) \geq 2|g_1| + 2|h_1| + f(s - a_r - a_l, n)$$

$$= (2|g_1| + 2|h_1|) + 3n \log_3(s - a_r - a_l) - 6(s - a_r - a_l) - 6n$$

$$\geq 2(n - s) + 3n \log_3(s/2) - 3s - 6n \geq 3n \log_3 s - 5s - 6n$$

if $a_r + a_l \leq \frac{s}{2}$.

□

Claim 5.1.6. *If $(s - a_r)(s - a_l) \geq \frac{s^2}{2}$, then $P(s)$ holds.*

Proof. The adversary proceeds by choosing the costliest of the following two executions.

In $\mathcal{E}_r$, r executes the causal chain g_1, while in $\mathcal{E}_l$, l executes the causal chain h_1. The still unexplored area at those times is of size at least $s-a_l$ and $s-a_r$, respectively. In both executions the pair (g_1, h_1) is by definition overlapping.

These executions, altogether constitute $2(n-s)$ moves with the return. In the first execution, by inductive hypothesis, the adversary can force at least $2|g_1|+f(s-a_r,n)$ moves. In the second execution, by inductive hypothesis, the adversary can force at least $2|h_1| + f(s-a_l,n)$ moves.

The adversary, by choosing the costliest of the two executions, can force a number of moves that is at least the average of the two executions. Hence,

$$M_{\mathcal{P}}(s,n)$$

$$\geq \frac{1}{2}(2|g_1| + f(s-a_r,n) + 2|h_1| + f(s-a_l,n))$$

$$= \frac{1}{2}((2|g_1| + 2|h_1|)+$$

$$+(3n\log_3(s-a_r) - 6(s-a_r) - 6n) + (3n\log_3(s-a_l) - 6(s-a_l) - 6n))$$

$$= \frac{1}{2}(2(n-s) + 3n\log_3((s-a_r)(s-a_l)) - 6s + 12n)$$

$$\geq 3n\frac{\log_2(s)}{\log_2 3} - 6s - 6n$$

if $(s-a_r)(s-a_l) \geq \frac{s^2}{2}$. □

Summarizing, by Claims 5.1.4-5.1.6, it suffice to consider only the cases in which the following three conditions simultaneously hold:

1. $Cond1$: $a_r a_l < \frac{s^2}{9}$ (by Claim 5.1.4).

2. $Cond2$: $a_r + a_l > \frac{s}{2}$ (by Claim 5.1.5).

3. $Cond3$:$(s-a_r)(s-a_l) < \frac{s^2}{2}$ (by Claim 5.1.6).

Claim 5.1.7. *Conditions $Cond1$, $Cond2$ and $Cond3$ cannot simultaneously hold.*

Proof. Without loss of generality, let $a_l \leq a_r$. By contradiction, let us assume that the three conditions simultaneously hold.

By condition $Cond1$

$$a_l \leq \frac{s}{3} \leq \frac{\frac{s}{3} - a_r}{s - 3a_r} s$$

This implies the following upper bound on $a_r + a_l$:

$$(a_r + a_l) \leq \frac{s}{3} + 3\frac{a_r a_l}{s} \tag{5.1}$$

By condition $Cond2$:

$$a_r \geq \frac{s}{4} \geq \frac{\frac{s}{4} - a_l}{s - 4a_l} s$$

This implies the following lowerbound on $a_r + a_l$:

$$(a_r + a_l) \geq \frac{s}{4} + 4\frac{a_r a_l}{s} \tag{5.2}$$

By condition $Cond3$:

$$s^2 - (a_r + a_l)s + a_r a_l < \frac{s^2}{2}$$

This implies a following lowerbound on $a_r + a_l$:

$$(a_r + a_l) > \frac{s}{2} + \frac{a_r a_l}{s} \tag{5.3}$$

By equations 5.1 and 5.2 it must be that :

$$\frac{s}{3} + 3\frac{a_r a_l}{s} \geq \frac{s}{4} + 4\frac{a_r a_l}{s}$$

that is:

$$\frac{a_r a_l}{s} \leq \frac{s}{12} \tag{5.4}$$

By equations 5.1 and 5.3 it must be that :

$$\frac{s}{3} + 3\frac{a_r a_l}{s} \geq \frac{s}{2} + \frac{a_r a_l}{s}$$

that is:

$$\frac{a_r a_l}{s} \geq \frac{s}{12} \tag{5.5}$$

Hence, it may be assumed that $\frac{a_r a_l}{s} = \frac{s}{12}$. Substituting in 5.1 and 5.2, it must be $a_r + a_l = \frac{7s}{12}$. Then a_r and a_l are the only roots of of the following quadratic polynomial.

$$X^2 - (a_r + a_l)X + a_r a_l = 0$$

that is:

$$X^2 - \frac{7s}{12}X + \frac{s^2}{12} = 0$$

Hence $a_l = \frac{s}{4}$ and $a_r = \frac{s}{3}$. But this implies that $(s - a_r)(s - a_l) = \frac{s^2}{2}$. Hence *Cond*3 does not hold. A contradiction. □

Hence, by all the previous Claims, $P(s)$ holds. This concludes the proof of Theorem 5.1.1. □

Let us point out that Lemma 5.1.2 can be easily generalized so that the lower bound of Theorem 5.1.1 holds regardless of the number $k \geq 2$ of agents used by a solution algorithm.

5.2 Matching Upper Bound

Now the description of an algorithm whose cost matches the lower bound up to the multiplicative constant is presented.

The algorithm is still based, as in [38], on the idea of recursively dividing the workload between the agents selecting disjoint unexplored areas; in this case however, the unexplored area is divided into three parts of roughly equal size (L,R,C), instead of two (See Figure 5.6). The two co-located agents are initially assigned two of those parts (L and R) and either one or both (depending on the location of the black hole) will eventually complete the task. The first agent completing its part will take care of C leaving a note for the other in the homebase. Note that the location of the homebase changes during execution of algorithm and it consists of the central node

of the current explored area. The second agent (if it returns) will reach the first and divide again the unexplored area in three parts. The algorithm continues until the unexplored area contains a single node. The details of algorithm TRISECT to locate the black hole using two mobile agents is reported in Algorithm 5.1, where E_i and U_i denote the explored and unexplored areas at step i.

In the following assume, for simplicity, that the size of the unexplored area is always divisible by 3. If this is not the case, the algorithm would work in the same way employing the appropriate rounding operations.

Algorithm 5.1 Algorithm TRISECT

Two co-located agents l and r with $E_0 = \{v_h\}$, $U_0 = V - E_0$.
At Stage i (starting with $i = 1$):

1. Agent l cautiously explores L_{i-1}, i.e., the left most $|U_{i-1}|/3$ nodes of the unexplored area, and returns to the homebase of E_{i-1}.
2. Agent r cautiously explores R_{i-1}, i.e., the right most $|U_{i-1}|/3$ nodes of the unexplored area and returns to the homebase of E_{i-1}.
3. The agent reaching the homebase first (say l) leaves a message indicating that it is going to explore the middle $(1/3)^{rd}$ of the unexplored area C_{i-1}.
4. Agent r (if it reaches the homebase) reads the message and travels until reaching the last node explored by l. It leaves an update message for agent l to divide again in three parts the unexplored area so that both agents agree on: explored area E_i, unexplored area U_i, and new homebase (chosen to be the center of E_i).Agent r then goes back to the right-most explored node.
5. Agent l, if agent r has not left a message for it in stage i, when it finishes exploring C_{i-1}, travels until reaching the last node explored by r. It leaves a message for agent r at the last explored node at r's side to divide again in three parts the unexplored area so that both agents agree on: explored area E_i, unexplored area U_i, and new homebase (chosen to be the center of E_i). Agent l then goes back to the left-most explored node.
6. If U_i is a singleton, the surviving agent terminates; otherwise, each surviving agent proceeds to stage $i + 1$.

Theorem 5.2.1. *Algorithm* TRISECT *solves* BHS *using two agents performing at least* $3n\frac{\log_2 n}{\log_2 3} + O(n)$ *moves in the worst case.*

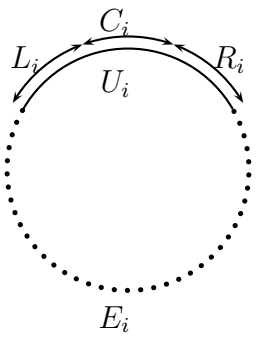


Figure 5.6: Stage i of Algorithm TRISECT.

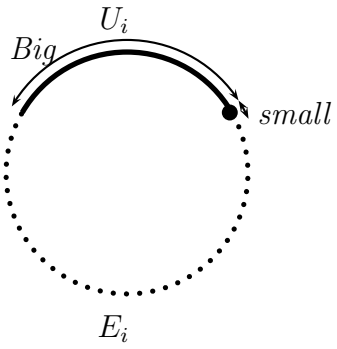
Proof. The correctness of the algorithm follows the same lines as the one in [38]: it is evident that the agents always explore disjoint segments of the unexplored area. Hence at least one agent survives every stage. The surviving agent(s) continue to (collaboratively) explore disjoint unexplored areas until all but one node are declared safe. Hence the protocol solves the black hole search problem in finite time.

Now the complexity will be established. Since the algorithm partitions the unexplored area into three segments of sizes differing by at most one, in stage i of the algorithm, and when the unexplored area reduces to a third (i.e., beginning of next stage, $i+1$), one of the following should happen:

1. Agent l and r finish exploring L_{i-1} and R_{i-1}.
2. Agent l finishes exploring L_{i-1} and C_{i-1}.
3. Agent r finishes exploring R_{i-1} and C_{i-1}.

In all three cases the following, and only the following, movements are performed:
(a) a movement from one end to the center of the previous explored area E_{i-1} and return;
(b) a movement from one end to the other end of the previous explored area E_{i-1} and return.

In each stage, at most $3n$ moves are made across the explored area. Since the unexplored area reduces exponentially, The moves made during the actual exploration through the unexplored area which make up a total of $O(n)$ during the whole



the *small* agent explores the first unexplored node (black circle);
the *big* agent explores the remaining unexplored area (bold black line).

Figure 5.7: Algorithm BIG-SMALL

algorithm.

Since it is that $|U_i| \leq \frac{1}{3}|U_{i-1}|$, the agents perform at most $\frac{\log_2 n}{\log_2 3}$ stages to obtain a single unexplored node for a total of at most $3n\frac{\log_2 n}{\log_2 3} + O(n)$ moves.

□

5.3 A Variant Achieving Optimal Time

Now a protocol which achieves *asymptotically* optimal ideal time, with an *exact* optimal number of agents and of moves is presented. In fact, the protocol employs 2 agents, has ideal time complexity $\Theta(n)$ and performs $3n\frac{\log_2 n}{\log_2 3} + O(n)$ moves.

The idea behind the new algorithm OPTIMAL is to combine Algorithm TRISECT, with another algorithm (BIG-SMALL) which is described subsequently. The main principle of the BIG-SMALL algorithm is to have the two agents play asymmetric roles. The unexplored area is in fact divided into two parts of different sizes to which agents are assigned: one part containing a single node and the other containing all other unexplored nodes (See Figure 5.7).

In the following denoted that the agent assigned to the small area is *small* and the other is *big*. Whenever the *big* agent returns after completing its task, the black hole is obviously located, being the only node under exploration by the small agent. On the other hand, if the *small* agent returns, the task might not be completed and

another stage might be needed.

Algorithm Optimal consists of a preliminary phase resembling algorithm Trisect where the unexplored area is divided in three portions (L, C, and R) and the two agents start exploring L and R. When one of them returns to the homebase, it takes charge of C after leaving a message for the other agent. Within finite time, one of the two agents will complete exploring the assigned region. At this point the execution of (at most three stages of) Big-Small starts. After this execution, if the black hole is not located yet, the algorithm continues following the rules of Trisect.

In the following when it is said that an agent acts as *small* it is meant that that the agent cautiously explores the *first* node in its direction (i.e., right if r, left if l) of the unexplored area. When an agent acts as *big* it cautiously explores in its direction *all but the last* node of the unexplored area. As in Trisect, the location of the homebase in the various steps of the algorithms is variable and it is always the central node of the current explored area. An *update* message contains the update information about the current unexplored area and the current location of the homebase. The details are given in Algorithm 5.2.

Theorem 5.3.1. *Algorithm* Optimal *solves the black hole search in ideal time* $\Theta(n)$ *using 2 agents and performing* $3n\frac{\log_2 n}{\log_2 3} + O(n)$ *moves.*

Proof. Since the segments of the ring explored by each agent are always disjoint, at least one agent survives after performing its current task. After the end of phase 0, at least an agent, say r, survives and returns. During phase 1, in the first stage of Big-Small, agent r becomes *small*, l becomes *big*, and they divide the area to be explored into two disjoint segments. If the *big* agent l returns, the problem is solved. If the *small* agent r returns, the unexplored area is again divided into two disjoint segments and the second stage of Big-Small is started, with the right agent r being *small* again. Also in this stage, if the *big* agent l returns, the black hole is located. If the *small* agent r returns, the third and final stage of Big-Small is started in which r becomes *big*. If r returns the black hole is located. If l returns, it goes to the other end, leaves a message to notify to begin Algorithm Trisect, updating the new explored and unexplored areas. Correctness in the subsequent execution follows from the correctness of algorithm Trisect. Hence, algorithm Optimal solves the

Algorithm 5.2 Algorithm OPTIMAL

Initially: two co-located agents l and r . $E = E_0 = \{v_h\}$. $U = U_0 = V - E$.

Phase 0 (Preliminary Phase):

1. Agent l cautiously explores L_0, i.e., the leftmost $(|U_0|)/3$ nodes of the unexplored area and returns to the homebase.
2. Agent r cautiously explores R_0, i.e., the right most $(|U_0|)/3$ nodes of the unexplored area and returns to the homebase.
3. The agent finishing the exploration first (say l) leaves a message at homebase that it is exploring the middle $(1/3)^{rd}$ nodes (C_0).

/* now the agents are exploring two disjoint areas: C_0 and either L_0 or R_0. */

Phase 1 (Big-Small Phase)

1. (*Stage 1*) The first agent that completes its assigned exploration, say r, moves to the last explored node on agent l's side, it leaves an *update* message to l indicating to become *big*. Agent r then moves to its side and acts as *small* (i.e., it cautiously explores the **first** unexplored node on its side). If/when l finds the message, it acts as *big* (i.e., it cautiously explores all but the *last* unexplored nodes on its side).
2. If the *big* agent l completes its work, then the black hole is located and the algorithm terminates.
3. (*Stage 2*) Otherwise, the *small* agent r, after exploring the assigned node, moves to the last explored node on agent l's side, and leaves an *update* message for l indicating to start the second stage in the same role (*big*). Agent r moves then back to its side, again acting as *small*. If/when l finds the message, it updates the assigned area and starts the second stage as *big*.
4. If the *big* agent l completes its work, then the black hole is located and the algorithm terminates.
5. (*Stage 3*) Otherwise, the *small* agent r, after exploring the assigned node, it moves to the last explored node on l's side, it leaves an *update* message for l. Agent r then moves to the other side and it changes role acting now as *big*. If r completes its work, then the black hole is located and the algorithm terminates.
6. If/when agent l finds the update message, it moves to agent r's side and leaves an *update* message to r indicating to perform algorithm TRISECT on the new unexplored area.

Algorithm OPTIMAL – continuation

/* at this point one agent start TRISECT, the other (if still alive) will join */

Phase i (Trisect Phase)

1. Agent l starts to perform TRISECT. If/when r finds the message, r joins l and they perform TRISECT together.

black hole search problem.

To prove that the algorithm has $\Theta(n)$ ideal time complexity, first observe that when the first stage of BIG-SMALL begins, the size of the explored area is at least $\frac{2}{3}n$.

For the *big* agent l to complete its exploration (assuming it does not die before) it would take $3\frac{n}{3} = n$ moves and time because each link is traversed with cautious walk. That is, by that time, in a synchronous execution, if l does not die, it has discovered the black hole.

Agent r, if it does not disappear in the black hole during its exploration as *small*, performs two stages as *small* traversing the explored area back and forth twice and thus making at least $\frac{8}{3}n$ moves and spending the same amount of time. This means that, since $\frac{8}{3}n > n$, if r successfully completes its two stages as *small* and starts the third stage as *big*, in a synchronous execution agent l must have died by that time; hence the algorithm terminates within an additional $O(n)$ time units. Summarizing, in a synchronous execution, within $O(n)$ time, either r dies in the black hole within the first two stages as *small*, and l would discover the black hole while it is acting as *big*, or l dies in the black hole and r discover it while *big* in the third stage of BIG-SMALL.

Consider now the number of moves. Since the agents perform Algorithm TRISECT except for the BIG-SMALL phase which clearly costs $O(n)$ moves, the overall number of moves is $3n\frac{\log_2 n}{\log_2 3} + O(n)$. □

5.4 Summary

In this chapter we have considered the black hole search problem in the ring focusing mostly on move complexity. We have shown lower and upper bounds which are exact

up to the constant of proportionality: Algorithm TRISECT achieves such an optimal bound. Finally, we have also modified Algorithm TRISECT to obtain a solution (Algorithm OPTIMAL) that achieves exactly optimal number of moves and team size and asymptotically optimal time complexity. Next we turn our attention to black hole search on arbitrary unknown graphs using tokens.

Chapter 6

Black Hole Search with Tokens in Arbitrary Graphs

In this chapter the black hole search problem is considered under the token model, where communication and coordination are achieved using t pebbles that an agent can pick up, carry, and drop on the nodes. If the graph is unknown to the agents, $\Delta + 1$ agents are necessary, and solution protocols exist with those many agents.

BHS in arbitrary unknown network is studied using the whiteboard model and enhanced token model. In the whiteboard model, the algorithm presented in [34] solves BHS in $O(n^2)$ moves using whiteboards of size $O(\log \delta)$ at nodes of degree δ. In the enhanced token model, the algorithm presented in [33] solves BHS with $O(\Delta^2 m^2 n^7)$ moves for a network with n nodes, m links and maximum degree Δ.

It has been recently shown that, in this setting, if the agents have a map of the graph, the problem can be solved with $t = O(1)$ pebbles in total; furthermore this can be done without increasing the size of the team.

The immediate open question is whether or not the same occurs also when the agents do not have a map, and, if so, with how many agents. The contribution of this chapter is to provide a definite answer to both these questions.

It is first proven the unexpected negative result that if $t = O(1)$, then $\Delta+1$ agents are *not* sufficient. It is then shown that, regardless of the team size, $t = 2$ pebbles are *not* sufficient. In other words, any solution protocol using a bounded number of pebbles, must consist of at least $\Delta + 2$ agents and use at least $t = 3$ pebbles. Finally, it is shown that these bounds are *tight* by presenting a protocol that locates a black hole in an unknown anonymous graph using only 3 pebbles and $\Delta + 2$ agents. The proposed protocol is rather complex; it employs as primitives a series of novel token-based communication protocols.

Most results of this chapter appeared in [6].

6.1 Impossibilities and Lower Bounds

Recall that $\Delta + 1$ agents are sufficient to locate the black hole if the token load is a function of the size of the network, and thus a priori unbounded [33, 34]. In this section we show that if the total number of tokens is bounded, $\Delta + 1$ agents no longer suffice.

The proof of this result is based on a game between an algorithm and the adversary. The adversary has the power to choose the graph, the port labeling and the asynchronous timing of the edges; the algorithm specifies agent movement; w.l.o.g. we can assume the steps of the algorithm and adversary alternate. At any moment of time, the agents can be classified as *free* and *blocked*. The free agents are located at the nodes, or traversing an already explored edge (an edge is unexplored if no agent has so far crossed it in either direction). The algorithm specifies in its step the movement of the free agents located at the nodes; w.l.o.g. we may assume the algorithm specifies movement (the port taken) of at most one free agent in its step. The *blocked* agents are the agents which are in transit over the yet unexplored edges. In its step, the adversary lets the free agents crossing edges reach their destinations. Furthermore, if the algorithm specifies that an agent enters an unexplored edge with a given label, the adversary chooses which of the incident unexplored edges the agent enters and assigns that label to the edge. Finally, the adversary can unblock a blocked agent and allow it to reach its destination.

The adversary is limited by the requirement that it can indefinitely block only the agents crossing the links entering the black hole. However, due to asynchrony, the unknown topology and the unknown location of the black hole, this means that the only case when the adversary cannot block the blocked agents indefinitely is if there are at least $\Delta + 1$ blocked agents or there are two blocked agents traveling on edges leaving the same node. In such a case, the algorithm can wait until the adversary unblocks one of the blocked agents; in all other cases, the algorithm must specify an agent movement in its step. We can now state:

Theorem 6.1.1. *When the overall number t of available tokens is less than a constant C, black hole search is unsolvable by $\Delta + 1$ agents without a map, even if the number of nodes and edges is known.*

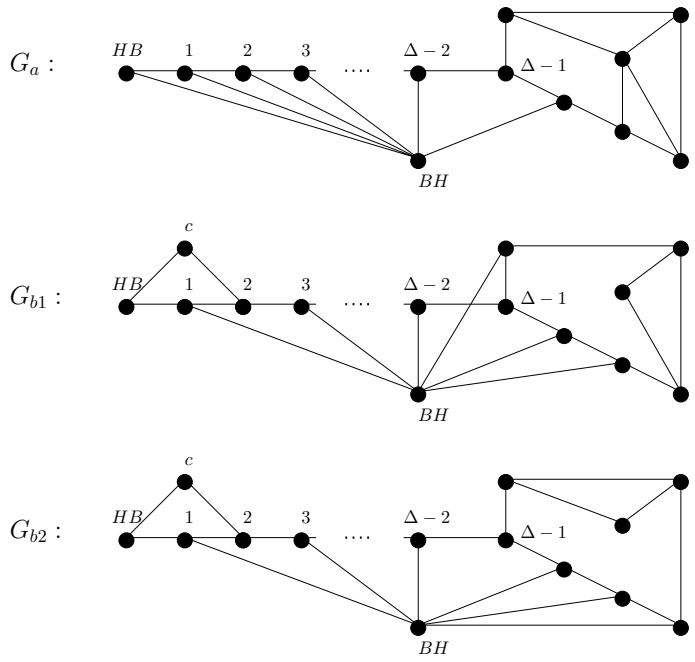


Figure 6.1: The graphs used in Theorem 6.1.1

Proof. By contradiction, assume $\Delta + 1$ agents are sufficient. The unknown graph G in which the agents must locate the black hole is one of G_a, G_{b_1} or G_{b_2} shown in Figure 6.1, where HB is the homebase, and BH is the black hole; the choice of which graph G really is up to the adversary. We now show that, even if the agents have the map of these three graphs, the uncertainty about which one G really is allows the adversary to send all $\Delta + 1$ agents to the black hole.

Starting from the homebase $HB = x_0$, whenever the algorithm sends an agent over the first unexplored link from node x_i $(0 \leq i \leq \Delta - 2)$, the adversary blocks it. Since G might be graph G_a, the algorithm must send an agent also over the second unexplored link incident on i. At this point, the adversary is forced to unblock one of these two links; it will unblock the link leading to node x_{i+1}. Hence, within finite time, $\Delta - 1$ agents are blocked and an agent reaches node $x_{\Delta-1}$.

The two free agents must now try to explore the remainder of the graph, since the adversary might have chosen $G = G_a$. In doing so, the adversary can force both of them to explore a link leading to the black hole by choosing either $G = G_{b1}$ or $G = G_{b_2}$, depending on the way these two agents try to explore the remainder of the

graph. Notice that at this point, all agents are blocked on links leading to the black hole, except for the two agents traversing the edges leading to node c. The adversary then unblock them; to finish the exploration of the graph, these two agents must reach node $x_{\Delta-1}$ and go beyond.

Regardless of the choice, $G = G_{b1}$ or $G = G_{b_2}$, made by the adversary, at each of the nodes $x_3, x_4, x_{\Delta-2}$ on the path to node $x_{\Delta-1}$, there is a link leading to the black hole, which the two agents must avoid. This means there are $2^{\Delta-5}$ possible labeled paths from x_3 to $x_{\Delta-1}$, only one of which is safe. However, the algorithm can place only C tokens on the Δ nodes $c, x_0, ..., x_{\Delta-2}$, resulting in $O(\Delta^C)$ possible token configurations. Hence, for large enough Δ the adversary can force also these last two agents to enter the black hole while they are moving from node x_3 to node $x_{\Delta-1}$. □

As a consequence, at least $\Delta + 2$ agents are required to locate the black hole.

We now focus on the needed token load t, i.e., how many tokens are really necessary.

Theorem 6.1.2. *With two tokens initially stored at the homebase and no additional tokens, no team of anonymous identical agents can locate the blackhole in an unknown network, even if the number of nodes and edges is known.*

Proof. By case analysis over the possible algorithms, we show that in every case the adversary can cause the algorithm to fail.

Let us call an agent that is in initial state a *fresh* agent. Note that thanks to asynchrony, it is sufficient to limit ourselves to algorithms which perform actions only on wake-up, when arriving to a node, or when noticing a change in the number of pebbles at a node they are currently waiting at.

If the algorithm's action for a fresh agent seeing two pebbles at the homebase does not change the number of pebbles, then the adversary wakes-up the fresh agents one by one and either all start waiting forever at the homebase, or all leave the homebase towards the same vertex v_1. In the first case, the algorithm obviously fails, in the second case the adversary places the black hole at v_1 and the algorithm fails.

Consider now the case that a fresh agent seeing two pebbles at the homebase picks both of them up. If the agent leaves the homebase, the adversary will direct it towards

the black hole and both tokens will be lost, leaving no possibility for the remaining agents to communicate and the algorithm fails. On the other hand, if the first agent starts waiting at the homebase, it will wait forever, again effectively eliminating the pebbles from the play and causing the algorithm to fail.

Therefore, the algorithm for a fresh agent seeing two pebbles at the homebase has only two options: (*O1W*): pick one pebble and wait at the homebase until the number of pebbles changes to 0, or (*O1M*): pick one token and move to a neighbouring node v_1.

Note that if a fresh agent seeing one pebble does not pick it up, the adversary can make all remaining fresh agents do the same thing. If all of them wait, the adversary places the black hole at v_1 and the algorithm fails; if all of them leave towards v_2 then the adversary places the black hole at v_2 and the single remaining agent cannot locate it. Therefore, there are only two options for the second agent: (*O2W*): pick up one pebble and wait at the homebase until a pebbles appears there, or (*02M*): pick up one pebble and leave (to a neighbour v_2 of the homebase).

Consider now what a fresh agent seeing no tokens can do: either leave the homebase towards its neighbour v_3, or wait in the homebase until a token or two appear there. However, in the first case the adversary can place the black hole in the vertex v_2 and wake-up the fresh agents one by one until all of them enter the black hole. Since $\Delta \geq 3$, the two surviving agents are insufficient to locate the black hole. Therefore we may assume a fresh agent seeing no tokens at the homebase will wait.

Let us analyze the possible combinations for the actions of the first two agents:

Case *01M* and *O2M*: The adversary will place the black hole at v_2 and allow the first agent to travel among homebase and its neighbours, while the fresh agents are kept asleep. Eventually, the first agent will either (a) enter v_2, (b) leave a neighbour of the homebase via unexplored link, (c) start waiting at a neighbour of the homebase or (d) drop a token at homebase (and do something). In case (a) both tokens disappear at the black hole and the algorithm fails. In case (b) the adversary will direct that unexplored link to the black hole and the same happens. In case (c) all agents are either dead or waiting and the algorithm fails. Finally, in case (d), the adversary

wakes-up the third agent, which will pick up the token and enter the black hole (the algorithm according to *O2M* case), again eliminating both tokens.

Case *O1W* and *O2M*: The adversary puts the black hole at v_2. Once the first agent notices that a token disappeared from the homebase, it will perform one of the actions (a), (b), (c) or (d) from the previous case and the adversary will deal with that in the same manner as before.

Case *O1M* and *O2W*: The adversary puts the black hole at v_1 and the first agent disappears there. Therefore all other agents will remain waiting at the homebase and the algorithm fails.

Case *O1W* and *O2W*: The adversary wakes-up the first agent while keeping the fresh agents asleep. Consider now the possible actions of the first agent once it notices that there are no tokens in the homebase. If the agent leaves the homebase without dropping its token, the adversary will direct it towards the black hole and everybody else will remain waiting. Therefore the first agent has to drop its token. Using asynchrony, the adversary will make sure that the second agent waiting for appearance of this token will not notice this. Instead, a fresh agents is woken-up, which (according to *O2W*) picks up the token and starts waiting. At this moment both tokens are effectively out of play as the agents holding them are waiting and the algorithm fails.

□

Finally we look at the number of moves necessary for black hole search in an unknown graph.

Theorem 6.1.3. *In an unknown graph, black hole search by* $\Delta+2$ *agents costs* $\Omega(n^2)$ *moves in the worst-case.*

Proof. In [34] it has been shown that with $\Delta + 1$ agents, the worst-case complexity of BHS in the whiteboard model is $\Omega(n^2)$. The proof is based on a specific graph with two exploration fronts, showing that the execution can be divided into phases, and that at the end of each phase, $\Delta - 1$ agents must cross the already explored middle part from one front to another in order to bring the total number of agents at that front to Δ. If there are more agents available instead of $\Delta + 1$, the whole proof

works as is; the only difference is that each excess agent reduces the number of agents that must cross the middle part. Still, if the number of crossing agents is $\Omega(\Delta)$, the $\Omega(n^2)$ lower bound follows. This yields $\Omega(n^2)$ lower bound for $a\Delta$ agents for $a < 2$, in particular for $\Delta + 2$ agents as well. □

6.2 Possibility and Upper Bounds

6.2.1 Overview

The basic idea of Algorithm BLACK HOLE SEARCH is to have one designated agent (the leader) coordinating and directing the exploration activities by the other agents (followers). The leader itself collects the map of the explored graph, identifies the black hole and at the end instructs the surviving follower(s) to terminate. We assume the knowledge of the number of agents $k = \Delta + 2$ is available to the agents.

The communication between the leader and the followers is performed using the communication module described in the following subsection. Algorithm BLACK HOLE SEARCH consists of three phases: *Leader Election and Initialization*, *Waiting Room Location, and Synchronization* and *Search*. The goal of the first phase is to elect a unique leader, and have all other agents become followers. In particular, this phase ensures that all agents join the computation and no agent remains sleeping. The purpose of this is to enable reuse of token configurations at the homebase.

As the exploration proceeds (and especially at the beginning), there might be the case that the exploration front (the number of unexplored edges leaving the already explored component) is smaller than the number of available agents. In order to avoid the unused agents cyclically asking the leader for work (causing potentially unbounded number of token transitions), the leader sends them to sleep and wakes them up once there is work for them. However, this sleeping cannot be done at the homebase, as the number of possible configurations is too low. Instead, a safe node (the *waiting room* (WR)) neighbouring the homebase is identified and used. The goal of the second phase is to identify this waiting room and ensure no interference with the third phase.

Once the waiting room is identified and necessary synchronization performed, the

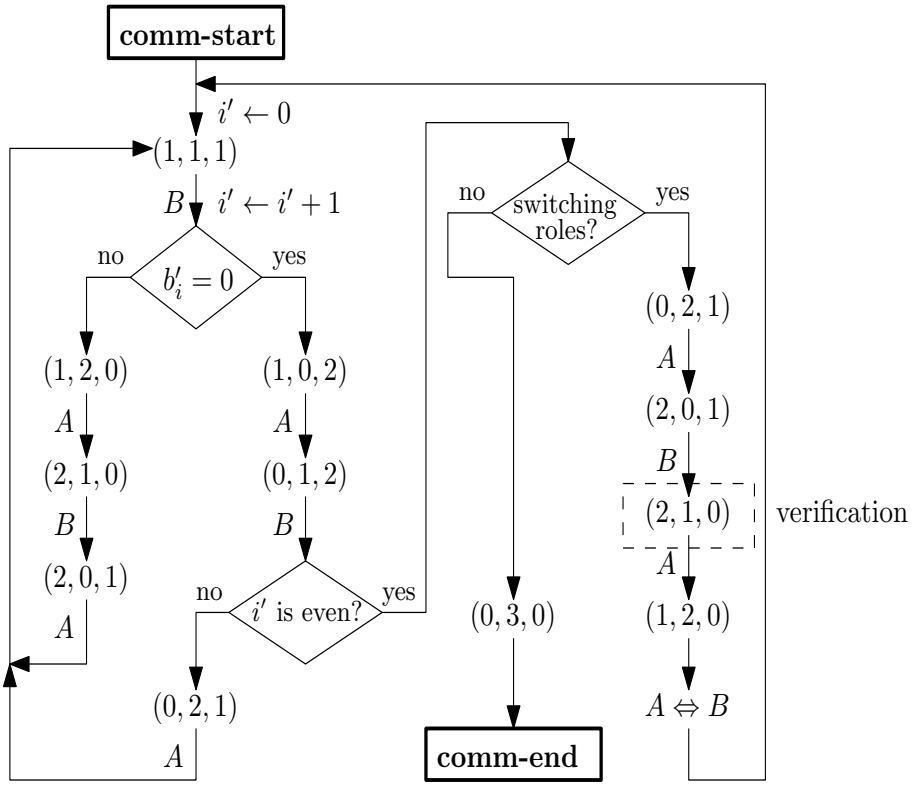
algorithm proceeds to the third phase, in which the leader coordinates the search of the graph by the followers, until the black hole is located. The protocol is designed in such a way, that there will always be at most one follower communicating with the leader; all other followers are either searching, sleeping or waiting for their turn to communicate.

6.2.2 Communication Using Tokens

At the core of our algorithm is a simple novel technique that uses three tokens to communicate finite but arbitrary information (a sequence of bits) between two agents: A and B. All communication takes place in the home base. Let (x, y, z) denote the configuration of tokens in which A has x tokens, B has z tokens, and the homebase contains y tokens. Let $\mathcal{S} = b_1 b_2 \cdots b_r$ be a sequence of r bits to be transmitted from B to A and let $\mathcal{S}' = b_1 1 b_2 \cdots 1 b_{r-1} 1 b_r 0$. The actual sequence that will be transmitted from B to A is $\mathcal{S}'$; in doing so, 1-bits in even positions will be discarded, while the presence of a 0-bit in an even position will indicate termination.

Communication always starts from configuration $(1, 1, 1)$, with B's turn. B indicates the next communicated bit by either picking or dropping a token. The subsequent moves transition the configuration back to the initial one, allowing communication of the next bit. The agents alternate their moves; due to asynchrony, an agent X knows that the other agent has observed X's action and performed its step only when observing a change of the configuration left by X. No other agents interfere with the communication, as all other agents present in the homebase wait until configuration $(0, 3, 0)$ appears – which happens only when the communication has been completed.

In the protocol, there is a provision for a two-way communication: After B has completed transmission of its message to A (indicated by 0 on an even position), it either indicates end of communication by setting a terminal configuration $(0, 3, 0)$, or signals (by configuration $(0, 2, 1)$) the need for switching roles between A and B. Consult Figure 6.2.2 for the full communication protocol.



The agent performing the move is indicated to the left of the corresponding transition arrow. In the search phase, the leader perform the *verification* protocol before performing its action on the configuration marked *verification*.

Figure 6.2: The communication subroutine

6.2.3 Leader Election and Initialization

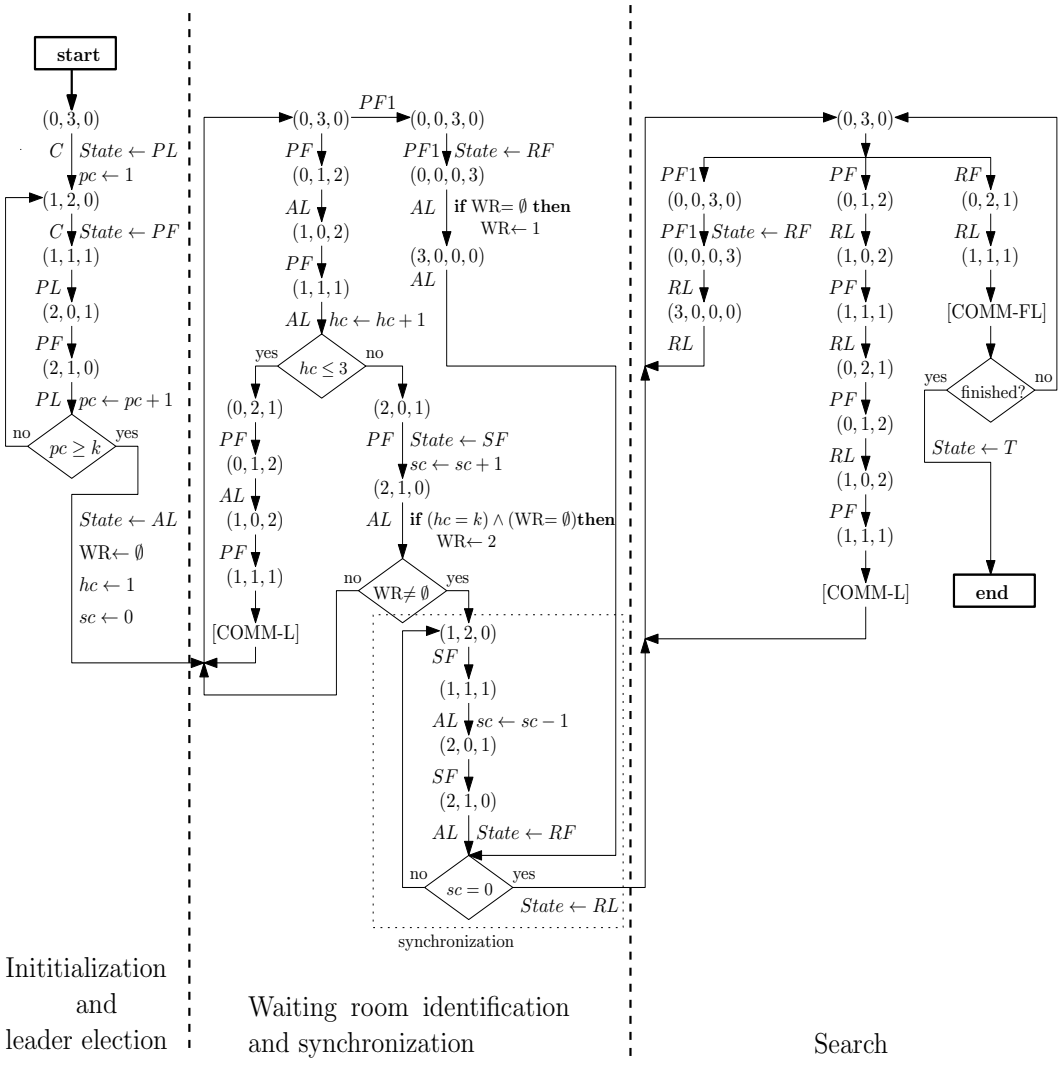
Figure 6.2.3 describes the token manipulation, the agent's state and the leader's variable transitions of Algorithm BLACK HOLE SEARCH. As with the communication module, the state of the tokens is described as a triple (x, y, z), where x is the number of tokens held by the leader, y is the number of tokens placed in the homebase, and z is the number of tokens held by the follower communicating with the leader.

Initially, all agents are in initial state C (*candidate*). The first agent to wake-up observes three tokens in the homebase, picks up one token and becomes the *proto-leader* (PL). An agent in state C observing two tokens picks up a token to register with the leader and becomes a *proto-follower* (PF). After some token manipulation (see Figure 6.2.3, left) the configuration is back to two tokens at the homebase and the next candidate can register. The leader is counting the registered followers and once $k-1$ of them have registered, it knows that all agents have woken-up, it transitions to state *acknowledged leader* (AL) and drops the tokens it holds so that the homebase now contains 3 tokens. From now on, 3 tokens in the homebase means that the leader is ready for interaction with followers (for agents in state C it means that no leader has been elected yet; however, the initialization phase ensures that there are no agents is state C left).

6.2.4 Waiting Room Location and Synchronization

The goal of this phase is to identify a safe node adjacent to the homebase. If the homebase is of degree one, its only neighbour is the waiting room and, at the end of the first phase, the leader becomes the *ready leader* (RL) and the followers upon registration directly become *ready followers* (RF) instead of *proto-followers*.

Consider now the case that the homebase is of degree at least two. The second phase starts in configuration $(0, 3, 0)$, with the leader making the last turn. A follower in state PF observing 3 tokens takes two tokens to indicate that it wants to help. After some token manipulation (see Figure 6.2.3, the leftmost chain of the middle phase), the leader sends the first two such followers (now in states $PF1$ and $PF2$) to explore the first and the second adjacent link. While these followers are exploring, other followers want to help too. The leader sends them to sleep in state *sleeping*



On the left of transition arrows is the state of the agent performing the transition. On the right are the actions performed by the agent. Note that the states are in fact groups of states, with sub-states associated with concrete locations in the diagram. COMM-L refers to a call to the communication subroutine in which the leader serves as agent B, without switching roles. In COMM-FL, first the follower acts as agent B, informing the leader; then the roles are switched and the leader tells the follower what to do next.

Figure 6.3: Token manipulation in Algorithm Black Hole Search

follower (SF). As the waiting room has not been identified yet, these followers "sleep" at the homebase, waiting until two tokens appear there.

Eventually, the follower sent to the first link and/or the one sent to the second link returns back to the homebase. If a follower returns from the first link, it waits until the leader is ready to communicate with it (i.e., when it sees 3 tokens in the homebase), grabs the three tokens and brings them to the node it went exploring. The leader knows that the only way all tokens can disappear is if the first follower found the first link safe, so it goes there and picks up the tokens (see Figure 6.2.3, middle top, where the fourth element in a tuple denotes the number of tokens in the node adjacent to the homebase over the first link). If the waiting room has not yet been identified, this node will be the waiting room.

If the second follower returns from the second link, it cannot communicate with the leader in the same way. Instead, it follows the protocol of a proto-follower trying to help. Eventually, (if the first follower meanwhile did not return) the leader will note that there were k proto-followers trying to help. Since there are altogether $k-1$ proto-followers, one must have done so twice, i.e., the second follower has returned and the second link leads to a safe node.

At the moment the waiting room is identified, the leader wants to become the *ready leader* (RL) and to transition to the main *search* phase. However, there may be sleeping followers waiting to be woken up (i.e., for two tokens in the homebase). Therefore, before becoming RL, the leader must wake all of them up into the *ready follower* (RF) state. This is achieved by putting two tokens in the homebase and subsequent token manipulation as shown in the lower right loop of the middle phase in Figure 6.2.3. The leader maintains variables *hc* (counting the followers trying to help) and *sc* (the number of followers put to sleep) to help it perform these tasks.

Note that at the end of the second phase, the waiting room is identified, the leader is in state RL and there are no followers in state SF. However, there might be followers in state PF1, PF2 and PF. As we will see in the next subsection, this pose no problem for the algorithm.

Search

The search is coordinated by the leader, which creates, maintains and updates a partial map of the explored component, until the location of the black hole is determined. As in the previous phase, configuration $(0, 3, 0)$ means that the leader is ready to interact with a follower.

Depending on what kind of follower starts the communication ($PF1$, PF or RF) the leader responds accordingly (consult Figure 6.2.3, the rightmost phase). In the first two cases, the follower is told to become *ready follower*, in the latter case a two-way communication between the leader and follower is performed. The follower tells the leader the possible result of its exploration journey, if any; the leader responds by a next work order, a command to go to wait to the waiting room or by indicating termination if the black hole has already been located.

The work order is specified as a sequence of link labels identifying a path π to be explored, with only the last edge being yet unexplored. A follower receiving such an instruction follows the path π and, if successful, returns back to the leader to report the degree of the last node v. The leader maintains the invariant that at any moment, at most one agent is exploring a given unexplored link. This ensures that at most $\Delta = k - 2$ agents enter the black hole, i.e., at least one follower survives.

When a follower reports that a path π leads to a safe node v, the leader must locate v in the partially constructed map. In particular, it must determine whether v is a new node, or an already visited node reached by the newly explored edge e at the end of π. To do so, the leader interrupts the communication during the switching roles phase at the moment when it holds two tokens and the homebase holds one token (in the node marked as *verification* in Figure 6.2.2) and determines the location of v as follows: It first traverses π until it comes to v, drops two tokens there[1], then returns to the already known part via e. The leader then visits all nodes explored so far, using the map it has constructed. If a node with two tokens is encountered, v has been visited before and only e is added to the map, otherwise, v is a new node and both e and v are added to the map. The leader then collects the tokens from v, brings them to the homebase and completes the communication with the follower.

[1] it can happen that v is the homebase, this is easily identified by finding one token there

If the black hole has not yet been located, the leader either gives the follower a new work order, or sends it to wait in the waiting room if all currently unexplored edges are being explored by other followers. If new unexplored edges appeared and there are waiting followers in the waiting room (the leader maintains a counter of them), the leader goes there and wakes one of them using the following protocol: When finishing the communication, instead of ending in state $(0, 3, 0)$, the leader brings the three tokens to the waiting room. The first waiting follower to see three tokens picks them up and places two at the homebase[2]. The leader returns to the homebase, picks one token and now the leader and freshly ready follower are ready to communicate.

Finally, if the black hole has been located and there are still ready or waiting followers, the leader wakes up the waiting followers (if needed), waits for the work requests of ready followers and tells them to terminate. Once there are no ready followers, the leader terminates.

6.2.5 Correctness and Complexity

One way to look at the Algorithm Black Hole Search is that each of the super-states has a "main-loop" sub-state in which the follower waits for the right to interact with the leader: C state waits for either 3 or 2 tokens in the homebase, SF state waits for 2 tokens in the homebase, WF state waits for 3 tokens in the waiting room and all other follower states wait for 3 tokens in the homebase.

The following lemma captures the crucial, designed-in, property of the algorithm:

Lemma 6.2.1. *At any moment in time (after the leader has been selected), there is at most one follower interacting with the leader, all the other followers are either exploring, or waiting in their respective main-loop sub-states. Furthermore:*

(1) Eventually, the first phase is completed by an agent reaching state AL. At that moment, there are no agents left in state C and all agents are in state PF.

(2) Eventually, the second phase is completed by an agent reaching state RL. At that moment, a safe waiting room is correctly identified and there are no agents

[2]This might look like signal for yet another switch of roles with the follower the leader previously communicated. However, from the context that follower knows that no further role switching is needed terminates its interaction with the leader.

in state SF.

Proof. In the first phase, an agent A in state C starts interaction with the leader only when seeing two tokens in the homebase. At that moment, A removes a token from there, preventing another agent from entering the interaction. At the moment two tokens are restored in the homebase, A is already in state PF, waiting for appearance of three tokens in the homebase.

(1) follows from construction: The first agent to wake-up sees three tokens, takes one of them and becomes the leader. As long as there are agents in state C, the leader's counter pc is less than k and the leader cannot progress to the second phase. Eventually, each agent in state C will be awaken and will transition to state PF; when the last one does so, the leader will notice and enter the AL state. As the only way a leader can be created is by an agent C observing three tokens, this implies there will be exactly one leader.

The followers in state $PF*$ (i.e., PF, $PF1$ or $PF2$) in the second stage enter interaction by observing three tokens and removing two of them, again preventing other followers from entering interaction. Since three tokens in the homebase do not appear during the communication subroutine, no other such follower can enter interaction before the previous interaction has run its course. However, a follower in state SF can do so in principle, as it is waiting for only two tokens. However, the first follower in such a state is created only after the communication with the first two helping followers has completed. Since these first two communications are the only calls to the communication subroutine during the second phase, the statement holds also during the second phase.

(2) now follows from construction: Eventually, either the agent exploring the first link or the one exploring the second link returns to the homebase. If the first agent returns first, or if it returns before the last proto-follower registers trying to help, the transition sequence corresponding to state $PF1$ is followed, and the waiting room is set to 1. If the second agent returns and all proto-followers register before the first agent returns, the waiting room is set to 2. In both cases the leader proceeds to release all followers sleeping in state SF into state RF and only after doing so changes its state to RL, completing the second phase.

As there are no agents in state C or SF in the third phase, all followers wait for three tokens to start interaction with the leader. The sequentiality of this interaction then follows from previously used arguments (an agent starting interaction removes a token from the homebase, three tokens are restored only when the interaction has concluded, no three tokens at the homebase are present during the communication subroutine). Note that the same arguments imply sequentiality of interaction in the waiting room as well.

Finally, there is no unwanted interaction due to verification phase: The only potential problem is if the node at the end of the newly explored edge is either the homebase or the waiting room. In the first case, the leader detects this by noticing a token at the homebase, in the second case two dropped tokens do not disrupt the waiting agents, as they wait for three tokens. □

Lemma 6.2.1 means that the token transformations do follow the diagrams from Figure 6.2.3, i.e., there is no unwanted/unexpected interference between the agents.

From Lemma 6.2.1 and by construction we get:

Lemma 6.2.2. *There is no interference with the communication subroutine, which correctly communicates the desired information between the agents.*

Now we are ready for the main theorem:

Theorem 6.2.1. *Algorithm* Black Hole Search *solves* Bhs *using* $\Delta+2$ *agents and* 3 *tokens, at the cost of* $O(mn)$ *agent moves and* $O((m+\Delta n)\log n)$ *token transitions.*

Proof. **Correctness:** From Lemma 6.2.1, the algorithm eventually reaches the third, exploration phase. By construction of the search phase, no two agents are sent to explore the same unexplored edge. Therefore, as there are $k-1=\Delta+1$ followers, at least one of them survives. As the communication between the leader and the followers works as intended (Lemma 6.2.2), eventually all links not leading to the black hole will be explored. As both n and m are known, at that moment the leader identifies the location of the black hole and Bhs is solved.

Cost: Each edge is explored at most twice (at most once from each direction). The cost of exploring an edge is bound by $O(n)$: $O(n)$ moves for the follower to reach

it and return back, $O(n)$ moves for the leader to verify whether it is a new vertex or not (e.g. using a spanning tree of the currently explored component). The possible cost $O(1)$ of waking-up a follower waiting at the waiting room can be charged to the edge it has been woken-up to explore (into this we can hide also the cost of sending it to the waiting room before). Summing over all edges yields $O(nm)$ bound on the total number of moves.

Let us now estimate the total number of token transitions. According to Figure 6.2.3, each follower performs $O(1)$ transitions until it reaches state RL. Subsequently, additional $O(1)$ transitions per explored edge are spent outside the communication subroutine. As the transitions of followers and the leader alternate, the overall number of token transitions outside the communication subroutine can be bound by $O(\Delta + m)$.

Let us now bound the overall number of token transitions during the communication subroutine. There, $O(b)$ token transitions are used in order to communicate b bits. Note that each call to the communication subroutine can be attributed either to a new edge being explored (if the call contains the work order for this edge), or to putting an agent to sleep. As the cost of putting an agent to sleep is $O(1)$ and can be charged to its next work order, it is sufficient to bound the total bit length of all work orders.

Directly encoding the exploration path π of each work order would result in overall $O(mn \log n)$ communicated bits. However, a more efficient scheme can be used: Each follower builds its own knowledge of safe edges, based on where the leader sends it. The leader remembers this information for each follower. When the leader gives a follower A a new edge e to explore, instead of specifying the full path π, it specifies the closest (over safe edges known to the leader) vertex u which is known to be safe to A (e.g. by telling A: "take i-th vertex you learned to be safe") and from there specifies a short(-ened) path π' by which to reach e from u over links known to be safe to the leader, but not to A.

Note that π' never traverses a vertex u' known to be safe to A (otherwise u' would have been used as the starting point of π' instead of u). Hence, $\bigcup \pi''$ is a tree, where π'' is obtained from π' by omitting the last edge and the union is taken over all π' told

to A by the leader. Therefore, the total number of communicated bits to A can be bound by $(n + m_A) \log n$, where m_A is the number of edges explored by A. Summing over all agents yields $O((m + \Delta n) \log n)$ for the overall number of token transitions during the communication subroutine. □

6.3 Summary

In this chapter we studied the BHS with pure tokens. We showed that three tokens are required regardless of the team size. We also established that when the number of tokens available is constant, then $\Delta + 2$ agents are required. Finally, we presented a constructive solution to solve black hole search with $O(mn)$ moves. In the next chapter we generalize the novel communication technique used to solve black hole search.

Chapter 7

Communication with Tokens

In the previous chapter we have solved the black hole search problem using tokens to coordinate the agents' activities. In this chapter we generalize that method introducing a novel technique by which agents communicate manipulating identical tokens.

Consider the simplest situation where two agents wish to communicate in the network: a *sender* has some information (e.g., a string of digits in some base q), a *receiver* is waiting to receive information. Both agents have some tokens available, and they can use dedicated nodes (called *rooms*) to manipulate their tokens, by dropping them and picking them up. Depending on the number of rooms and of tokens available, clearly different mechanisms for communicating information can be devised. In particular, if the resources are not sufficient, it may be impossible for the agents to communicate unambiguously any information. We design several protocols depending on the amount of resources available. We then show that in all the cases not considered, communication is actually impossible.

We then consider the case of communication among $a > 2$ agents focusing on the Broadcast problem: The technique is easily generalizable to other problems. The goal of this chapter is in fact to introduce mechanisms that are general enough to be employed for any problem requiring communication among the agents.

Tables 7.1 and 7.4 summarize our results. Note that, with little resources, communication occurs slower than with more resources available. In fact, it can be seen from the table that when few tokens/rooms are available, information can be communicated using a binary encoding of the string to be transmitted, while with more resources the encoding can be done in higher bases, and thus faster.

7.1 Model

The system is composed of a set $\mathcal{R} = \{R_0, R_1, ..., R_{n-1}\}$ of $n \geq 1$ distinguished *rooms* and a set $\mathcal{A} = \{A_0, ..., A_{k-1}\}$ of $k \geq 2$ possibly identical *agents*. Agents can move freely from every room to every other room; up to k agents can stay in the same room at the same time. In the system there are $p \geq 1$ identical *tokens* which can be placed into rooms, and can be held and carried by agents; initially all tokens are placed in room R_0. When in a room, an agent can pick up any subset of the tokens placed there, and it can place there any subset of the tokens it holds.

The operation of an agent accessing the tokens in a room and possibly altering their number (by dropping some held tokens or picking up some tokens placed there) is *atomic*; at any time any agent wanting to access the tokens in a room will be granted access within finite time. In other words, access to the tokens in a room is granted in fair mutual exclusion. Agents can only see the tokens placed in the room they currently are in, and only upon being granted access.

Time is discrete; however, the amount of time an agent spends moving from a room to another, as well as the time spent in a room before requesting access to the tokens placed there is finite but otherwise unpredictable. In other words, the system is fully *asynchronous*.

A configuration is a snapshot of the allocations of the tokens in the system; more precisely, a *configuration* $C = [r_0, r_1, ..., r_{n-1}, a_0, ..., a_{k-1}]$ is a $n + k$-tuple where $0 \leq r_i \leq p$ indicates the number of tokens placed in R_i, and $0 \leq a_j \leq p$ indicates the number of tokens held by A_j; by definition, $\sum r_i + \sum a_j = p$. As indicated before, the initial configuration is $C_0 = [p, 0, ..., 0, 0, ..., 0]$. We shall denote by $\mathcal{C}$ the set of all configurations. Given a configuration $C = [r_0, r_1, ..., r_{n-1}, a_0, ..., a_{k-1}]$, let $\hat{C} = [r_0, r_1, ..., r_{n-1}]$ be the associated n-tuple describing the *rooms configuration*.

Agents do not see other agents nor the tokens those agents hold, and they do not have any means of direct communication other that the tokens.

The behaviour of an agent is determined by a protocol, the same for all agents. At any time, an agent is either *active* or *inactive*. When *active* an agent performs a finite sequence of operations according to the protocol; operations involve moves, token manipulations, and internal computations and state modifications. Having

performed at least one token manipulation (i.e., a change of configuration), it then becomes *inactive*.

Activation of the agents is determined by an adversarial *scheduler* which however activates each agent infinitely often. Among all possible schedulers, an *alternating scheduler* is one that alternates the activation of the agents, having only one agent active at any time.

Let W be a (possibly infinite but) countable set of *messages*. We consider several types of communication problem, some between a pair of agents, others among a group of agents.

Consider first the communication problem between two agents:

- The *unidirectional communication* (or basic communication) requires one agent, called the sender to communicate to the other agent, called the *receiver*, any $w \in W$ in *finite* time with a *bounded* number of actions (moves and token operations) in each activation.

- The *bidirectional communication* (or basic exchange) requires one agent, called the *sender* to communicate to the other agent, called the *receiver*, any $w \in W$ in *finite* time with a *bounded* number of actions (moves and token operations) in each activation; once this is done the two agents exchange roles. This process can be generalized to a *sequence of alternating exchanges* in which the two agents keep on switching roles until all communication between them is exhausted.

Among the communication problems involving any number of agents:

- The *broadcast* process requires one agent, called the *sender* to communicate to the other agent, called the *receiver*, an input message $w \in W$ in *finite* time with a *bounded* number of actions (moves and token operations) in each activation.

- The *convergecast* process requires one agent, called the *sink* to receive from each of the other agents, called *sources*, an input message $w \in W$ (possibly different for each source) in *finite* time with a *bounded* number of actions (moves and token operations) in each activation.

- The *gossip* process requires every agent to communicate to every other agent its input message $w \in W$ in *finite* time with a *bounded* number of actions (moves and token operations) in each activation.

7.2 Communication between Two Agents

In this section, we investigate *communication* between $k = 2$ agents, A and B, which have access to $p \geq 1$ tokens distributed over $n \geq 1$ rooms. In particular we examine both the unidirectional and the bidirectional (reversible) communication problems. The results for bidirectional communication are summarized in Table 7.1, where $q = \binom{p+n}{n} - \binom{p+n-1}{n-1} - 2$; when communication is possible, the entry indicates the base in which the communicated message is encoded.

Tokens \ Rooms	1	2	$n > 2$
1	*impossible*		
2	*impossible*	2	q
$p > 2$	$p - 2$	q	q

Table 7.1: Summary of results for bidirectional communication between two agents. When communication is possible, the entry indicates the base in which the communicated message is encoded; $q = \binom{p+n-1}{n} - 2$

7.2.1 Basics

Let W be a (possibly infinite but) countable set of *messages*; without loss of generality, let $W \subseteq \mathcal{Z}^+$. The *unidirectional communication problem* requires to devise a protocol which enables agent A, called the *sender* to communicate to the other agent B, called the *receiver*, any $w \in W$ in *finite* time, and with a *bounded* number of operations (moves and token operations) in each activation. This problem can be generalized to a finite sequence of unidirectional communications. *Bidirectional communication* is the problem of performing a unidirectional communication from A to B followed by a unidirectional communication from B to A; this problem can be generalized to a finite sequence of alternating unidirectional communications.

Any protocol P solving the basic communication problem on W is fully described in terms of two transition functions, a *sender transition* function $\tau_A : W \times S_A \times \mathcal{C} \mapsto S_A \times \mathcal{C}$ and a *receiver transition* function $\tau_B : S_B \times \mathcal{C} \mapsto S_B \times \mathcal{C}$, where S_A and S_A denote the (possibly infinite) set of internal states of A and of B, respectively.

The mapping $\tau_A(w, s, C) \mapsto (s', C')$, where $C = [r_0, r_1, ..., r_{n-1}, a, b]$ and $C' = [r'_0, r'_1, ..., r'_{n-1}, a', b]$, means that if A is activated in state s and the configuration is C, to (continue to) communicate w, A will (1) move and manipulate the tokens it holds and those placed in the rooms creating configuration C', (2) enter state s' and (3) become inactive; since A has no access to the tokens held by B, clearly the transaction must satisfy the property $\sum r_i + a = \sum r'_i + a'$ (token conservation). Similarly for τ_B. Let ι_A, and ι_B be the initial states of A and B, respectively.

Since any correct solution protocol must terminate, without loss of generality we can assume that there are two special terminal states: $\phi_A \in S_A$ and $\phi_B \in S_B$; if in the terminal state, no activity will be performed by the agent. That is $\tau_A(w, \phi_A, C) \mapsto (\phi_A, C)$ and $\tau_A(w, \phi_B, C) \mapsto (\phi_B, C)$

Since, by definition, each activity must include a change of configuration (unless in a terminal state), then for each transition $\tau_A(w, s, C) \mapsto (s', C')$, if $s \neq \phi_A$, then $C \neq C'$ and $s \neq s'$; similarly, for $\tau_B(s, C) \mapsto (s', C')$.

Note that, if an agent at the end of action holds all the tokens, then the other agent cannot act; hence, unless it is in a terminal state, no correct protocol can allow such a transition. More precisely,

Lemma 7.2.1. *Let $C^a = [0, 0, 0, ..., p, 0]$ and $C^b = [0, 0, 0, ..., 0, p]$. For any transition $\tau_A(w, s, C) \mapsto (s', C')$, if $s \neq \phi_A$, then $C' \neq C^a$; and for any transition $\tau_B(s, C) \mapsto (s', C')$, if $s \neq \phi_B$, then $C' \neq C^b$.*

All our protocols make the agents act in turn, like in an alternating scheduler, ensuring that no ambiguity arises when an agent (inappropriately activated) observes an intermediate configuration; furthermore, they all use the observation of Lemma 7.2.1.

7.2.2 General Communication Protocol: $n \geq 2, p \geq 2$

In the following, we devise a general solution for the problem of bidirectional communication; unidirectional communication can be easily achieved by terminating after the first communication is terminated.

Let $\mathcal{R} = \{R_0, R_1, ..., R_{n-1}\}$ be the $n \geq 2$ rooms available for communication from the sender A in state *sending* to the receiver B in state *receiving*. Initially all $p \geq 2$ tokens are in room R_0 and both agents have no tokens in hand, a configuration we denote as C_b.

The message $w \in W$ to be transmitted from the sender to the receiver is coded into a sequence of digits: $\mathcal{S}(w) = b_0 b_1 \cdots b_m$, where every digit b_i belongs to an alphabet $\mathcal{Z}_q = \{0, \ldots, q-1\}$ of size q, for some q. In this case we talk about q-ary communication.

The idea of the protocol is to code each element of $\mathcal{Z}_q$ into a configuration of tokens within the n rooms, and to have the sender communicate each digit of the string $\mathcal{S}(w)$ by placing the tokens into the rooms, according to the predefined code.

The communication of each digit is acknowledged by the receiver before proceeding to the communication of the next digit. Clearly termination has to be communicated when the entire string has been communicated. We consider two types of termination: *final* and *with-switch*. In the first case, the string is finished and communication is concluded; in the second, the string is finished and a reply is required. The termination with-switch is used to switch the direction of communication so to perform bidirectional communication.

There are several different ways to indicate termination, each using different resources. We consider two: explicit termination, and implicit termination.

String Communication with Explicit Termination.

Explicit termination is achieved by keeping two dedicated room configurations, $\hat{C}_f$ indicating *final termination* and $\hat{C}_s$ indicating *termination-with-switch*, which are communicated by the sender instead of a digit, when appropriate.

Let $code(b_i)$ indicate a placement of tokens into the n rooms corresponding to digit b_i. The definition of the code and the size q of the alphabet depends on the number

of rooms and of the tokens available; that is, it depends on the number of different configurations that can be created distributing the p tokens into the n rooms, and the details will be discussed later. An important property of the code is that the code of any digit includes *the presence of at least one token in room* R_0.

When the sender wishes to start the communication of any digit of the string (and in particular, of the first one), it picks up all tokens from R_0. To communicate digit b_i, the sender places tokens into rooms according to $code(b_i)$, dealing with room R_0 last, and then changes state to *digit-sent*. Note that the code might use any number of tokens to be distributed among the rooms (at least one in R_0). To communicate final termination, and switch, the sender uses two dedicated configurations of tokens and becomes, respectively *termination-sent*, and *switch-sent*.

When the receiver in state *receiving* sees that room R_0 is not empty and contains less than p tokens, it moves through the rooms to "read" the transmitted code corresponding to the transmitted bit, decoding the digit. While doing so, it picks up all the tokens starting from room R_{n-1} downwards.

- Consider first the case when the transmission does not indicate termination. In this case the receiver becomes *digit-received*.

  If the receiver, in state *digit-received*, has p tokens (note that the sender is empty handed in this case), it puts them all down in R_0 so that the sender can start the transmission of the next digit and becomes *receiving*. The sender in state *digit-sent* seeing all tokens in R_0 becomes *sending* again and can restart.

  If instead the receiver, in state *digit-received*, has less than p tokens (i.e., the sender is not empty handed), the receiver does nothing; when the sender in state *digit-sent* sees that R_0 is empty, it puts down its own tokens in R_0 becoming *wait-to-restart*. When the receiver, in state *digit-received*, sees that R_0 is not empty, it drops all its token so that there will be p tokens again in R_0, it becomes *receiving* and the sender can start again. When the sender in state *waiting-to-restart* sees that room R_0 contains again all the available tokens, it can continue with the communication of the next digit becoming *sending* again.

- Consider now the case when the transmission indicates final termination: the

receiver drops all tokens, it becomes *done*, and communication terminates.

- Consider finally the case when the transmission indicates a switch. In this case the receiver acknowledges it by dropping all the collected tokens and becoming the new sender entering state *sending* (a switch of roles is occurring);

  the previous sender, in state *sent-switch*, once it sees all tokens except its own are in R_0, drops all its tokens in room R_0 becoming the new receiver and entering state *receiving*.

Let $\mathcal{P}$ be the set of all possible distributions of up to p tokens in the n rooms. The *code* used by the algorithm with explicit termination is any injective mapping $\kappa_{ex} : \mathcal{Z}_q \cup \{b, s, f\} \to \mathcal{P}$ specifying which allocation of tokens to the rooms is assigned to each digit, as well as specifying the dedicated configurations $\hat{C}_b, \hat{C}_s, \hat{C}_f$, subject to the constraints shown in Table 7.2. Note that room R_0 containing all tokens is the dedicated configuration $\hat{C}_b$, indicating the beginning of a new digit to communicate. Moreover, in any code, room R_0 contains at least a token.

Config.	*Constraint*	*Meaning*
$\kappa_{ex}(b) = \hat{C}_b$	$r_0 = p$	Beginning
$\kappa_{ex}(s) = \hat{C}_s$	$p > r_0 > 1$, $\sum_{i=0}^{n-1} r_i < p$	Switch
$\kappa_{ex}(f) = \hat{C}_f$	$p > r_0 > 1$	Full Termination
$\kappa_{ex}(b_i), b_i \in \mathcal{Z}_q$	$p > r_0 > 1$	digit

Table 7.2: COMM2(p,n) with Explicit Termination: Constraints on the code.

The size q_{ex} of the largest alphabet which satisfies the constraints imposed on the code with explicit termination, is determined as follows.

Lemma 7.2.2. $q_{ex} = \binom{p+n-1}{n} - 3$

Proof. The number of different configurations obtained by distributing up to p tokens between the n rooms is $[p, n] = \binom{p+n}{p}$. The constraints that $r_0 > 1$ implies that $[p, n-1]$ cannot be used by the code; similarly, three distinct configurations $(\hat{C}_b, \hat{C}_s, \hat{C}_f)$ cannot be used by the code of the digits. Hence, $q_{ex} = [p, n] - [p, n-1] - 3 = \binom{p+n}{n} - \binom{p+n-1}{n-1} - 3 = \binom{p+n-1}{n} - 3$. □

String Communication with Implicit Termination. Another way to achieve termination is to keep only one dedicated configuration $\hat{C}_t$ indicating that the string is finished.

Once this is communicated and acknowledged, the distinction on whether it is a switch or a final termination will be communicated by an appropriate subsequent digit as follows.

If the sender wishes to communicate a switch, its picks up all its tokens and it places one in R_1. Upon detecting the change in R_1, the receiver puts its only token in R_0, waits until the sender puts down in R_0 the remaining $p-1$ tokens, and the switch occurs.

If the sender wants to communicate final termination, it puts all the tokens in R_1. Upon detecting the presence of p tokens in R_1, the receiver picks all of them up, puts them into R_0 and the process is completed.

The *code* used by the algorithm with implicit termination is any injective mapping $\kappa_{im} : \mathcal{Z}_q \cup \{b, t\} \rightarrow \mathcal{P}$ specifying which allocation of tokens to the rooms is assigned to each digit, as well as specifying the dedicated configurations $\hat{C}_b, and \hat{C}_t$, subject to the constraints shown in Table 7.3. Note that room R_0 containing all tokens is the dedicated configuration $\hat{C}_b$, indicating the beginning of a new digit to communicate. Moreover, in any code, room R_0 contains at least a token.

Config.	*Constraint*	*Meaning*
$\kappa_{ex}(b) = \hat{C}_b$	$r_0 = p$	Beginning
$\kappa_{ex}(t) = \hat{C}_t$	$p > r_0 > 1$	Termination
$\kappa_{ex}(b_i), b_i \in \mathcal{Z}_q$	$p > r_0 > 1$	digit

Table 7.3: COMM2(p,n) with Implicit Termination: Constraints on the code.

The size q_{im} of the largest alphabet which satisfies the constraints imposed on the code with implicit termination, is determined as follows.

Lemma 7.2.3. $q_{im} = \binom{p+n-1}{n} - 2$

Proof. The number of different configurations obtained by distributing up to p tokens between the n rooms is $[p, n] = \binom{p+n}{p}$. The constraints that $r_0 > 1$ implies that

$[p, n-1]$ cannot be used by the code; similarly, two distinct configurations $(\hat{C}_b, \hat{C}_t)$ cannot be used by the code of the digits. Hence, $q_{im} = [p, n] - [p, n-1] - 2 = \binom{p+n}{n} - \binom{p+n-1}{n-1} - 2 = \binom{p+n-1}{n} - 2$. □

Algorithm 7.1 and 7.2 show the sequence of interactions in the case of explicit and implicit termination respectively where: $code(d)$ indicates the distribution of tokens corresponding to digit d described previously, $\hat{C}_f$ and $\hat{C}_s$ are the special codes indicating termination and switch for explicit termination and $\hat{C}_t$ is the special code indicating termination for implicit termination. Recall that r_i indicates the number of tokens in room R_i.

Theorem 7.2.1. *Algorithm 7.1 solves the bidirectional communication problem for $n \geq 2, p \geq 2$ using an alphabet of size q_{ex} when $n + p \geq 5$.*

Proof. To prove the theorem, it suffices to show that the transmission of each piece of information (whether digit or termination) is successfully achieved from the *starting configuration* (i.e., with all p tokens in room R_0) with the current sender in state *sending* and the current receiver in state *receiving*, and ending in the starting configuration, with either the (possibly new) sender in state *sending* and the (possibly new) receiver in state *receiving* (in the case of digits or switches), or with both agents in state *done* (in case of full termination).

By construction, the sender in state *sending*, seeing a starting configuration, picks up all tokens from R_0, distributes (a subset of) them according to $code(d)$ (where d is the piece of information to be sent), and becomes *digit/termination/switch-sent*. Since, by construction, R_0 is the last room where the tokens are going to be placed, and since, again by construction, in any code R_0 contains at least a token but never all p of them, for an agent in state *receiving*, the condition $0 < r_0 < p$ tokens in R_0 uniquely signifies that the distribution of the tokens in the rooms has been completed. Upon determining the occurring of such condition, the receiver collects all the tokens from all the rooms starting from room R_{n-1} upwards. When it is in room R_0, the receiver can decode the information recognizing the transmission of a digit or of a termination message; it then becomes *digit-received*, or *termination-received*, or *switch-received* accordingly.

Algorithm 7.1 Communication Module: COMM2(p,n) with explicit termination

Initially: $r_0 = p$	
Finally: $r_0 = p$	
Sender:	
in state *sending*: to communicate	digit d or termination
$r_0 = p$	pick up all tokens from R_0
	distribute tokens into rooms
	according to $code(d)$, $\hat{C}_f$ or $\hat{C}_s$
	with room R_0 the last to be filled.
	become *digit-sent*, *termination-sent* or *switch-sent*
in state *digit-sent*: $r_0 = 0$	drop all tokens (if any in hand)
	become *wait-to-restart*
in state *wait-to-restart*: $r_0 = p$	become *sending*
in state *switch-sent*: $r_0 \neq 0$	drop tokens,
	become *receiving*
in state *termination-sent*: $r_0 \neq 0$	drop tokens,
	become *done*
Receiver:	
in state *receiving*:	
$0 < r_0 < p$	scan all rooms to decode digit or detect $\hat{C}_f$ or $\hat{C}_s$
	with room R_0 the last to be emptied
	become *digit-received*,
	termination-received, *switch-received*
	accordingly
in state *digit-received*:	
p tokens in hand (and $r_0 = 0$)	drop all tokens, become *receiving*
$r_0 \neq 0$	drop all tokens, become *receiving*
in state *termination-received*	drop tokens, become *done*
in state *switch-received*	drop tokens, become *sending*

Algorithm 7.2 Communication Module: COMM2(p,n) with implicit termination

Initially: $r_0 = p$	
Finally: $r_0 = p$	
Sender:	
in state *sending*: to communicate	digit d or termination
$r_0 = p$	pick up all tokens from R_0
	distribute tokens into rooms
	according to $code(d)$, or $\hat{C}_t$
	with room R_0 the last to be filled.
	become *digit-sent*, *termination-sent*
in state *digit-sent*: $r_0 = 0$	drop all tokens (if any in hand)
	become *wait-to-restart*
in state *wait-to-restart*: $r_0 = p$	become *sending*
in state *switch-sent*: $r_0 = 1$	drop tokens, become *receiving*
in state *termination-sent*: $r_0 = p$	
for final-termination	pick all tokens from R_1 and drop all on R_1,
	become *done*
for switch	pick all tokens from R_1 and drop one on R_1,
	become *switch-sent*
Receiver:	
in state *receiving*:	
$0 < r_0 < p$	scan all rooms to decode digit or detect $\hat{C}_t$
	with room R_0 the last to be emptied
	become *digit-received*,
	termination-received
	accordingly
in state *digit-received*:	
p tokens in hand (and $r_0 = 0$)	drop all tokens, become *receiving*
$r_0 \neq 0$	drop all tokens, become *receiving*
in state *termination-received*	
$r_1 = p$	pick all tokens and drop on R_0, become *done*
$r_1 = 1$	pick-up token from R_1, become *switch-received*
in state *switch-received*	drop token on R_0, become *sending*

What happens next depends by the type of information communicated and on the number of tokens held by the sender.

- If the information is a digit and the sender has some tokens, the sender unambiguously interprets the absence of tokens in R_0 as the signal that the information has been read, and drops its tokens becoming *wait-to-restart*. The receiver unambiguously determines the presence of the tokens in R_0, drops its token in R_0 (creating the starting configuration again) and becomes *receiving*. The sender, upon detecting all p tokens in R_0 becomes *sending* again, and the theorem holds.

- If the information is a digit and the sender has no tokens (i.e., the receiver has all p tokens), the receiver drops all its token in R_0 (creating the starting configuration again) and becomes *receiving*. The sender, upon detecting all p tokens in R_0, becomes *sending* again, and the theorem holds.

- If the information is a switch, the sender has some tokens in hand by definition. The receiver, aware that the information is a request for a switch, drops its token in R_0 and becomes the new sender in state *sending*. The old sender unambiguously interprets the presence of tokens in R_0 as indicating that the other agent has already performed the switch; it then drops its token in R_0 (creating the starting configuration again) and becomes *receiving*, and the theorem holds.

- If the information is full termination, the receiver drops its token and terminates becoming *done*; the sender, upon seeing tokens in R_0, drops its tokens (if any) creating the starting configuration again, and terminates becoming *done*; and the theorem holds.

Observe that both agents move from room to room only when the number of tokens on R_0 changes, and this occurs only a constant number of times during the transmission of an item of information (digit or termination). Hence, the assertion of the theorem is established. □

Theorem 7.2.2. *Algorithms 7.2 solves the bidirectional communication problem for* $n \geq 2, p \geq 2$ *using an alphabet of size* q_{im}

The proof follows the same lines as that of Theorem 7.2.1.

7.2.3 Special Protocol: $n = 1, p > 2$

The case of a single room is one that is not taken into account in the general Algorithm of the previous section, because when there is a single room available, the general technique cannot be employed.

As in the previous section, the idea is to communicate the string one digit at a time. At each step during the string communication, there are two types of transmissions that can be performed: digit communication or termination (switch or final termination). The code used is to encode digit i as $i+1$ tokens in the room; the configuration with p tokens in the room is reserved for the beginning of communication of a digit; hence the size of the alphabet used is $q = p - 2$.

We denote by (x, y, z) the situation when agent A has x tokens, agent B has z tokens, and the room contains y tokens; notice that this situation corresponds to configuration $[r_0 = y, a = x, b = z]$. The initial situation corresponds to: $(0, p, 0)$ where agents A is the sender and agent B is the receiver, in states *sending* and *receiving* respectively.
To communicate a *digit* from $(0, p, 0)$:

- The sender picks up the tokens and drops $i + 1$ tokens to communicate digit i becoming *i-sent*: $(p - i - 1, i + 1, 0)$

- The receiver seeing $i+1$ tokens in the room with no tokens in hand understands that bit i has been communicated, changes state to *i-received*, and picks up a token to acknowledge reception: $(p - i - 1, i, 1)$. Call this configuration *decisional*.

- The sender, in state *i-sent* wanting to continue with another digit, and seeing i tokens, drops all its tokens and becomes *sending*: $(0, p - 1, 1)$

- The receiver seeing $p - 1$ tokens in state *i-received*, understands that another bit has to be communicated and drops its token becoming *receiving* again and creating an initial configuration: $(0, p, 0)$

Termination (switch or final termination) is communicated from states *i-sent* and *i-received* in a *decisional* configuration $(p-i-1, i, 1)$ as follows:

- The sender, in state *i-sent* wanting to terminate, seeing i tokens picks all tokens, if there are none it drops 1 creating either $(p-1, 0, 1)$ or $(p-2, 1, 1)$.
- The receiver in state *i-received* seeing a number of tokens different from i and from $p-1$, understands that termination is being communicated and it drops its token becoming *wait-for-termination*: $(x, y, 0)$ with $y = 2$ if $i = 0$, and $y = 1$ otherwise.
- To perform a *switch*
  - the sender in state *terminating* noticing the added token, drops all its tokens and becomes the new receiver entering state *receiving*: $(0, p, 0)$.
  - The receiver in state *wait-for-termination* seeing p tokens, understands that a switch has been communicated, it changes role and becomes the new sender entering state *sending* and creating the initial configuration.
- To perform a *final termination*
  - the sender in state *terminating* seeing the added token, drops all its tokens but 1 : $(1, p-1, 0)$.
  - The receiver in state *wait-for-termination* seeing $p-1$ tokens with no tokens in hand, understands that a final termination has been communicated, it becomes *terminating* and it picks up a token: $(1, p-2, 1)$.
  - The sender in state *terminating* seeing $p-2$ tokens, drops its token and becomes *done*: $(0, p-1, 1)$.
  - The receiver in state *terminating* seeing $p-1$ tokens with 1 in hand, becomes *done* and drops its token: $(0, p, 0)$.

Theorem 7.2.3. *Algorithm 7.3 solves the bidirectional communication problem for* $n = 1, p > 2$ *with* $q = p-2$ *encoding.*

The proof follows by construction, in a straightforward manner.

Algorithm 7.3 Communication Module: COMM2(p,1) $p > 2$

Initially: $r_0 = p$
Finally: $r_0 = p$

Sender:	
in state *sending*: to communicate	digit $d = i$
$r_0 = p$	pick up all tokens from R_0
	drop $i + 1$ tokens on R_0
	become *i-sent*
in state *i-sent*: $r_0 = p$, to continue	become *sending*
in state *i-sent*: to terminate	
$r_0 \neq 0$,	pick up r_0 tokens
	become *terminating*
in state *i-sent*: to terminate	
$r_0 = 0$,	drop one token
	become *terminating*
in state *terminating*: to switch	
r_s on hand, $r_0 + r_s = p$,	drop all tokens
	become *receiving*
in state *terminating*: for final termination	
r_s on hand, $r_0 + r_s = p$	drop all but one
in state *terminating*: $r_0 = p - 2$	drop token, become *done*
Receiver:	
in state *receiving*:	
$0 < r_0 = i < p$ and empty handed	pick up one token
	become *i-received*
in state *i-received*:	
$r_0 = p - 1$	drop the token, become *receiving*
in state *i-received*:	
$r_0 \neq p - 1, i$	drop the token, become *wait-for-termination*
in state *wait-for-termination*: $r_0 = p$	become *receiving*
in state *wait-for-termination* : $r_0 = p - 1$	pick up a token, become *terminating*
in state *terminating* : $r_0 = p - 1$	drop tokens, become *done*

7.2.4 Improved Communication Protocol: $n = p = 2$

Algorithm 7.2 described in Section 7.2.2, when applied to the case of 2 rooms and 2 tokens, achieves only unary communication ($q = 1$). We can however design a specific algorithm that communicates in binary ($q = 2$).

Initially two tokens are in room R_0 and both agents have no tokens in hand. The string to be communicated is coded in binary alphabet. As in the previous sections, the idea is to communicate the string one bit at a time. At each step during the string communication, there are two types of transmissions that can be performed: digit communication or termination. The idea is for the sender to transmit bit 0 by picking up a single token from R_0; bit 1 by picking up both tokens and placing them in R_1.

Let (x, y, z, w) indicate a situation where agent A has x tokens, room R_0 contains y token, room R_1 contains z, and agent B has w tokens. The initial configuration corresponds to $(0, 2, 0, 0)$ with agents A (sender) and B (receiver) in states *sending* and *receiving* respectively.
To communicate a *digit* from $(0, 2, 0, 0)$:

- To communicate 0, the sender in state *sending* picks up a single token from R_0 becoming *0-sent*: $(1, 1, 0, 0)$; The receiver in state *receiving* seeing 1 token in R_0, understands that bit 0 has been communicated, changes state to *0-received*, and picks up the token to acknowledge reception: $(1, 0, 0, 1)$. The sender in state *0-sent* on seeing R_0 empty, drops its token in R_0 becoming *termination-sending*: $(0, 1, 0, 1)$. The receiver in state *0-received*, seeing a single token in R_0, drops its token and becomes *termination-receiving*: $(0, 2, 0, 0)$.

- To communicate 1, the sender in state *sending* picks up both tokens from R_0 and places on R_1 to communicate digit 1 becoming *1-sent*: $(0, 0, 2, 0)$; The receiver in state *receiving* seeing 0 tokens in R_0 changes its state to *1-received*. The receiver in state *1-received* goes to room R_1, picks up both tokens from R_1, and places both in R_0 and becoming *termination-receiving*: $(0, 2, 0, 0)$. The sender in state *1-sent* seeing both tokens in R_0 becomes *termination-sending*.

- The sender, in state *termination-sending* wanting to continue with another digit,

picks up a token from R_0 and becomes *continue-to-send*: $(1, 1, 0, 0)$. The receiver in state *termination-receiving*, on seeing 1 token in R_0, picks it up and becomes *continue-to-receive*. The sender in state *continue-to-send*, seeing R_0 empty, drops its token and becomes *sending*. The receiver in state *continue-to-receive*, seeing a single token in R_0, drops its token on R_0 and become *receiving*.

- The sender in state *termination-sending* wanting to terminate, picks up both tokens from R_0 and places them in R_1 and becomes *terminating*: $(0, 0, 2, 0)$. The receiver in state *termination-receiving* seeing 0 tokens in R_0, goes to R_1, picks up both tokens, and it places them in R_0 for final termination, becoming *Done*. The sender in state *terminating*, seeing two tokens in R_0, becomes *Done*. On the other hand, to switch roles, the receiver in state *termination receiving* places a single token in R_0 and becomes *switch-sent*. The sender in state *terminating*, seeing one token in R_0, picks it up and becomes *switch-received*. The receiver in state *switch-sent*, seeing no token in R_0, drops its token on R_0 and becomes *sending*. The sender in state *switch-received* drops its token on R_0 and become *receiving*.

The protocol for sender and receiver are presented in Algorithms 7.4 and 7.5.

Theorem 7.2.4. *The Algorithm composed of modules 7.4 and 7.5 solves the bidirectional communication problem for $n = p = 2$ with a binary encoding.*

Proof. We show that the transmission of each piece of information (whether a digit or termination) is successfully achieved starting and ending from the *starting configuration*, with the current sender in state *sending* and the *current receiver* in state *receiving* (for digits or switches) and with them in state *done* for full termination.

Initially the system is in the *starting configuration* with the two tokens on R_0. By construction, the sender in state *sending* seeing a starting configuration proceeds depending on the bit to be transmitted. By construction, to transmit bit 0 it picks up a single token and becomes *0-sent*. On the contrary, to transmit bit 1 it picks up both tokens, places both on R_1 and becomes *1-sent*. The *receiver* seeing $r_0 \neq 2$ tokens on R_0 understands that a bit has been communicated. The *receiver* in state *receiving* picks up a token and becomes *0-received* if $r_0 = 1$ and becomes *1-received*

Algorithm 7.4 Communication Module(for sender): COMM(2,2)

Initially: $r_0 = 2$	
Finally: $r_0 = 2$	
Sender:	
in state *sending*: to communicate	bit $d = 0$
$r_0 = 2$	pick up a single token
	become *0-sent*
to communicate	bit $d = 0$
$r_0 = 2$	picks up both tokens
	places both on R_1
	become *1-sent*
in state *0-sent*: $r_0 = 0$,	
	drop one on R_0
	become *termination-sending*
in state *1-sent*: $r_0 = 2$	become *termination-sending*
in state *termination-sending*: $r_0 = 2$	
to continue with next bit	pick a token
	become *continue-to-send*
to terminate	pick both tokens
	place on R_1
	become *terminating*
in state *continue-to-send*: $r_0 = 0$	
	drop one on R_0
	become *sending*
in state *terminating*:	
$r_0 = 1$	pick one
	become *switch-received*
$r_0 = 2$	become *done*
in state *switch-received* : $r_0 = 1$	drop one token on R_0
	become *receiving*

Algorithm 7.5 Communication Module(for receiver): COMM(2,2)

Initially: $r_0 = 2$
Finally: $r_0 = 2$

Receiver:	
in state *receiving*:	
$r_0 = 1$	pick up one token
	become *0-received*
$r_0 = 0$	become *1-received*
in state *0-received*: $r_0 = 1$,	
	drop one on R_0
	become *termination-receiving*
in state *1-sent*: $r_0 = 0$	go to R_1
$r_1 = 2$	pick up both tokens
	place on R_0
	become *termination-receiving*
in state *termination-receiving*:	
$r_0 = 1$	pick a token
	become *continue-to-receive*
$r_0 = 0$	go to R_1
to terminate	move both to R_0
	become *Done*
to switch	pick both and place one on R_0
	become *switch-sent*
in state *continue-to-receive*: $r_0 = 1$	drop a token on R_0
	become *receiving*
in state *switch-sent*: $r_0 = 0$	drop one on R_0
	become *sending*

if $r_0 = 0$. In the former case the *sender*, seeing that $r_0 = 0$, drops a token on R_0 and becomes *termination-sending*. The *receiver* in state *0-received* seeing a token on R_0, by construction, drops a token on R_0 and becomes *termination-receiving*. In the latter case, the *receiver* in state *1-received* moves to R_1, picks up both tokens, places them on R_0 and becomes *termination-sending*. The *sender* in state *1-sent* seeing two tokens on R_0 becomes *termination-sending*. Now, regardless of the bit transmitted, the *sender* is in state *termination-sending*, the *receiver* is in state *termination-receiving* and both tokens are on R_0.

By the rules of the algorithm, the next action of the *sender* in state *termination-sending* depends on whether it is going to continue to send bits or is terminating the string of bits. By construction, to continue transmit bits it picks up a single token and becomes *continue-to-send*. On the contrary, to terminate it picks up both, places them on R_1 and becomes *terminating*. The *receiver* seeing $r_0 \neq 2$ tokens on R_0 acts depending on r_0 by the rules of the algorithm, The *receiver* in state *termination-receiving* on seeing a single token on R_0 picks up token and becomes *continue-to-receive*. The *sender* seeing that $r_0 = 0$ drops a token on R_0 and becomes *sending*. The *receiver* in state *continue-to-receive* seeing a token on R_0, by construction, drops a token on R_0 and becomes *receiving*. The *starting configuration* is reached and the sender can continue with the transmission of next bit. Hence the theorem holds.

In the case that the string of bits have terminated the *receiver* in state *termination-receiving* on seeing no tokens on R_0, moves to R_1 and picks up both tokens. Now the actions of the *receiver* in state *termination-receiving* depends on whether a reply i.e. switch is required or not. If it is required, by construction, it places a token on R_0 and becomes *switch-sent*. Otherwise, it places both tokens on R_0 and becomes *Done*. Now the action of the *sender* in state *terminating* depends on r_0. In the case that the switch is not required *sender* seeing two tokens on R_0 becomes *Done*. Hence the theorem holds. In the case that switch is required *sender* seeing a token on R_0 picks it up and becomes *switch-received*. Now the *receiver* in state *switch-sent* seeing no token on R_0 drops it token on R_0 and becomes *sending*. The *sender* in the state *switch-received* seeing a token on R_0 drops a token and becomes *receiving*. The *starting configuration* is reached and the new sender can continue with the transmission of

next bit. Hence the theorem holds.

Observe that both agents move from room to room only when the number of tokens on R_0 and/or R_1 changes. But the number of the changes in the number of tokens occurring in R_0 and R_1 during the transmission of one piece of information (digit or termination) is constant. Hence, the assertion of the theorem is established. □

7.2.5 Impossibility

The only cases still unsolved by our strategies are (1) $n = 1$ and $p = 2$, and (2) $p = 1$.

We now prove that in both cases bidirectional communication is *impossible*.

Theorem 7.2.5. *There is no solution protocol for the bidirectional communication problem when $n = 1$ and $p = 2$, even if the scheduler is an alternating one.*

Proof. By contradiction, let P be a deterministic solution protocol for the bidirectional communication process when $n = 1$, and and $p = 2$. Consider an alternating scheduler, i.e., a scheduler that alternates the activation of the agents, having only one agent active at any time.

Let us focus now on the execution of P. Both tokens are initially in the room. In the first turn, by Lemma 7.2.1, the sender A cannot pick up both tokens; hence A must necessarily pick up one token to perform an action, thus changing the configuration from $[2, 0, 0]$ to $[1, 1, 0]$. The only possible action of B is to pick up the remaining token (so to change the number of tokens in the room), thus creating configuration $[0, 1, 1]$. The only possible action of A at this point is to drop its token generating configuration $[1, 0, 1]$. When activated, by Lemma 7.2.1, B cannot pick up the token; hence, B must necessarily drop its token creating the configuration $[2, 0, 0]$ in which both tokens are in the room again. This means that, until (and except) termination, the sequence of configurations is just a word in the language $L = ([2, 0, 0], [1, 1, 0], [0, 1, 1], [1, 0, 1])^+$. Hence, termination can be notified only by the "forbidden" configurations of a an active node picking and holding all the tokens; i.e., $[0, 0, 2]$ and $[0, 2, 0]$. Consider now the two executions of P when A wants to transmit w and w', respectively, with $w \neq w'$. Consider the words $l(w), l(w') \in L$ corresponding to those two executions, respectively, until B determines the message sent by A and starts its own transmission.

W.l.g. let $l(w) < l(w')$; that is $l(w)$ is a prefix of $l(w')$. Therefore the state of B at $l(w)$ is the same in both executions; however, it is supposed to enter different states in the two executions, contradicting the determinism of P.

□

It is easy to see that if only one token is available, regardless of the number of rooms, there exist no unidirectional communication protocol that terminates in finite time with a bounded number of moves by the agents.

Theorem 7.2.6. *There is no solution protocol for the unidirectional communication problem when $p = 1$, regardless of the number n of rooms.*

Proof. By contradiction, let P be a deterministic solution protocol for the unidirectional communication process when $p = 1$. Consider an alternating scheduler, i.e., a scheduler that alternates the activation of the agents, having only one agent active at any time.

The contradiction derives from the requirement that in each action an agents must perform a bounded number of operations. Since there is only one token, and since (by Lemma 7.2.1) an agent cannot end its action holding all tokens (unless in a terminal state), to change the configuration (required while active), the only action while not in a terminal state consists of moving the token from the current room into a distinct one.

Consider now the two executions of P under an alternating scheduler when A wants to transmit w and w', respectively, with $w \neq w'$. Consider now the sequence of configurations $D(w) =< C_0, C_1, ...C_j >$ and $D(w') =< C_0, C'_1, ...C'_j >$ corresponding to those two executions, respectively, until termination. Let i be the smallest index such that $C_i \neq C'_i$. Thus the sequence $< C_0, C_1, ...C_{i-1} >$ is a prefix of both $D(w)$ and $D(w')$. Consider the execution of P up to this point. Notice that, by necessity, the next active agent is A, and that the state of B at this point is the same in both executions.

To change the configuration $C_{i-1} = [r_1, ..., r_n, 0, 0]$, A must pick up the token from the occupied room (say R_s) and move it to the appropriate room R_t to form $C_i = [r'_1, ..., r'_n, 0, 0])$, or to a different room R_v to form $C'_i = [r''_1, ..., r''_n, 0, 0]$ $(s \neq t \neq v)$. In

both cases, it has to form the intermediate configuration $[0, ..., 0, 1, 0]$ while moving the token to the appropriate room. Let the adversarial scheduler activate B when this occurs and delay the action of A to drop the token. Since the state of B at this time is the same in both executions, it will execute the same operations in both unless it discovers any difference. To determine which of the two configurations is going to be formed, B has to check both R_t and R_v, eventually stopping in one of them (since we are allowing a bounded number of moves) until it notices a change in that room. At that point the adversary let A drop the token in the room not occupied by B, which will thus never discover the change.

□

7.3 Multiagent Communication: Broadcast

In this section, we develop protocols for more than two agent. We focus on the problem of *Broadcast*, which is defined as follows: a designated agent, called the *broadcaster*, needs to communicate some information to all the other agents, the *listeners*.

Note that the communication between the broadcaster and each listener could be bidirectional communication requiring several rounds of alternating interactions between broadcaster and each listener.

Table 7.4 summarizes our results for the case of Broadcast, where the impossibilities follow from the ones obtained for communication between 2 agents described in Section 7.2.5.

Tokens \ Rooms	1	2	$n > 2$
1	*impossible*		
2	*impossible*	q	q
$3, 4, 5$	2	q	q
$p > 5$	$p - 3$	q	q

Table 7.4: Summary of results for broadcast among multiple agents. When communication is possible, the entry indicates the base in which the communicated message is encoded, and $q = \binom{p+n-1}{n} - 2$.

7.3.1 General Communication Protocol: $n, p \geq 2$

The protocol described in this section is based on Algorithms 7.1 and 7.2 for communication between two agents. The idea is to have one of the listeners "register" to receive the information; once a listener has registered, communication between the broadcaster and the listener can proceed similarly to the case of two agents. Once the transmission of the message is terminated, another listener register, until all listeners have received the information.

Let $R_0, \ldots R_{n-1}$ be the n rooms available for communication. Initially all p tokens are in room R_0 and the agents have no tokens in hand. The configuration with p tokens in R_0 is called *ready-configuration*. The registered listener is the first agent to pick up the all p tokens from R_0 and to place them in R_1 becoming *receiving*. The *broadcaster* seeing that $r_0 = 0$ moves to R_1 becoming *sending*. At this point the communication is between the broadcaster and the registered receiver, and the protocol is similar to Algorithms 7.1 (for explicit termination) and 7.2 (for implicit termination).

Since there are other agents in the system, sender and receiver must insure that the *ready-configuration* never occurs while the message is communicated, otherwise another agent might interfere with the communication by picking up the tokens to become the receiver. To avoid this situation, in the communication between the broadcaster and the registered receiver, room R_1 plays the same role that R_0 plays in Algorithms 7.1 and 7.2; thus, in the broadcast protocol, room R_1 will never be empty to communicate a digit.

Let $\mathcal{P}$ be the set of all possible distributions of up to p tokens in the n rooms. The *code* used by the algorithm with explicit (resp.implicit) termination is any injective mapping $\kappa_{ex} : \mathcal{Z}_q \cup \{b, s, f\} \to \mathcal{P}$ specifying which allocation of tokens to the rooms is assigned to each digit, as well as specifying the dedicated configurations $\hat{C}_b, \hat{C}_s, \hat{C}_f$ (resp, $\hat{C}_b, \hat{C}_t$), subject to the constraints shown in Table 7.5 and Table 7.6.

Algorithm 7.6 presents the protocol allowing an agent to become the registered receiver. Algorithms 7.7 and 7.8 show the sequence of interactions in the case of explicit termination and implicit termination where: $code(d)$ indicates the distribution

Config.	*Constraint*	*Meaning*
$\kappa_{ex}(b) = \hat{C}_b$	$r_1 = p$	Beginning Forbidden for other tasks
$\kappa_{ex}(s) = \hat{C}_s$	$p > r_1 > 1$, $\sum_{i=0}^{n-1} r_i < p$	Switch
$\kappa_{ex}(f) = \hat{C}_f$	$p > r_1 > 1$	Full Termination
$\kappa_{ex}(b_i), b_i \in \mathcal{Z}_q$	$p > r_0 > 1$	digit

Table 7.5: COMM(p,n) with Explicit Termination: Constraints on the code.

Config.	*Constraint*	*Meaning*
$\kappa_{ex}(b) = \hat{C}_b$	$r_0 = p$	Beginning Forbidden for other tasks
$\kappa_{ex}(t) = \hat{C}_t$	$p > r_1 > 1$	Termination
$\kappa_{ex}(b_i), b_i \in \mathcal{Z}_q$	$p > r_0 > 1$	digit

Table 7.6: COMM(p,n) with Implicit Termination: Constraints on the code.

of tokens corresponding to digit d , $\hat{C}_f$ and $\hat{C}_s$ are the special codes indicating termination and switch for explicit termination, t is the special code indicating termination for implicit termination and r_i is the number of tokens in room R_i.

Algorithm 7.6 Module for registering a listener

Initially: $r_0 = p$	
Finally: $r_1 = p$	
broadcaster : $r_0 = 0$	moves to R_1 become *sending*
listener : $r_0 = p$	picks all from R_0 places all on R_1 become *receiving*

Following the same reasoning as in the correctness proofs of Algorithms 7.1 and 7.2, we have:

Theorem 7.3.1. *Algorithms 7.7 and 7.8 enable the broadcaster to exchange arbitrary messages with each listener using a bounded number of moves when no other agent interfere.*

Theorem 7.3.2. *When $n, p \geq 2$ and $n+p \geq 5$, the broadcaster can exchange arbitrary*

Algorithm 7.7 Communication Module: COMM(p,n) with explicit termination

Initially: $r_1 = p$
Finally: $r_1 = p$

Sender:	
in state *sending*: to communicate	digit d or termination
$r_1 = p$	Pick up all tokens from R_0 Distribute tokens into rooms according to *code*(d), $\hat{C}_f$, or $\hat{C}_s$ with room R_1 the last to be filled. become *digit-sent*, *termination-sent* or *switch-sent*
in state *sent*: $r_1 = p$	become *sending*
in state *digit-sent*: $r_1 = 1$	drop all tokens (if any in hand) become *wait-to-restart*
in state *wait-to-restart*: $r_1 = p$	become *sending*
in state *switch-sent*: $r_1 \neq 0$	drop tokens, become *receiving*
in state *termination-sent*: $r_1 \neq 0$	drop tokens, become *done*
Receiver:	
in state *receiving*:	
$0 < r_1 < p$	Scan all rooms to decode digit or detect $\hat{C}_f$ or $\hat{C}_s$ with room R_1 the last to be emptied become *digit-received*, *termination*, *switch* accordingly
in state *digit-received*:	
p tokens in hand (and $r_1 = 0$)	drop all tokens, become *receiving*
$r_1 \neq 0$	drop all tokens, become *receiving*
in state *termination-received*	drop tokens, become *done*
in state *switch-received*	drop tokens, become *sending*

Algorithm 7.8 Communication Module: COMM(p,n) with implicit termination

Initially: $r_0 = p$	
Finally: $r_0 = p$	
Sender:	
in state *sending*: to communicate	digit d or termination
$r_1 = p$	pick up all tokens from R_1
	distribute tokens into rooms
	according to *code*(d), or $\hat{C}_t$
	with room R_1 the last to be filled.
	become *digit-sent*, *termination-sent*
in state *digit-sent*: $r_1 = 0$	drop all tokens (if any in hand)
	become *wait-to-restart*
in state *wait-to-restart*: $r_1 = p$	become *sending*
in state *switch-sent*: $r_1 = 1$	drop tokens on R_1, become *receiving*
in state *termination-sent*: $r_1 = p$	
for final-termination	pick all but one token from R_1 and drop on R_0,
	become *done*
for switch	pick all tokens from R_1 and drop one on R_0,
	become *switch-sent*
Receiver:	
in state *receiving*:	
$0 < r_0 < p$	scan all rooms to decode digit or detect $\hat{C}_t$
	with room R_0 the last to be emptied
	become *digit-received*,
	termination-received
	accordingly
in state *digit-received*:	
p tokens in hand (and $r_0 = 0$)	drop all tokens on R_1, become *receiving*
$r_0 \neq 0$	drop all tokens on R_1, become *receiving*
in state *termination-received*	
$r_1 = 1$	pick token form R_1 and drop on R_0, become *done*
$r_0 = 1, r_1 = 0$	pick-up token from R_0, become *switch-received*
in state *switch-received*	drop token on R_1, become *sending*

messages in q-ary encoding (where $q = [p, n] - [p, n-1] - 2$) with each listener, with a bounded number of moves.

Proof. Observe that there are $[p, n]$ ways of distributing the tokens among n rooms and the sender, but not all of them are used to code a digit. In fact, in $[p, n-1]$ configurations R_1 is empty and those configurations are not used to communicate digits, and the ready configurations is sin the set of configurations in which R_1 is empty. Finally, one configuration is dedicated for termination; So, we have: $q = [p, n] - [p, n-1] - 2$. Hence, the result follows. □

Theorem 7.3.3. *Using Algorithms 7.7, 7.8 and 7.6 a designated agent can broadcast to all other agents.*

Proof. By Algorithm 7.6 a receiver can register. Once a receiver is registered, Algorithms 7.7 and 7.8 can be used to exchange arbitrary message with the listener by Theorem 7.3.1 since no other agent will interfere as no time in execution of Algorithms 7.7 and 7.8, $p-1$ tokens are placed on R_0. □

7.3.2 Special Protocol $n = 1, p > 3$

The case of a single room is not taken into account in the general Algorithm of Section 7.3.1 because, similarly to the case of two agents, when there is only a single room available, the general technique cannot be employed.

Initially all p tokens are in room R_0 and both agents have no tokens in hand. The broadcaster must first transmit that there is communication to be sent, and a receiver has to register to receive the message.

The configuration with p tokens in R_0 is called *ready-configuration* and is dedicated to registrations. The first listener to pick up a token from the p available in R_0 becomes registered to receive the information. When the number of token in R_0 changes to $p-1$, the broadcaster picks up a token and becomes *sender*. When the number of tokens in R_0 is $p-2$, the registered listener drops its token and become the *receiver* in state *receiving*. Then the broadcaster drops two tokens in R_0.

At this point the communication is between two agents, and the protocol is similar to the one described in Section 7.2.3. In this case, however, sender and receiver must

insure that the *ready-configuration* never occurs while the message is communicated, otherwise another agent might interfere with the communication by picking up the tokens to become the receiver. The code used is to encode digit i as $i+1$ tokens in the room; the configuration with p tokens in the room is reserved for the beginning of communication of a digit, and the *ready-configuration* is dedicated for the listener registration; hence the size of the alphabet used is $q = p - 32$. Algorithms 7.10 and 7.11 show the sequence of interactions of the protocol.

Algorithm 7.9 Module for registering a listener

Initially: $r_0 = p$	
Finally: $r_0 = p$	
broadcaster : $r_0 = p - 1$	picks up one become *sending*
listener : $r_0 = p$	picks one from R_0 becomes receiver
receiver: $r_1 = p - 2$	places one on R_0 become *receiving*

Following a reasoning similar to the one of the proof of Lemma 7.2.3, we have:

Lemma 7.3.1. *The Algorithm composed of Algorithms 7.10 and 7.11 enables the broadcaster to exchange arbitrary messages with a registered listener using a bounded number of moves per digit, when no other agents interfere.*

Finally:

Theorem 7.3.4. *When $n = 1, p > 3$, the broadcaster can exchange arbitrary messages in q-ary encoding (where $q = p - 3$) with each listener.*

Proof. By Algorithm 7.9 a receiver can register. Once a receiver is registered, using Algorithms 7.10 and 7.11, the broadcaster can exchange arbitrary messages with the listener by Theorem 7.3.1 since no other agent will interfere because, in the execution of Algorithms 7.10 and 7.11 p tokens are never placed in R_0. □

Note that, for the case of $p = 4, 5$, a special protocol devised for $n = 1, p = 3$ would be more efficient achieving binary communication. The special protocol is described in the following section.

Algorithm 7.10 Communication Module: COMM(p,1) $p \geq 4$

Initially: $r_0 = p$
Finally: $r_0 = p$

Sender:	
in state *sending*: to communicate	digit $d = i$
$r_0 = p - 1$	pick up all tokens from R_0
	drop $i + 1$ tokens on R_0
	become *i-sent*
in state *i-sent*: $r_0 = p - 1$, to continue	become *sending*
in state *i-sent*: to terminate	
$r_0 \neq 0$,	pick up r_0 tokens
	become *terminating*
in state *i-sent*: to terminate	
$r_0 = 0$,	drop one token
	become *terminating*
in state *terminating*: to switch	
r_s on hand, $r_0 + r_s = p$,(if broadcaster)	drop all but one
r_s on hand, $r_0 + r_s = p - 1$,(if listener)	drop all token
in state *terminating*: for final termination	
r_s on hand, $r_0 + r_s = p$ (if broadcaster)	drop all but one
r_s on hand, $r_0 + r_s = p - 1$ (if listenet)	drop all but two
in state *terminating*: $r_0 = p - 3$	drop a token, become *done*

Algorithm 7.11 Communication Module: COMM(p,1) $p \geq 4$

Receiver:	
in state *receiving*:	
$0 < r_0 = i < p-1$ and empty handed	pick up one token become *i-received*
in state *i-received*:	
$r_0 = p-2$	drop the token, become *receiving*
in state *i-received*:	
$r_0 \neq p-2, i$	drop the token become *wait-for-termination*
in state *wait-for-termination*: $r_0 = p-1$	become *receiving*
in state *wait-for-termination* : $r_0 = p-2$	pick up a token, become *terminating*
in state *terminating* : $r_0 = p-2$	drop a token, become *done*
Broadcaster:	
in state *Done*	drop a token

7.3.3 Special Protocol $n = 1, p = 3$

We are now left only with the case $n = 1, p = 3$, which has not been taken into account so far.

Initially 3 tokens are in room R_0 and both agents have no tokens in hand. A listener registers by picking up a token from R_0.

The idea is for the sender to transmit bit 0 by picking up a single token from R_0; bit 1 by picking up both tokens..

The string to be communicated is coded in binary alphabet, and the idea is to communicate it one bit at a time.
To communicate a *digit* from $(0, 2, 1)$:

- The sender in state *sending* picks up a single token from R_0 to communicate digit 0 becoming *0-sent*: $(1, 1, 1)$; The receiver in state *receiving* seeing 1 token in R_0, understands that bit 0 has been communicated, changes state to *0-received*, and picks up the token to acknowledge reception: $(1, 0, 2)$. The sender in state *0-sent* on seeing R_0 empty, drops its token in R_0 becoming *termination-sending*:

$(0, 1, 2)$. The receiver in state *0-received*, seeing a single token in R_0, drops a token and becomes *termination-receiving*: $(0, 2, 1)$.

- The sender in state *sending* picks up both tokens from R_0 to communicate digit 1 becoming *1-sent*: $(2, 0, 1)$; The receiver in state *receiving* seeing 0 tokens in R_0 drops its token and changes its state to *1-received*: $(2, 1, 0)$. The sender in state *1-sent* seeing a token in R_0 drops one token and becomes *termination-sending*: $(1, 2, 0)$. The receiver in state *1-received* seeing two tokens in R_0 picks both and becomes *wait-for-termination*: $(1, 0, 2)$. The sender in state *termination-sending* seeing R_0 empty drops its token. The receiver in state *wait-for-termination* seeing a token on R_0 drops one token and becomes *termination-receiving* .

- The sender, in state *termination-sending* seeing two tokens on R_0 wanting to continue with another digit, picks up a token from R_1 and becomes *continue-to-send*: $(1, 1, 1)$. The receiver in state *termination-receiving*, on seeing 1 token in R_0, picks it up and becomes *continue-to-receive*. The sender in state *continue-to-send*, seeing R_0 empty, drops its token and becomes *sending*. The receiver in state *continue-to-receive*, seeing a single token in R_0, drops one token and become *receiving*.

- The sender in state *termination-sending* to terminate, picks up both tokens from R_0 and becomes *terminating*: $(2, 0, 1)$. The receiver in state *termination-receiving* seeing 0 tokens in R_0 drops its token and becomes *terminating*: $(2, 1, 0)$. The sender in state *terminating*, seeing a token on R_0 drops a token and becomes *wait-for-switch*:$(1, 2, 0)$. The receiver in state *terminating* for switch picks a token and becomes *switch-sent*:$(1, 1, 1)$; the sender in state *wait-for-switch* seeing one token on R_0 picks up and becomes *switch-received*:$(2, 0, 1)$. The receiver in state *switch-sent* seeing R_0 empty drops a token and becomes *sending*: $(2, 1, 0)$; the sender in state *switch-received* seeing one token on R_0 drops one and becomes *receiving*. On the other hand for final termination, the receiver in state *terminating* picks up two tokens and becomes *Done*. The sender in state *wait-for-switch* seeing R_0 empty drops its token and becomes *Done*. The receiver in state *Done* seeing a token on R_0 drops its tokens.

Algorithm 7.12 Module for registering a listener

Initially: $r_0 = 2$	
Finally: $r_0 = 2$	
broadcaster : $r_0 = 2$	become sender
listener : $r_0 = 3$	picks one from R_0
	become receiver

Theorem 7.3.5. *The Algorithm composed of Algorithms 7.13 and 7.14 enables the broadcaster to exchange arbitrary messages in binary encoding with a registered listener using a bounded number of moves per digit, when no other agents interfere.*

Proof. We show that the transmission of each piece of information (whether a digit or termination) is successfully achieved starting and ending from the *starting configuration*, with the current sender in state *sending* and the *current receiver* in state *receiving* (for digits or switches) and with them in state *done* for full termination.

Initially the system is in the *starting configuration* with the two tokens on R_0 and one with the receiver. By construction, the sender in state *sending* seeing a starting configuration proceeds depending on the bit to be transmitted. By construction, to transmit bit 0 it picks up a single token and becomes *0-sent*. On the contrary, to transmit bit 1 it picks up both tokens, places both on R_1 and becomes *1-sent*. The *receiver* seeing $r_0 \neq 2$ tokens on R_0 understands that a bit has been communicated. The *receiver* in state *receiving* picks a token and becomes *0-received* if $r_0 = 1$ and drop a token and becomes *1-received* if $r_0 = 0$. In the former case the *sender*, seeing that $r_0 = 0$, drops a token on R_0 and becomes *termination-sending*. The *receiver* in state *0-received* seeing a token on R_0, by construction, drops a token on R_0 and becomes *termination-receiving*. In the latter case, the *sender* in state *1-sent* drops a token, becomes *termination-sending*. The *receiver* in state *1-received* seeing two tokens on R_0 picks both and becomes *wait-for-termination*. The *sender* in state *termination-sending* seeing R_0 empty drops a token, The *receiver* in state *wait-for-termination* seeing one tokens on R_0 drops one and becomes *termination-receiving*. Now, regardless of the bit transmitted, the *sender* is in state *termination-sending*, the *receiver* is in state *termination-receiving*, two tokens are on R_0 and one token is with *receiver*.

Algorithm 7.13 Communication Module(for sender): COMM(3,1)

Initially: $r_0 = 3$	
Finally: $r_0 = 3$	
Sender:	
in state *sending*: to communicate	bit $d = 0$
$r_0 = 2$	pick up a single token
	become *0-sent*
to communicate	bit $d = 1$
$r_0 = 2$	picks up both tokens
	become *1-sent*
in state *0-sent*: $r_0 = 0$,	
	drop one
	become *termination-sending*
in state *1-sent*: $r_0 = 1$	drop one
	become *termination-sending*
in state *termination-sending*: $r_0 = 2$	
to continue with next bit	pick a token
	become *continue-to-send*
to terminate	pick both tokens
	become *terminating*
in state *termination-sending*: $r_0 = 0$	drop a token
in state *continue-to-send*: $r_0 = 0$	
	drop one on R_0
	become *sending*
in state *terminating*:	
$r_0 = 1$	drop one
	become *wait-for-switch*
in state *wait-for-switch* : $r_0 = 1$	pick up one token from R_0
	become *switch-received*
in state *switch-received* : $r_0 = 1$	drop one token on R_0
	become *receiving*

Algorithm 7.14 Communication Module(for receiver): COMM(3,1)

Initially: $r_0 = 2$
Finally: $r_0 = 2$

Receiver:	
in state *receiving*:	
$r_0 = 1$	pick up one token become *0-received*
$r_0 = 0$	drop a token become *1-received*
in state *0-received*: $r_0 = 1$,	drop one on R_0 become *termination-receiving*
in state *1-sent*: $r_0 = 2$	pick both become *wait-for-termination*
in state *wait-for-termination*: $r_0 = 1$	drop one become *termination-receiving*
in state *termination-receiving*:	
$r_0 = 1$	pick a token become *continue-to-receive*
$r_0 = 0$	drop a token become *terminating*
in state *terminating*: $r_0 = 2$	
for switch	pick-up one become *switch-sent*
for final termination	pick two become *Done*
in state *continue-to-receive*: $r_0 = 1$	drop a token on R_0 become *receiving*
in state *switch-sent*: $r_0 = 0$	drop one on R_0 become *sending*
in state *Done*: $r_0 = 1$	drop two

By the rules of the algorithm, the next action of the *sender* in state *termination-sending* depends on whether it is going to continue to send bits or is terminating the string of bits. By construction, to continue transmit bits it picks up a single token and becomes *continue-to-send*. On the contrary, to terminate it picks up both and becomes *terminating*. The *receiver* seeing $r_0 \neq 2$ tokens on R_0 acts depending on r_0 by the rules of the algorithm. The *receiver* in state *termination-receiving* on seeing a single token on R_0 picks it up and becomes *continue-to-receive*. The *sender* seeing that $r_0 = 0$ drops a token on R_0 and becomes *sending*. The *receiver* in state *continue-to-receive* seeing a token on R_0, by construction, drops a token on R_0 and becomes *receiving*. The *starting configuration* is reached and the sender can continue with the transmission of next bit. Hence the theorem holds.

In the case that the string of bits have terminated the *receiver* in state *termination-receiving* on seeing no tokens on R_0, drops a token and becomes *terminating*. The sender in state *terminating* drops a token and becomes *wait-for-switch*. Now the actions of the receiver in state *terminating* depends on whether a reply i.e. switch is required or not. If it is required, by construction, it picks a token on R_0 and becomes *switch-sent*. Otherwise, it picks both tokens on R_0 and becomes *Done*. Now the action of the *sender* in state *terminating* depends on r_0. In the case that the switch is not required *sender* seeing zero tokens on R_0 drop one and becomes *Done*. The receiver in state *Done* seeing a token on R_0 drops all token. Hence the theorem holds.

In the case that switch is required *sender* seeing a token picks it up and becomes *switch-received*. Now the *receiver* in state *switch-sent* seeing no token on R_0 drops it token on R_0 and becomes *sending*. The *sender* in the state *switch-received* seeing a token on R_0 drops a token and becomes *receiving*. The *starting configuration* is reached and the new sender can continue with the transmission of next bit. Hence the theorem holds. Hence, the assertion of the theorem is established. □

Theorem 7.3.6. *When $n = 1, p = 3$, the broadcaster can exchange arbitrary messages in binary encoding with each listener.*

Proof. By Algorithm 7.12 a receiver can register. When the receiver is registered two tokens are in R_0. The listener's first interaction occurs when R_0 contains three tokens; however, that configuration is not reached in the execution of Algorithms 7.13 (nor

7.14). It then follows that no interference can occur and correctness follows from the proof of Lemma 7.3.5. □

7.4 Summary

In this chapter, we presented novel communication protocols using tokens. We began with the protocols for two agents. Then we generalized those protocols to the broadcast problem with multiple agents.

In particular, for the case of two agents, we have shown that, when only one token is available, bounded communication is impossible both in case of unidirectional and bidirectional communication; bidirectional communication is impossible also when two tokens are available in a single room. In all other cases, we have shown that bidirectional communication is possible and we have provided protocols indicating the impact of the amount of resources available on the alphabet used to communicate the information.

For the case of multiple agents, we establish that broadcasting is possible whenever bidirectional communication between two agents is possible and we describe various solution protocols.

Chapter 8

Conclusions and Open Problems

In this thesis, we studied the problem of black hole search and we introduced some general communication mechanisms using tokens.

Chapters 4 and 5 focus on the black hole search problem in the specific topology of the ring, when agents communicate through whiteboards.

In Chapter 4 we considered black hole search with the goal of optimizing the *time complexity*. We established a lower bound on the average time complexity and we provided an algorithm matching the lower bound using $2(n-1)$ agents. We also designed a protocol using $n-1$ agents that has complexity two time units more than the existing worst-case complexity, but better average case complexity. We simultaneously optimized the team size and the asymptotic time complexity. In Chapter 5, we turned our attention toward *move complexity* and we determined the exact move complexity of the problem with a lower bound an an algorithm which matches it. We then presented an algorithm that simultaneously achieves optimal asymptotic time, exact move complexity and team size.

An interesting problem regarding black hole search in the whiteboard model would be to derive the exact move and ideal time complexities in topologies other than the ring, and in networks with arbitrary topology.

In Chapter 6 we investigated the black hole search problem when agents communicate through token manipulation. In this case we considered arbitrary unknown graphs. We showed that, in order to use a constant number of tokens, 3 tokens and $\Delta+2$ agents are required, where Δ is a known upper bound on the maximum degree. We presented an algorithm using 3 tokens with move complexity $O(mn)$. Several problems remain open on black hole search without a map, both in the whiteboard and in the token model. For example:

1. What is the whiteboard size required for black hole search with optimal team

size of $\Delta + 1$?

2. What is the move complexity for black hole search using constant number of tokens and constant size whiteboard?

3. In the enhanced token model, pebbles can be placed, not only on the "centre" of a node (in a "room"), but also in correspondence of its incident edges (on the "doors" of the room). This model is clearly more powerful than the usual token model and some results on black hole search have been established already for this model. In particular, it has been shown that it is possible to locate a black hole using $\Delta + 1$ agents (which is optimal) using $O(\Delta^2 m^2 n^7)$ moves , where n is the number nodes and m is the number of links [32]. What is the complexity of black hole search using optimal team size in the enhanced token model?

4. For the problem of black hole search with unknown map using whiteboard, it is shown in [37] that though for arbitrary graphs the move complexity is $\Omega(n^2)$ there are large classes of graphs where a better complexity could be achieved when $m = o(n^2)$. Are there similar protocols in the token model ?

Chapter 7 presents several protocols for communicating using tokens and the application of *Broadcast* is discussed. An interesting direction will be apply these protocols for other problems, for example: leader election, rendezvous, map construction, gossiping. Another interesting problem is to determine the exact number of tokens required for those problems. Observe that the number of tokens used corresponds to cumulative whiteboard size that is used in all the node of the network at a moment. Several questions remain open, for example:

1. Is it possible to solve any problem solvable using a team of k agents in the whiteboard model in safe networks using the same number of agents and a constant number of tokens?

2. Is it possible to solve any problem using a team of k agents in the whiteboard model in unsafe networks using $k + 1$ agents and a constant number of tokens?

In the models we considered agents begin their execution from the same node. Can the technique of using tokens to communicate be adapted for the case in which

agents are anonymous and are scattered ? One difficulty is immediately evident and it is due to possible symmetric placements of the agents. In fact, if the network and the location of the agents are symmetric, the agents will perform identical actions and symmetry can not be broken; on the other hand, communication mechanisms could possibly be devised also in those settings.

Determining the total number of bits transmitted in black hole search and other mobile agent algorithms is another interesting future direction. Determining the trade-offs in the resources such as time, moves, whiteboard size or tokens, total number of bits transmitted has not been solved for many problems. Establishing these trade-offs and providing constructive protocols would lead to effective solutions for different environments.

Bibliography

[1] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

[2] C. Ambühl, L. Gasieniec, A. Pelc, T. Radzik, and X. Zhang. Tree exploration with logarithmic memory. *ACM Transactions on Algorithms*, 7(2):17:1–17:21, 2011.

[3] I. Averbakh and O. Berman. A heuristic with worst-case analysis for minimax routing of two traveling salesmen on a tree. *Discrete Applied Mathematics*, 68(1–2):17 – 32, 1996.

[4] I. Averbakh and O. Berman. (p - 1)/(p + 1)-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective. *Discrete Applied Mathematics*, 75(3):201–216, 1997.

[5] B. Awerbuch, M. Betke, R. L. Rivest, and M. Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152(2):155–172, 1999.

[6] B. Balamohan, S. Dobrev, P. Flocchini, and N. Santoro. Asynchronous exploration of an unknown anonymous dangerous graph with o(1) pebbles. In *Proceedings of the 19th International Conference on Structural Information and Communication Complexity*, SIROCCO'12, pages 279–290, 2012.

[7] B. Balamohan, P. Flocchini, A. Miri, and N. Santoro. Time optimal algorithms for black hole search in rings. In *Proceedings of the 4th International Conference on Combinatorial Optimization and Applications - Volume Part II*, COCOA'10, pages 58–71, 2010.

[8] B. Balamohan, P. Flocchini, A. Miri, and N. Santoro. Improving the optimal bounds for black hole search in rings. In *Proceedings of the 18th International Conference on Structural Information and Communication Complexity*, SIROCCO' 11, pages 198–209, 2011.

[9] B. Balamohan, P. Flocchini, A. Miri, and N. Santoro. Time optimal algorithms for black hole search in rings. *Discrete Mathematics, Algorithms and Applications*, 3(4):457–472, 2011.

[10] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '02, pages 200–209, 2002.

[11] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Can we elect if we cannot compare? In *Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 324–332, 2003.

[12] M. A. Bender, A. Fernández, D. Ron, A. Sahai, and S. P. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation*, 176(1):1–21, 2002.

[13] M. A. Bender and D. K. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, FOCS '94, pages 75–85, 1994.

[14] L. Blin, P. Fraigniaud, N. Nisse, and S. Vial. Distributed chasing of network intruders. In *Proceedings of the 13th International Conference on Structural Information and Communication Complexity*, SIROCCO'06, pages 70–84, 2006.

[15] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, FOCS'78, pages 132–142, 1978.

[16] N. Borselius. Mobile agent security. *Electronics Communication Engineering Journal*, 14(5):211 – 218, 2002.

[17] J. Chalopin, S.Das, A. Labourel, and E. Markou. Black hole search with finite automata scattered in a synchronous torus. In *Proceedings of the 25th International Symposium on Distributed Computing*, DISC'11, pages 432–446, 2011.

[18] J. Chalopin, S.Das, A. Labourel, and E. Markou. Tight bounds for scattered black hole search in a ring. In *Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity*, SIROCCO'11, pages 186–197, 2011.

[19] D. M. Chess. Security issues in mobile code systems. In *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 1–14. 1998.

[20] C. Cooper, R. Klasing, and T. Radzik. Searching for black-hole faults in a network using multiple agents. In *Proceedings of the 10th International Conference on Principles of Distributed Systems*, OPODIS'06, pages 320–332, 2006.

[21] C. Cooper, R. Klasing, and T. Radzik. Locating and repairing faults in a network with mobile agents. In *Proceedings of the 15th International Colloquium on Structural Information and Communication Complexity*, SIROCCO '08, pages 20–32, 2008.

[22] J. Czyzowicz, S. Dobrev, R. Královič, S. Miklík, and D. Pardubská. Black hole search in directed graphs. In *Proceedings of the 17th International Colloquium*

on Structural Information and Communication Complexity, SIROCCO'10, pages 182–194, 2010.

[23] J. Czyzowicz, D. R. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability & Computing*, 16(4):595–619, 2007.

[24] A. Das and Y. Gongxuan. A secure payment protocol using mobile agents in an untrusted host environment. In *Proceedings of the 2nd International Symposium on Topics in Electronic Commerce*, ISEC '01, pages 33–41, 2001.

[25] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science*, 385(1-3):34–48, 2007.

[26] S. Das, P. Flocchini, A. Nayak, and N. Santoro. Distributed exploration of an unknown graph. In *Proceedings of the 12th international Colloquium on Structural Information and Communication Complexity*, SIROCCO'05, pages 548–548, 2005.

[27] S. Das, P. Flocchini, A. Nayak, and N. Santoro. Effective elections for anonymous mobile agents. In *Proceedings of the 17th International Symposium on Algorithms and Computation*, ISAAC'06, pages 732–743, 2006.

[28] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

[29] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. *Theoretical Computer Science*, 326(1-3):343–362, 2004.

[30] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.

[31] S. Dobrev, P. Flocchini, R. Královič, P. Ruzicka, G. Prencipe, and N. Santoro. Black hole search in common interconnection networks. *Networks*, 47(2):61–71, 2006.

[32] S. Dobrev, P. Flocchini, R. Královič, and N. Santoro. Exploring an unknown graph to locate a black hole using tokens. In *Proceedings of the 4th IFIP International Conference on Theoretical Computer Science*, TCS'06, pages 131–150, 2006.

[33] S. Dobrev, P. Flocchini, R. Královič, and N. Santoro. Exploring an unknown dangerous graph using tokens. *Theoretical Computer Science*, 472(0):28 – 45, 2013.

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

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

[36] S. Dobrev, P. Flocchini, and N. Santoro. Improved bounds for optimal black hole search with a network map. In *Proceedings of the 10th International Colloquium on Structural Information and Communication Complexity*, SIROCCO'04, pages 111–122, 2004.

[37] S. Dobrev, P. Flocchini, and N. Santoro. Cycling through a dangerous network: A simple efficient strategy for black hole search. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, ICDCS'06, pages 57–, 2006.

[38] S. Dobrev, P. Flocchini, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48:67–90, 2007.

[39] S. Dobrev, R. Královič, N. Santoro, and W. Shi. Black hole search in asynchronous rings using tokens. In *Proceedings of the 6th International Conference on Algorithms and Complexity*, CIAC'06, pages 139–150, 2006.

[40] S. Dobrev, N. Santoro, and W. Shi. Locating a black hole in an un-oriented ring using tokens: The case of scattered agents. In *Proceedings of the 13th International European Conference on Parallel and Distributed Computing*, Euro-Par'07, pages 608–617, 2007.

[41] S. Dobrev, N. Santoro, and W. Shi. Scattered black hole search in an oriented ring using tokens. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, IPDPS'07, pages 1–8, 2007.

[42] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[43] M. Dynia, J. Lopuszanski, and C. Schindelhauer. Why robots need maps. In *Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity*, SIROCCO'07, pages 41–50, 2007.

[44] P. Flocchini, M. J. Huang, and F. L. Luccio. Decontaminating chordal rings and tori using mobile agents. *International Journal of Foundations of Computer Science*, 18(3):547–563, 2007.

[45] P. Flocchini, M. J. Huang, and F. L. Luccio. Decontamination of hypercubes by mobile agents. *Networks*, 52(3):167–178, 2008.

[46] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. In *Proceedings of the 11th International Conference on Principles of Distributed Systems*, OPODIS'07, pages 105–118, 2007.

[47] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theoretical Computer Science*, 411(14-15):1583–1598, 2010.

[48] P. Flocchini, D. Ilcinkas, and N. Santoro. Ping pong in dangerous graphs: Optimal black hole search with pure tokens. In *Proceedings of the 22nd International Symposium on Distributed Computing*, DISC '08, pages 227–241, 2008.

[49] P. Flocchini, D. Ilcinkas, and N. Santoro. Ping pong in dangerous graphs: Optimal black hole search with pebbles. *Algorithmica*, 62:1006–1033, 2012.

[50] P. Flocchini, M. Kellett, P. Mason, and N. Santoro. Map construction and exploration by mobile agents scattered in a dangerous network. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium*, IPDPS'09, pages 1–10, 2009.

[51] P. Flocchini, M. Kellett, P. C. Mason, and N. Santoro. Finding good coffee in Paris. In *Proceedings of the 6th International Conference on Fun with Algorithms*, FUN'12, pages 154–165, 2012.

[52] P. Flocchini, M. Kellett, P. C. Mason, and N. Santoro. Searching for black holes in subways. *Theory of Computing Systems*, 50(1):158–184, 2012.

[53] P. Flocchini, B. Mans, and N. Santoro. Sense of direction: Formal definitions and properties. In *Proceedings of the 1st Colloquium on Structural Information and Communication Complexity*, SIROCCO'94, pages 9–34, 1994.

[54] P. Flocchini, B. Mans, and N. Santoro. Exploration of periodically varying graphs. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, ISAAC'09, pages 534–543, 2009.

[55] P. Fraigniaud, L. Gasieniec, D. R. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006.

[56] P. Fraigniaud and D. Ilcinkas. Digraphs exploration with little memory. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science*, STACS'04, pages 246–257, 2004.

[57] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.

[58] J. A. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27(1):302–316, 1998.

[59] P. Glaus. Locating a black hole without the knowledge of incoming link. In *5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, ALGOSENSORS'09, pages 128–138, 2009.

[60] M. S. Greenberg, J. C. Byington, and D. G. Harper. Mobile agents and security. *CIEEE Communications Magazine*, 36(7):76–85, 1998.

[61] F. Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Mobile Agents and Security*, pages 92–113. 1998.

[62] F. Hohl. A framework to protect mobile agents by using reference states. In *Proceedings of the 20th International Conference on Distributed Computing Systems*, ICDCS'00, pages 410–417, 2000.

[63] J. Chalopin, S. Das, and N. Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. In *Proceedings of the 21st Conference on Distributed Computing*, DISC'07, pages 108–122, 2007.

[64] Y. Jung, M. Kim, A. Masoumzadeh, and J. Joshi. A survey of security issue in multi-agent systems. *Artificial Intelligence Review*, 37:239–260, 2012.

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

[66] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Approximation bounds for black hole search problems. *Networks*, 52(4):216–226, 2008.

[67] A. Kosowski, A. Navarra, and C. M. Pinotti. Synchronous black hole search in directed graphs. *Theoretical Computer Science*, 412(41):5752 – 5759, 2011.

[68] A. Kosowski, A. Navarra, and M. C. Pinotti. Synchronization helps robots to detect black holes in directed graphs. In *Proceedings of the 13th International Conference on Principles of Distributed Systems*, OPODIS 2009, pages 86–98, 2009.

[69] R. Královič and S. Miklík. Periodic data retrieval problem in rings containing a malicious host. In *Proceedings of the 17th International Colloquium on Structural Information and Communication Complexity*, SIROCCO'10, pages 157–167, 2010.

[70] E. Kranakis, D. Krizanc, and E. Markou. Mobile agent rendezvous in a synchronous torus. In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics*, LATIN'06, pages 653–664, 2006.

[71] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998.

[72] S. Ng and K. Cheung. Protecting mobile agents against malicious hosts by intention spreading. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, PDPTA'99, pages 725–729, 1999.

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

[74] P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33(2):281–295, 1999.

[75] M. Perry and Q. Zhang. Sita: Protecting internet trade agents from malicious hosts. In *Proceedings of the 3rd International Workshop on Mobile Agents for Telecommunication Applications*, MATA '01, pages 173–183, 2001.

[76] M. O. Rabin. Maze threading automata. October 1967. Seminar talk presented at the University of California at Berkeley.

[77] T. Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, volume 1419, pages 44–60. 1998.

[78] N. Santoro. Sense of direction, topological awareness and communication complexity. *SIGACT News*, 16(2):50–56, July 1984.

[79] K. Schelderup and J. Ones. Mobile agent security - issues and directions. In *Proceedings of the 6th International Conference on Intelligence and Services in Networks: Paving the Way for an Open Service Market*, IS&N '99, pages 155–167, 1999.

[80] C. E. Shannon. Presentation of a maze-solving machine. In *Proceedings of 8th Cybernetics Conference, Josiah Macy Jr. Foundation*, pages 173–180, 1951.

[81] W. Shi. Black hole search with tokens in interconnected networks. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS'09, pages 670–682, 2009.

[82] G. J. van 't Noordende, F. M. T. Brazier, and A. S. Tanenbaum. Security in a mobile agent system. In *Proceedings of the 1st IEEE Symposium on Multi-Agent Security and Survivability*, MASS'04, pages 35–45, 2004.

[83] X. Yu and M. Yung. Agent rendezvous: A dynamic symmetry-breaking problem. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, ICALP'96, pages 610–621, 1996.