uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Shantanu Das
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Ph.D. (Computer Science)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Distributed Computing with Mobile Agents :
Solving Rendezvous and Related Problems

TITRE DE LA THÈSE / TITLE OF THESIS

Amiya Nayak
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

Nicola Santoro
CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Jaroslav Opatrny | Evangelos Kranakis

Paola Flocchini | Nejib Zaguia

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Distributed Computing with Mobile Agents: Solving Rendezvous and Related Problems

by

SHANTANU DAS

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfilment of
the requirements for the degree of
Ph.D. Computer Science

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada.

November 2007

©2007, Shantanu Das

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-49344-1
Our file Notre référence
ISBN: 978-0-494-49344-1

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Abstract

In this thesis we study *mobile agent systems* consisting of a network of nodes and a set of autonomous agents travelling through the network. The mobile agents can traverse the edges(channels) of the network to move from one node to another and they can communicate with each other using public *whiteboards* available at each node of the network. We focus on the *Rendezvous* problem which is the central problem in such systems. The objective of the *Rendezvous* problem is to gather all the agents at a single node of the network. A procedure for Rendezvous is useful while performing many collaborative task by distributed agents—for instance, the agents may need to gather together for planning and coordinating their next operation. The problem of Rendezvous is in fact, related to several other problems in this setting, such as leader election, network exploration, distributed spanning-tree construction and graph-labelling or enumeration.

Compared to most existing solutions, we study the Rendezvous problem in a weaker (and computationally difficult) model where both the agents and the network nodes are anonymous, and the network is asynchronous. Our focus in this thesis is not only determining when the problem is solvable in this setting but also designing efficient algorithms for the solvable cases. The main measure of efficiency in our algorithms is the number of moves (edge traversals) made by the agents in total. A secondary goal is to minimize the memory required for the whiteboard at each node of the network.

We present solutions for the Rendezvous problem, both in the whiteboard model and the more restrictive *token*-model. We also consider situations where some network components may fail or the tokens used by the agents may disappear. We present fault-tolerant algorithms for these cases. We also show how to simulate any distributed computation in mobile agent systems when some of agents crash unexpectedly. Our results show that any problem that can be solved in message-passing systems can be solved in mobile agent systems while tolerating any number of crash faults, short of a total system collapse.

// Acknowledgments

I take this opportunity to convey my sincere thanks to my thesis supervisors Dr. Amiya Nayak and Dr. Nicola Santoro, whose constant encouragement, help and support over the last four years, made this work possible. I am specially grateful to Prof. Nayak for always believing in me and giving me the freedom to work on the problems that I liked the most. His great enthusiasm, sound advice and timely help in all academic matters made it easy for me to complete this work.

It was a pleasure for me to work with Prof. Santoro whose passion and enthusiasm for research has always fascinated me. He is undoubtedly one of the best teachers that I have known and his advice and guidance has been essential in my professional development as a researcher.

Many other professors from both universities helped me in my work. I am very grateful to Dr. Paola Flocchini whose advice, help and support were invaluable in completing this work. My sincere thanks goes to Dr. Evangelos Kranakis whose comments and advice helped improve the thesis considerably. I would like to thank Dr. Stefan Dobrev for introducing me to the basic concepts of distributed computing. I thoroughly enjoyed his classes and the stimulating discussions I had with him both inside and outside the class.

My interactions with many other researchers helped in developing my understanding of this field of research and influenced the writing of this thesis. My sincere thanks goes to Jérémie Chalopin, Shay Kutten, and Masafumi Yamashita for many helpful discussions.

I am grateful to the University of Ottawa for providing the ideal atmosphere and excellent resources for conducting high quality research. The staff and students at both the University of Ottawa and Carleton University have been very helpful during the time that I spent here. I am also grateful to the Ontario Government and CISTEL Technologies Inc. for generously funding my research.

Finally, I would like to thank my parents who have always been behind me, supporting and encouraging me in all my endeavors.

Contents

1 Introduction **1**
1.1 Overview 1
1.2 Motivation and Background 3
1.3 Main Contributions 4
1.4 Thesis Organization 4

2 The Model, Definitions and Properties **6**
2.1 Our Model 6
2.1.1 Mobile Agent Model 6
2.1.2 Whiteboards versus Tokens 8
2.1.3 Initial Knowledge 9
2.2 Other Models 9
2.2.1 Stationary Agent Model 9
2.2.2 The Mobile Robots Model 10
2.3 Definitions and Problems 10

3 Review of Known Results **14**
3.1 Solutions for the Mobile Agent Model 14
3.1.1 Agent Rendezvous 14
3.1.2 Graph Traversal and Exploration 15
3.1.3 Capturing an Intruder (Graph Search) 16
3.1.4 Locating Faults (Black Holes) 17
3.2 Solutions for the Stationary Agent Model 17
3.2.1 Leader Election 17
3.2.2 Spanning Tree Construction 18
3.2.3 Distributed Enumeration or Labelling 18
3.3 Solutions for the Mobile Robots Model 18

4 Basic Techniques for Computing in Anonymous Networks **20**
4.1 Computing the View of an agent . . . 20
4.2 Constructing the minimum base . . . 23
4.2.1 Symmetric Directed Graphs . . . 23
4.2.2 Digraph Coverings and their Properties . . . 24
4.3 Distributed Traversal Algorithm . . . 24

5 Rendezvous in the Token Model **28**
5.1 Solving Rendezvous in the Token Model . . . 28
5.2 Rendezvous When Tokens Fail . . . 30
5.2.1 In Ring Networks . . . 31
5.2.2 In Arbitrary Networks . . . 37
5.3 Conclusions . . . 39

6 Rendezvous in the Whiteboard Model **40**
6.1 Conditions for Solving Rendezvous . . . 40
6.2 Solving RV when n and k are co-prime . . . 41
6.2.1 Algorithm MERGE-TREE . . . 41
6.2.2 Analysis of the Algorithm . . . 44
6.2.3 Reducing the number of Phases . . . 48
6.3 Designing Effective Solutions . . . 51
6.3.1 Partial-Views . . . 51
6.3.2 Algorithm *Agent-Elect* . . . 54
6.3.3 Reducing the Size of the Whiteboards . . . 57
6.4 Summary of Results and Discussions . . . 60

7 Grouping and Near-Gathering **62**
7.1 The Grouping Problem . . . 62
7.1.1 Definitions and Properties . . . 63
7.1.2 Solution Protocol . . . 64
7.2 Near-Gathering . . . 66
7.2.1 Necessary Conditions . . . 66
7.2.2 Solution and Sufficient Conditions . . . 67
7.3 Conclusions . . . 69

8 Rendezvous in Faulty Networks **71**
8.1 Some Definitions and Properties . . . 71
8.2 Characterizations based on rendezvous problem . . . 72

8.3 Initial Knowledge Required 75
8.4 Solution Protocols 77
8.5 Conclusions 81

9 Computing with Faulty Mobile Agents **82**
9.1 Simulating a Distributed Algorithm 83
9.2 Computing in Anonymous Graphs 89
9.3 Conclusions 95

10 Conclusions and Future Work **96**
10.1 Summary of Results 96
10.2 Open Problems 98

Chapter 1

Introduction

1.1 Overview

A distributed system consists of multiple autonomous entities which are communicating and cooperating with each-other to achieve a common goal. The most prevalent model of distributed systems is one where the entities are stationary and they exchange information using fixed communication links between pairs of entities. This is called the point-to-point message-passing network model. Another model of distributed computation is the mobile agent model, where the entities (called agents) are mobile, moving on a fixed network of paths. Here the communication between the agents is achieved either by direct information exchange when two agents meet or, by reading and writing information in some fixed locations called whiteboards. The environment is usually modelled as a graph, where each node contains a whiteboard and the agents may traverse the edges of the graph to move from one node to another.

The mobile agent model studied in this thesis has been inspired by the use of software agents (mobile code that can move autonomously from host to host) in the Internet, for performing a variety of tasks such as collecting information, negotiating a deal, online shopping etc. The use of mobile agents has been advocated for various reasons such robustness against disruptions in network connectivity, improving the network latency and reducing network load, providing more autonomy in workflow solutions and so on [81]. A major concern with mobile agents has been ensuring security both for the agents and the hosts[34, 88]. Host computers must be protected from malicious agents such as viruses and worms. On the other hand, the agents have to protected from being tampered by malicious hosts[71, 96]. In spite of the security concerns, mobile agent systems are popular as an alternative model of computation and they are expected to gain more popularity, once the security issues has been properly addressed.

In general, the mobile agent model makes it easier to describe and study distributed environments where mobility is inherent. The study of such systems has recently gained momentum

with the introduction of several new problems in this setting such as *rendezvous*, *intruder detection*, *network decontamination*, *distributed search*, *locating black holes* etc. However, this model is relatively new and our understanding of such systems is not complete. In this thesis, we study mobile agent systems by focussing on the central problem in such systems—the *Rendezvous* or gathering of mobile agents. An algorithm for Rendezvous could be used as a basic procedure in any task involving the coordination among multiple mobile entities. Moreover, the problem of Rendezvous is also related to some well-known problems in distributed computing such as *leader election*, *graph exploration* and *enumeration*. Thus, the study of the Rendezvous problem should also provide more insight into some of these problems.

The *Rendezvous* problem was first proposed as an optimization problem by Steven Alpern during a seminar he presented in 1976, in Vienna[4]. He stated it as follows:

> Two astronauts land on a spherical body that is much larger than the detection radius (within which they can see each other). The body does not have fixed orientation in space, nor does it have an axis of rotation, so that no common notion of position or direction is available to the astronauts for coordination. Given unit walking speeds for both astronauts, how should they move about so as to minimize the expected meeting time T (before they come within the detection radius)?

The problem attracted the attention of mathematicians and scientists and since then various versions of the problem has been studied by researchers, under different settings. The problem has been talked about in the context of two ships lost in the sea, two friends separated from each-other in a crowded mall, or a team of rescuers trying to find a lost and distraught explorer. In most cases, the objective has been to find a strategy for movement (of both parties) that minimizes the expected time to meet. Such solutions thus resort to some kind of randomization and do not actually guarantee Rendezvous within a finite time. In this thesis, on the other hand we consider only deterministic solutions to the Rendezvous problem. In those cases when deterministic solution to Rendezvous is not possible, we want the agents to detect this within a finite time and terminate.

There has been some studies on the comparison between mobile agent systems and the conventional message-passing systems [66]. It is known that the two models are computationally equivalent in the presence of whiteboards, such that any problem is solvable in one model if and only if it is also solvable in the other model [32]. This makes it possible to determine which problems are solvable in the mobile agent systems using the impossibility results already established for similar problems in message-passing systems. However, it must be noted that there are considerable differences between the two models in terms of complexity of solutions. The two models use different measures of complexity. Thus, designing algorithms for the mobile agent systems requires us to think from a different perspective and use techniques minimizing

movement rather than direct communication. This thesis highlights some of these differences between the two models.

1.2 Motivation and Background

The study of mobile agent systems is motivated by the fact that mobility is inherent for many tasks in distributed environments, for example searching and locating information, or monitoring a computer network and so on. Mobile agents are also well suited for computation in dynamic networks [68] because of their ability to autonomously relocate from one host to another.

Another motivation for studying distributed mobile environments comes from the field of robotics, i.e. the coordination among a team of robots moving in an unknown environment. Even though the robot model is slightly different from the model studied here, some of our results (with minor modifications) are applicable also in the mobile robots model when the environment explored by the robots can be modelled as a graph.

The Rendezvous problem, i.e. the gathering the mobile agents to a single location, is the central problem in mobile agent systems. Many other tasks such as mapping the network, capturing an intruder, or locating a fault, become much easier once the agents are co-located. Moreover, in case the network structure changes dynamically, the agents need to gather periodically in order to re-group and reorganize their strategy. Thus, the Rendezvous problem is a basic task in distributed environments containing autonomous mobile entities.

Most previous work on the Rendezvous problem assumed unique identities or labels have been assigned to the agents. However, in many situations such unique labels may not be available. Moreover for security considerations, it may be necessary to keep the identities of agents hidden. Thus, we consider the agents to be anonymous and the network structure to be initially unknown. Our objective is to design algorithms for solving Rendezvous in any arbitrary network topology and for any initial placement of the agents. Thus, our solutions could be also applied to dynamic networks such as ad-hoc mobile networks where the network structure is not fixed in advance and changes occur dynamically.

The mobile agent model considered in this thesis is one of the weakest possible model. The motivation for studying the weakest model is that it allows us to better understand the computability issues and to determine under what conditions the problem is solvable. Another advantage is that any solutions designed for the weaker model can be used in stronger models too. Thus, the results obtained in this thesis are generic solutions that are applicable in many different scenarios.

1.3 Main Contributions

The main contributions of this thesis are as follows. We first survey the existing literature on Rendezvous of mobile agents and show how the problem is related to some other well-known problems. We then present solutions for Rendezvous of mobile agents in both the token model and the whiteboard model. Existing solutions to the Rendezvous problem assume that the agents have unique labels, are moving in a synchronous environment or, the network topology is known in advance. We present the first solution to the Rendezvous problem for *anonymous*, *asynchronous* agents in an unknown and *arbitrary* network without any *sense of direction*. This solution is applicable only when the network size, n and the number of agents, k are co-prime to each other (a condition which guarantees deterministic solution to Rendezvous). For the general case, we provide *effective* algorithms i.e. algorithms which solve Rendezvous whenever it is deterministically possible and otherwise detects the fact that it is not solvable. Our results show that when solving Rendezvous in the whiteboard model, there is a tradeoff between the two cost measures viz the number of agent moves required and the amount of whiteboard memory used.

We also study two relaxed versions of the Rendezvous problem called *Grouping* (i.e. gathering in multiple groups) and *Near-Gathering* (i.e. gathering within a small area). The former problem is introduced in this thesis, while the latter one has been studied previously, though only in the simplest case (i.e. for ring networks). For these two problems, we characterize those networks where the problems are solvable and present simple solution protocols.

Finally we explore fault-tolerant solutions to the Rendezvous problem. We consider two scenarios — (i) when the environment, i.e. the network, is faulty and (ii) when the agents themselves may develop faults. We study the Rendezvous problem in networks containing faults at arbitrary locations and provide a characterization of faulty networks where Rendezvous is possible. This is a major extension of previous work which considered only networks of fixed topology with one specific faulty node called the *Black Hole*. We are also the first to study mobile agents systems where agents may crash. For such a scenario, we developed a fault-tolerant way of simulating any distributed computation with mobile agents. We show that our simulation works correctly even if up to $k-1$ out of k agents fail, achieving the same result as would have been obtained in the absence of failures.

1.4 Thesis Organization

The thesis is organized as follows. In Chapter 2, we present a complete description of the mobile agent model that we consider in this thesis. We present two variations of the model — one uses tokens and the other uses whiteboards for inter-agent communication. We compare our

model with the traditional *stationary agent* model and also the *mobile robot* model. We then define the Rendezvous problem and show how it is related to some other commonly known problems. In the next chapter (Chapter 3), we review previous work related to solving the Rendezvous problem. We include results for each of the models discussed above. In Chapter 4, we discuss some known concepts and techniques which are used to obtain many of the results in this thesis. In particular, we define the concepts of *views*, graph coverings and *minimum bases*. We also present an algorithm for the distributed exploration of a network by multiple agents. This method is later used as a sub-procedure in some of the algorithms.

In the next four chapters, we present and analyze solutions to the problem under different settings. Chapter 5 and Chapter 6 solve the Rendezvous problem in the token model and the whiteboard model respectively. In Chapter 7 we consider those scenarios where it is not possible to achieve Rendezvous by deterministic means. For these scenarios we present algorithms for solving the *Grouping* and *Near-gathering* problems, which can be seen as relaxed version of the Rendezvous problem. In Chapter 8, we consider fault-tolerant solutions to Rendezvous. We characterize the conditions for solving Rendezvous in networks containing faulty nodes and edges, and present algorithms for the Rendezvous of the maximum number of agents possible in such faulty networks.

Finally, in Chapter 9, we show how to simulate any distributed algorithm in a mobile agent system when agents may crash or suddenly disappear. Chapter 10 summarizes all the results obtained in this thesis and presents some open problems that can be considered for future work on this problem.

Chapter 2

The Model, Definitions and Properties

In this chapter, we describe in detail our model for mobile agent systems and some of its variations. We also compare our model with the standard model for distributed computation in networked systems viz the *Message-Passing Network* model (which we refer to as the *Stationary-Agent model*). We also briefly mention some other models found in the literature. Note that all algorithms and results in this thesis are for the mobile agent model; The other models are only presented for comparison.

The chapter also defines some of the terminologies used in this thesis. In particular, we give a formal definition of the rendezvous problem and show how it is related to some other well-known problems in distributed computing. Finally we present some known properties regarding the solvability of these problems.

2.1 Our Model

2.1.1 Mobile Agent Model

The environment is modelled as an undirected connected graph $G(V, E)$, containing $n = |V|$ nodes and $m = |E|$ edges. There are k computational entities (called *mobile agents*) each of which may be located at any of the nodes of the graph at any time during the computation. Each edge of the graph represents a bidirectional channel on which agents can travel in either direction. There is no restriction on the number of agents that may be traversing an edge simultaneously.

The nodes of the graph are labelled by the labelling function $L : V(G) \rightarrow \Sigma$ and these labels are visible to the agents. In general, we want to design algorithm which work irrespective of the labelling and thus, we shall assume that the labels assigned to the nodes are identical. In

other words, the network is **anonymous**, unless otherwise specified.

There exists a local orientation among the edges incident at each node of the graph, so that an agent arriving at a node can distinguish among them. This local orientation is defined by an edge labelling of the graph G given by $\lambda = \{\lambda_v : v \in V\}$, where for each vertex v of degree d, $\lambda_v : \{e(v,u) : u \in V \text{ and } e \in E\} \rightarrow \{1, 2, ...d\}$ is a bijection specifying the local labelling at v.

Each agent is initially located in a distinct node of the graph, called its homebase. In general, we shall assume that no two agents have the same homebase. The initial placement of agents in the graph G is denoted by the placement function $p : V \rightarrow \{0, 1\}$ on the set of vertices, where those vertices that are colored 1(or, black) are the homebases of the agents. Thus, the environment would be represented by the tuple (G, L, λ, p) or (G, λ, p) (when all nodes have the same label, the labelling function L would be omitted).

In this model, the nodes of the graph are just repositories of data, with no "intelligence" (i.e. computational ability) associated with them. The local memory at each node, called the *whiteboard* of that node, is accessible to any agent that is physically present at this node. The contents of the *whiteboard* defines the state of the node. The agents communicate by reading and writing information on the whiteboards available at the nodes of the network. Access to the whiteboard is restricted by fair mutual exclusion, so that, at most one agent can access the whiteboard of a node at the same time and every requesting agent is granted access within a finite time. Initially, the whiteboard of a node v contains the value $p(v)$ and the label $L(v)$ of the node.

Each agent also has its individual memory that is accessible only by that agent and is called its *notebook*; This local memory moves with the agent when it travels from one node to another. The contents of *notebook* defines the state of the agent (in particular, it contains a special variable called *Next-Node*). Each agent starts in the same initial state with the notebook containing only the algorithm to be executed.

The agents are *anonymous* in the sense that they do not have distinct names or labels. They execute a protocol (the same for all agents) that specifies the computational and navigational steps. They are *asynchronous*, in the sense that every action they perform (computing, moving, etc.) takes a finite but otherwise unpredictable amount of time.

An agent sitting on a node v can read the labels on the edges incident to v. The agent can read and modify the contents of the whiteboard of node v as well as its own notebook memory, and it can leave node v through any incident edge $e(v, w)$. When an agent leaves a node v through an edge $e(v, w)$, the agent reaches node w within a finite amount of time and on reaching node w, it knows the label $\lambda_w(v, w)$ of the edge through which it arrived. In general, a mobile agent at any node v would repeatedly execute the following cycle of steps:

1. [READ] Read the contents of the whiteboard of node v to the agent's notebook.

2. [COMPUTE] Perform a sequence of computations modifying the contents of the agent's notebook.

3. [WRITE] Write to the whiteboard of node v, part of the results of the computation.

4. [MOVE] If *Next-Node*$= x > 0$, then leave the node v through the edge labelled x. Otherwise if *Next-Node*=0, remain at node v.

The main cost measure of an algorithm in this model is the number of moves (i.e. edge traversals) performed by all the agents combined, during the execution. This cost measure also indirectly reflects the time taken by an execution, if we assume unit delay for each edge traversal. In general we attempt to minimize this cost while designing our algorithms.

Another cost measure for this model is the memory requirement. There is a tradeoff between the memory available in the whiteboards and that available in the agents local notebook memory. In this thesis, we focus on minimizing the whiteboard memory whenever possible.

2.1.2 Whiteboards versus Tokens

The whiteboard model [55] is the most common model used for inter-agent communication in mobile agent systems. Whiteboards can be easily implemented in systems of networked processors, for example. There are some other more restrictive models but such models can be easily implemented using whiteboards.

One of the more restrictive model is the token model, first proposed in [80] based on the original idea of Baston and Gal [19]. In this model, agents use *tokens* (or pebbles) to mark the nodes. An agent that contains a token can place it on a node v before leaving the node; this token would be visible to any agent visiting node v, i.e. the visiting agent can determine whether or not there is a token at that node. If tokens are unmoveable, they may be placed only on the starting node of the agent. However some models use moveable tokens which can be moved and placed on a different node [79]. Sometimes there can be multiple distinct tokens, in this case, they are called markers [56].

Notice that tokens can be simulated by using 1-bit whiteboards, because putting a token at a node is equivalent to turning on a bit. However if the number of tokens available to an agent is limited, an agent may not leave tokens on multiple nodes at the same time. Thus, the token model is much more restrictive than the whiteboard model. In this thesis, we consider some solutions to the rendezvous problem using tokens only. The number of tokens present and whether they are moveable or not affects the effectiveness of the solutions.

2.1.3 Initial Knowledge

The amount of initial knowledge available to an agent, about its environment, often has a significant impact on the solvability of certain problems in our model. The agents may have some a priori information about the network, for instance they may know the topology or the size of the network or they may have a complete map of the network which can be used for navigation.

We would consider the following different kinds of initial knowledge about the network (G, λ, p) which may be available to the agent at start-up:

[NON] No prior knowledge of G,

[UPB] Knowledge of an upper bound on n, the size of G,

[EXS] Knowledge of the exact value of n, the size of G

[EXK] Knowledge of the exact value of k, the number of agents.

[S&K] Knowledge of the exact value of both n and k.

[MAP] Knowledge of a map (i.e. an isomorphic copy) of the labelled graph (G, λ).

[MPK] Knowledge of a map of the labelled graph with positions of the k homebases marked, i.e. (G, λ, p).

2.2 Other Models

Though there are many different models of distributed computation proposed in the literature, we shall restrict our discussion to the following two models discussed below (other than the mobile agent model discussed above).

2.2.1 Stationary Agent Model

The conventional model for distributed computation is that of a message-passing network of processors connected by point-to-point communication links. We call this the stationary agent(SA) model, which can be described as follows. The computing environment is modelled by a connected undirected graph $G(V, E)$. Each vertex of the graph G is associated with a fixed computational entity, called a *stationary* agent. Each vertex also contains a local memory which is accessible only to the agent associated with this vertex. Each agent can perform any number of computational steps, it can read and write to the local memory of the node and it can send messages and receive messages on each edge connected to the node. The agents are reactive entities, i.e. they react in response to external stimuli, e.g. the receipt of a message.

The state of an agent is defined by the contents of the local memory. Initially some of the agents would be in the special state "Initiator"; Such agents would start the computation process spontaneously. The initial value stored in the local memory of a node defines its identity. Thus, the nodes have distinct identities only if the initial contents of memory at each node is different.

The edges incident to a node are labelled by local orientation $\lambda = \{\lambda_v : v \in V\}$, where for each vertex u, $\lambda_u : \{(u,v) \in E : v \in V\} \to \{1, 2, \ldots, degree(u)\}$ defines the labelling on its incident edges. We shall use $\lambda(e)$ to denote the labels on the edge $e = (u,v)$ i.e. the pair $(\lambda_u(u,v),\ \lambda_v(u,v))$. The agent at node u can send a message m on any incident edge $e = (u,v)$ using the primitive SEND$(m, \lambda_u(e))$. In this case, the agent at node v receives the information $(m, \lambda_v(e))$, within a finite amount of time.

2.2.2 The Mobile Robots Model

This model is used mainly for problems related to navigating mechanical robots on a terrain. Thus, the environment here is a geometric plane (instead of a graph) and the agents (called robots in this case) can move in any direction on the plane. Sometimes the movement may be restricted by the presence of obstacles on the plane. In general, it is assumed that the robots have $360°$ vision, i.e. they can see in all directions and detect the presence of other robots or obstacles. The vision of a robot may be restricted to a circle of fixed radius around the robot or it may be unlimited. In both cases, the robots are assumed to be point objects so that one robot may not block the vision of another robot.

In this model, there are no whiteboards (or, tokens) for communication. It is generally assumed there is no explicit means of communication between the robots. Thus robots use movement to communicate information to one another.

There are two variations of the mobile robot model. In the first one [101], the robots act in synchronous steps (though not all robots act in every step) and all movements are instantaneous. The agents are also assumed to have unbounded memory. In the second and more weaker version [92], the robots are assumed to be memoryless and they operate in periodic LOOK-COMPUTE-MOVE cycles, where every step may take an unspecified amount of time.

A slight variation of this model is where the robots are moving on a graph instead of a plane [75]. In this case the movements of the robot are restricted to traversing the edges of the graph, but their vision provides them a snapshot of the whole graph.

2.3 Definitions and Problems

In this section we formally define the Rendezvous problem. We then show the relationship between the Rendezvous problem and some other well-known problems under the same setting.

For each of the problems that we consider, we represent an instance of the problem with the tuple (G, λ, p) where G is a connected undirected graph, λ is a local orientation on G, and $p : V(G) \to \{0, 1\}$ is a bi-coloring on the nodes of G. A *solution* to a given problem instance is a deterministic algorithm $\mathcal{P}$ (consisting of a finite set of instructions) such that when each agent individually executes the algorithm, on *termination* of the algorithm, the states of the agents and the contents of the whiteboard satisfy certain conditions (these are specific to the particular problem as described below). An agent terminates its individual execution when it reaches one of the designated *final* states at which point it is not required to perform any further computation.

Note that the termination of an algorithm can in general mean one of the following:

1. Local Termination: When an agent has terminated its individual execution of the algorithm.

2. Global Termination: When every agent has terminated its individual execution.

3. Global Termination with Detection: When every agent has terminated it s individual execution and is aware of the fact that every other agent has also terminated.

We say that the algorithm has *terminated* when every agent has locally terminated its execution. However, the knowledge of global termination may not be available to each individual agent. In other words, we do not require *global termination detection* for the problems that we consider in this thesis. We only require each agent to terminate its local execution and be aware of the fact it has terminated. (This can be called *local termination detection.*)

Since we are considering a totaly asynchronous system, the order in which the actions performed by different agents occur could be arbitrary (we can assume that an adversary schedules these events). Each possible ordering of these actions is defined as an *execution* of the algorithm $\mathcal{P}$.

The algorithm $\mathcal{P}$ is said to solve a given instance (G, λ, p) of a problem if every possible execution of the algorithm on the network (G, λ, p), solves the problem.

We say that an algorithm is a *terminating* algorithm if the execution of the algorithm on any network (G, λ, p) terminates after a finite time (whether succeeding in solving the problem or not).

Solvable Instance: A given instance (G, λ, b) of a problem is said to be *solvable*, if there exists a deterministic (distributed) algorithm $\mathcal{P}$ such that every execution of the algorithm $\mathcal{P}$ on that instance, solves the problem within some finite time.

Effective Algorithm: A deterministic algorithm $\mathcal{P}$ for a given problem, is said to be *effective* if every execution of $\mathcal{P}$ on every instance of the problem detects whether the instance is solvable,

and terminates in finite time, succeeding in solving the problem whenever the given instance is solvable.

In other words, an *effective* algorithm solves every solvable instance of the problem and is a terminating algorithm.
We define the Rendezvous problem as follows:

Definition 2.3.1 *The* Rendezvous*(RV) problem (or, gathering of all agents together in one node): A given instance (G, λ, p) of the Rendezvous problem is said to have been solved when all the k agents are located in a single node of G.*

Note that the solution to any problem (including Rendezvous) in the mobile agent model, usually requires the agents to traverse the graph and obtain information about its environment. In fact, if each agent is able to obtain a map of its environment, then it is easy to solve many problems. We define below the problem of constructing a unique map of the environment.

Definition 2.3.2 *The* Labelled Map Construction *(LMC) problem: A given instance (G, λ, p) of the LMC problem is said to have been solved when each agent obtains a* labelled *map of the graph, (with the position of this agent marked in it) such that the label assigned to any particular node is the same in all the maps.*

The following problems are also related to solving *Rendezvous*:

Definition 2.3.3 *The* Agent Election *(AEP) problem (electing a leader among the agents): A given instance (G, λ, p) of the AEP problem is said to have been solved when exactly one of the k agents is in the final state 'LEADER' and all other agents are in the final state 'FOLLOWER'.*

The problem of electing a leader is quite well-known for the stationary agent (i.e. message-passing) systems; In that case, we call it the *Node Election* problem to distinguish it from the problem of election among mobile agents.

Definition 2.3.4 *The* Labelling *problem (assigning unique labels to the nodes of an unlabeled graph) : A given instance (G, λ, p) of the* Labelling *problem is said to have been solved when the whiteboard of each node is marked with a label and no two nodes have the same label.*

Definition 2.3.5 *The* Spanning Tree Construction*(SPT) problem (constructing a spanning tree of the graph) : A given instance (G, λ, p) of SPT problem is said to have been solved if each edge of the graph G is marked as either a Tree-edge or Non-Tree edge, such that the set of Tree-edges, T represents a spanning tree of the graph G.*

We now discuss the relationship among the above problems.

Theorem 2.3.1 *The following problems are computationally equivalent such that for any given instance (G, λ, p) either all of them are solvable or none of them are solvable: (i) Labelled Map Construction, (ii) Agent Election, (iii) Labelling, and (iv) Rendezvous.*

Proof : *LMC => RV*: Once the LMC problem has been solved, *Rendezvous* can be solved too. When each agent has a uniquely labelled map of the graph, the agents simply move to the node having smallest label in the map, thus solving *Rendezvous.*
RV => AEP: Once the agents rendezvous at a single node v, the mutual exclusion property of the whiteboards allows us to break the symmetry among the agents and elect a leader. The agent that first writes to the whiteboard of node v becomes 'LEADER', while all other agents become 'FOLLOWER'.
AEP => LBL: Once a leader agent is elected, this agent can explore the graph, assigning (i.e., writing) unique labels to the nodes, while the other agents remain stationary in their homebases.
LBL => LMC: Once the graph is labelled, each agent can execute a depth-first traversal of the labelled graph, to obtain a uniquely labelled map of the graph. ■

Hence, the problems of *Labelling*, *Rendezvous*, *Agent Election* and *Labelled Map Construction* are computationally equivalent in our model. This relationship among the problems is useful in determining some conditions for solvability of the rendezvous problem, to be presented in later chapters.

Chapter 3

Review of Known Results

3.1 Solutions for the Mobile Agent Model

3.1.1 Agent Rendezvous

The *Rendezvous* problem was first proposed by Steven Alpern [3] in a seminar he presented in 1976. Since then he and other researchers have provided various solutions to the problem under different scenarios, assuming the search domain to be either a geometric space (such as a line [7], circle [72], plane [8] etc.) or a graph [5]. In most of these cases, the problem was studied only for two agents and the objective was to find a search strategy that minimized the expected time before the two agents met. An extensive survey of such results (mostly using randomization) can be found in [6]. In this thesis, we are concerned with only deterministic solutions to the problem and this requires us to follow a different approach.

Among deterministic solutions to rendezvous, Yu and Yung [106] proposed an algorithm for gathering multiple agents in synchronous graphs assuming that the topology is known to the agents. Dessmark *et al.* [48] showed how two agents can meet in synchronous rings or arbitrary graphs, when they have *distinct* identities. This result was later improved in [77] to obtain a polynomial solution. A solution to rendezvous in the asynchronous graphs was provided by De Marco et al. in [83]. All the above results are for agents having distinct labels and but having no ability to mark the nodes of the graph.

The idea of performing rendezvous search using marks on the starting places was explored first by Baston and Gal[19]. Notice that if marking of nodes is permitted then it may be possible to achieve rendezvous of anonymous agents in unlabelled graphs. Kranakis *et al.* [80] and Flocchini *et al.*[59] gave solutions for the rendezvous of two (resp. multiple) agents, on a ring networks using unmovable pebbles (called tokens) to mark the starting nodes. Gasieniec et al.[69] gave an optimal memory solution for the same setting. Kranakis et al.[79] studied the problem for a synchronous torus using both fixed and movable tokens.

Using whiteboards for marking the nodes, Barrière *et al.* achieved rendezvous of multiple agents on arbitrary unlabelled graphs with *sense of direction* [17]. As mentioned in Chapter 2, the rendezvous problem is equivalent to the agent election problem, in presence of whiteboards. In [16], Barrière *et al.* consider solutions to the agent election problem in presence of distinct but incomparable agent labels. This result was improved by Chalopin [30] who gave an *effective* solution for this model.

Among fault-tolerant solutions, Flocchini et al.[58] considered the case when tokens fail, i.e. when the tokens suddenly disappear. Dobrev et al.[52] solved the rendezvous and near-gathering problems in the presence of a dangerous node (*black hole*). Both these results are for the ring networks only. In this thesis, we present the generalized solutions for both.

3.1.2 Graph Traversal and Exploration

When the nodes of the graph have distinct identifiers, then the problems of traversal and exploration are equivalent, (i.e. if the agent succeeds in traversing every edge of the graph, it can obtain a map of the graph). Previous studies on the exploration of such graphs (or digraphs), have concentrated on analyzing the cost of exploration in terms of the total number of edge traversals (see for example [2, 46, 47, 89]). In this case, a simple depth-first traversal is optimal in terms of number of moves (i.e. making $\Theta(m)$ edge traversals) while a modified version given by Panaite and Pelc [89] takes $m+O(n)$ moves. Awerbuch *et al.* [13] have studied the problem of piece-meal exploration, where the robot exploring the graph has to periodically return to its homebase (e.g. for refuelling), during the exploration.

A more challenging problem is to traverse an anonymous graph (i.e. where the nodes are unlabelled). In this case, traversing the graph is not equivalent to mapping the graph. The knowledge of the size n (or the diameter D) of the graph is a necessary and sufficient condition for achieving traversal (with termination). But solving the mapping problem in general, requires that the robots have some means of marking the nodes (except for some special graphs e.g. trees). Different models for marking the nodes have been used by different authors. Bender *et al.* [20] used the method of dropping a pebble on a node to mark it and showed that any strongly connected directed graph can be explored using just one pebble if the size of the graph is known, and using $O(\log\log n)$ pebbles, otherwise. Dudek *et al.* [56] used a set of distinct markers to explore unlabelled undirected graphs. Yet another approach, used by Bender and Slonim [21] was to employ two cooperating agents, one of which would stand on a node, thus marking it, while the other explores new edges. Another much stronger model is the whiteboard model (used in [63, 62]) where the robot can write on public whiteboards available at each node. With unbounded whiteboard memory, single agent exploration(and mapping) is again reduced to performing a traversal.

Most of the known results on exploration are for single agent exploration with the exception

of [21](which uses two agents as mentioned above) and [62] where multiple agents are used to explore a tree. In both these cases, the agents are co-located (i.e. they start from the same location). Notice that when the nodes have to be marked in order to map the graph, the distributed exploration by multiple agents is more difficult than single agent exploration ([41]). However the distributed exploration can be reduced to single agent exploration if we can gather the robots together in one node (*Rendezvous*) or elect a leader among the agents (*Leader Election*).

There has been several studies on graph traversals when the memory available to an agent (i.e. its ability to remember) is limited. The agent is often modelled as a finite automaton and the emphasis is on minimizing the number of states of the automaton. Fraigniaud *et al.* [64] show that $\Theta(D \log \Delta)$ memory is sufficient for traversing graphs of diameter D and maximum degree Δ. They also show that at least $\log(n)$ bits of memory is necessary for traversing a graph of size n. In [65], the authors show that k non-cooperating agents each having $\log(Q)$ memory cannot traverse all graphs of size $k \cdot Q$. There are many previous results on exploration of mazes by one or more finite automata (see [78, 86, 94, 97]). However, exploring mazes is much more easier in general, than exploring graphs, due to the presence of sense of direction, as was shown in [24].

Another related problem in that of *perpetual traversal*, where the agent is not required to halt after it has visited all the nodes. A perpetual traversal algorithm must ensure that every node is visited within a finite amount of time. It was shown in [49] that $\log(\Delta)$ bits of memory are sufficient for perpetual traversal of trees of maximum degree Δ.

3.1.3 Capturing an Intruder (Graph Search)

The problem of intruder-capture (or pursuit-evasion) is one where there are two parties with conflicting interests—a team of searchers and an intruder; The objective of the searchers is to locate and capture the intruder while the objective of the intruder is to evade capture. This problem was first studied in the graph setting by Parsons [90]. Several variants of the problem have been studied with respect to different applications such as detecting intruders in computer networks [11], decontaminating a network of tunnels, capturing a criminal hiding in a labyrinth and so on. The problem is often called Graph-search and the minimum number of searchers required to clean a graph is called the search number of the graph. Megiddo et al. [85] showed that finding the search number of a graph is NP-hard. Bienstock and Seymour [22] studied the monotone version of the problem (when recontamination is not allowed), while LaPaugh [82] was the first to show that recontamination does not help in decreasing the number of searchers required for searching. Barrière et al.[15, 18] introduced and studied the connected version of graph searching, where the decontaminated part of the graph must always remain connected. Recently, Blin et al. [23] have given a distributed algorithm for constructing a

monotone connected search strategy in an unknown network.

3.1.4 Locating Faults (Black Holes)

In the context of mobile agent systems, the only type of fault that has been studied previously is the *black hole*, which is a harmful network site that destroys any visiting agent. The problem of locating a black hole using mobile agents was first studied by Dobrev et al. [53]. Most of the studies on this problem has faulty networks have concentrated on locating the black hole. In asynchronous systems, this has been studied under two different methods—using whiteboards [53, 51] or using tokens [50, 54] to mark edges. The objective here is minimizing the number of agents that fall into the black hole and the number of moves. In the case of synchronous agents, the objective is to minimize the time taken by the surviving agents to locate the black hole [38, 39, 76]. The general case of multiple black holes has been considered only by Cooper et al. [36]. All these problems assume that the team of agents start from the same node, i.e. they are co-located. When the agents start from distinct nodes, it is more difficult to detect the location of the black hole nodes. The case of scattered agents has been considered in the Ph.D. thesis of Wei Shi [99]. The problem of gathering the scattered agents in the presence of a black hole, has been studied earlier (only in the case of ring networks) by Dobrev et al. [52], assuming the knowledge of topology and the size of the network. In this thesis, we generalize these results to networks of arbitrary topology with faults in multiple locations, i.e. both faulty nodes and edges.

3.2 Solutions for the Stationary Agent Model

3.2.1 Leader Election

The leader election problem has been extensively studied mostly under the assumption that nodes of the network have distinct identities. Angluin [10] was the first to study the problem in anonymous networks (she called it the problem of establishing a "center"). This work was later extended by Johnson and Schneider [73] and then by Yamashita and Kameda [104, 105] who gave a characterization of graphs based on solvability of the leader election problem under different communication models. Boldi *et al.* [25, 27] used the concepts of graph fibrations [28] and coverings to characterize labelled networks where election is solvable. Boldi and Vigna [26] and Yamashita and Kameda [103] have also studied the problem of general computability in directed and undirected graphs respectively.

Korach, Kutten and Moran [57] showed that the leader election problem is closely related to graph exploration and they proposed the territory acquisition approach for electing a leader in arbitrary graphs. (Their algorithm uses the token-passing model and assumes unique identities

for the nodes of the graph.)

3.2.2 Spanning Tree Construction

The problem of leader election is also related to the problem of distributed spanning tree construction in a network. Many distributed algorithms have been proposed for minimum spanning tree construction in labelled graphs, notably the ones proposed by Gallager, Humblet and Spira [67] and by Awerbuch[12]. For anonymous networks, Sakamoto[95] gave an algorithm that builds a spanning forest of the graph under a variety of initial conditions.

3.2.3 Distributed Enumeration or Labelling

The problem of distributed enumeration problem, i.e. for numbering the nodes of an undirected graph G with integers from 1 to $|V(G)|$, was considered by Mazurkiewicz [84] who gave an enumeration algorithm (in the *local-computation* model). They showed that it is possible to do this only when the graph G is "unambiguous". Godard et al. [70] translated this property in terms of coverings of simple graphs. Chalopin and Métivier [33] later adapted the Mazurkiewics algorithm to the message passing model and showed that the enumeration problem is solvable in a symmetric directed graph G, if and only if G is *symmetric-covering-prime*.

The problem of assigning distinct labels to the nodes of a graph was studied by Fraigniaud *et al.*[91] assuming the presence of a designated leader node. They concentrated on the problem of assigning short labels to the nodes in an efficient way.

3.3 Solutions for the Mobile Robots Model

The problem of distributed coordination among multiple mobile robots moving in a terrain, has been studied by many authors mainly from an engineering point of view (see for example [14, 29, 74]). From an algorithmic point of view, the problem of pattern formation by robots in a plane has been studied by Sugihara et al. [100] and by Suzuki et al. [101] among others. The point formation or gathering problem has been specifically studied by Ando et al. [9] and Agmon et al.[1]. These results are for robots which can move instantaneously. Under a different model based on the LOOK-COMPUTE-MOVE cycle, the robot gathering problem was studied by Cielebak et al.[35], Flocchini et al.[60] and Prencipe [93]. Some of these results assumed that the robots can see everything, while others have restricted the visibility of the robots to a circle of fixed radius around the robot. Most results on gathering neglect the size of the robots, i.e. they assume the robots to be point objects such that two robots may occupy the same position in a plane. An exception was the result by Czyzowicz et al.[37] who considered robots

having non-zero size (i.e. fat robots). The gathering of robots in a graph (more specifically a ring) instead of a plane was studied recently by Klasing et al. in [75].

Chapter 4

Basic Techniques for Computing in Anonymous Networks

4.1 Computing the View of an agent

The concept of the *view* of a node in a network, was introduced by Yamashita and Kameda [104, 105] with respect to the symmetry-breaking or the leader election problem in the S.A. model. Intuitively, the view of a node v contains all the information about the network, that may be obtained by the node v using message-passing. The *view* of a node v in the network represented by (G, L, λ), is an infinite rooted tree with edge labels, that contains all (infinite) walks starting at v. For the Mobile Agent model, we extend the concept of view to bi-colored views, where the vertices in the view are colored black or white depending on whether or not they represent the homebase of some agent. The view of an agent is taken to be the bi-colored view of its homebase.

Definition 4.1.1 *Given the distributed environment represented by* (G, λ, p)*, the* view $T_v(G, \lambda, p)$ *of node* v*, in this environment is an infinite edge-labelled rooted tree* T*, whose root represents the node* v *and for each neighboring node* u_i *of* v*, there is a vertex* x_i *in* T*(with same color as* u_i*) and an edge from the root to* x_i *with the same labels as the edge from* v *to* u_i *in* G*. The subtree of* T *rooted at* x_i *is again the view* $T_{u_i}(G, \lambda, p)$ *of the node* u_i*.*

The closed view is obtained from the view by merging the paths in the view such that multiple occurrences of the same vertex (or the same edge) coincide together. The closed view is also called the *Surrounding* (introduced in [61]) and it formally defined as follows.

Definition 4.1.2 *The surrounding* $T_v^*(G, \lambda, p)$ *of a node* v *in the network* (G, λ, p)*, is a labelled graph* S *with* n *vertices* $(x_1, x_2, ..., x_n)$ *where* x_1 *corresponds to node* v *and is labelled* ϕ*; for each*

$1 < i \leq n$, x_i corresponds to some node v_i of G and is labelled with the sequence of edge-labels in a shortest path from v to v_i in G. For each edge from node v_i to v_j in G, there is an edge from x_i to x_j in S with the same labelling.

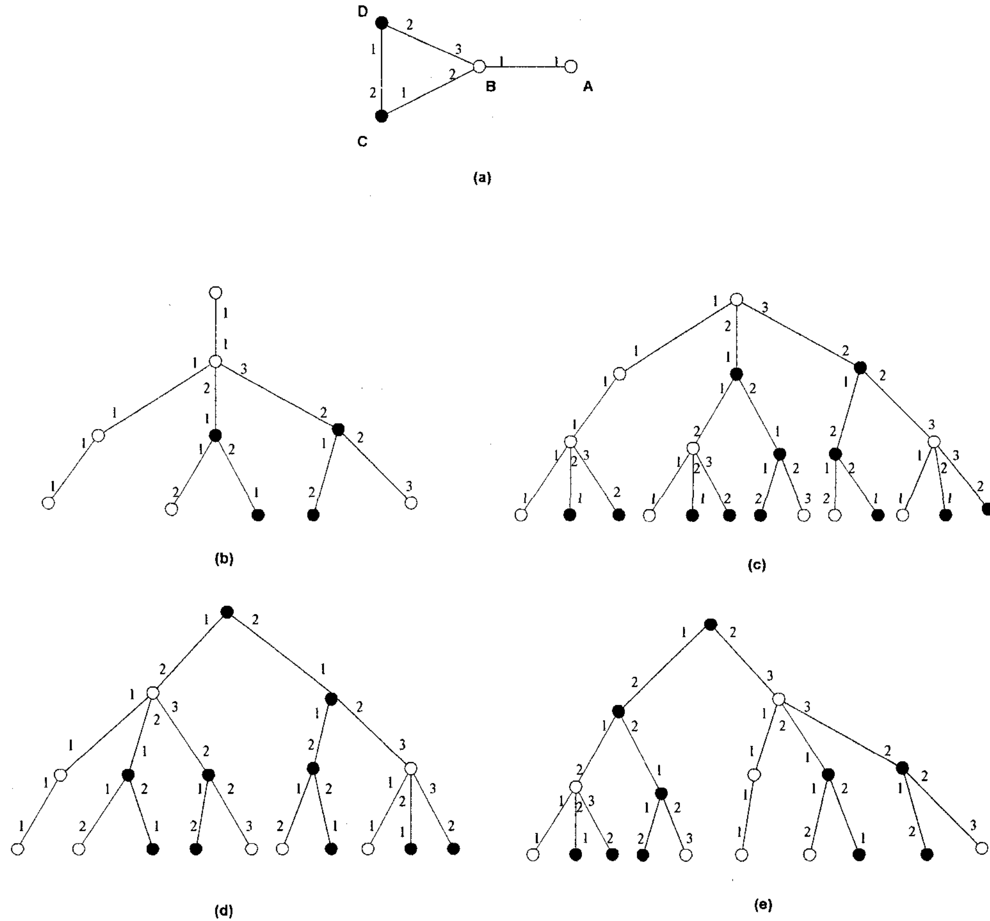

Figure 4.1: The view of a node: A network of $n = 4$ nodes is shown in (a), and the views of the nodes A, B, C, and D, truncated to a depth of $n - 1$, are shown in (b), (c), (d) and (e) respectively.

Note that each vertex in the surrounding is uniquely labelled and only the vertex corresponding to node v is labelled by an empty sequence. Intuitively speaking, the surrounding of a node v can be considered to be a map of the graph G, that is oriented with respect to v.

The view from node u, truncated to a depth of h is denoted by T_u^h. An interesting observation is that the view T_u^h from node u, contains the view T_v^{h-d} for all such nodes v that are a distance of $d < h$ from node u. Thus, the view up to depth $2n$ from any node contains as sub-views, the views up to depth n, of all other nodes.

Property 4.1.1 ([87, 104, 105]) *The views $T_u(G, \lambda, p)$ and $T_v(G, \lambda, p)$ of any two nodes u and v in a graph G of size n, are equal if and only if the truncated view up to depth $n-1$, of these two nodes are equal, i.e. $T_u(G, \lambda, p) = T_v(G, \lambda, p)$ if and only if $T_u^{n-1}(G, \lambda, p) = T_v^{n-1}(G, \lambda, p)$.*

In [104], Yamashita and Kameda showed that if we partition the nodes of a graph into equivalence classes such that all nodes in a class have the same view, the sizes of the equivalence classes thus obtained would be same. They defined the *Symmetricity* of a graph as the cardinality of the equivalence classes containing nodes having identical views. We extend this definition to closed-views (surroundings).

Definition 4.1.3 *The* Symmetricity *of an edge-labelled and bi-colored graph (G, λ, p) is defined as $\sigma(G, \lambda, p) = |\{v : T_v(G, \lambda, p) = T_u(G, \lambda, p)\}|$ for some arbitrary $u \in V$.*

Definition 4.1.4 *The* Closed-Symmetricity, *of an edge-labelled and bi-colored graph (G, λ, p) is defined as $\sigma^*(G, \lambda, p) = |\{v : T_v^*(G, \lambda, p) = S\}|$ where S is the* surrounding *of some arbitrary node $u \in V$.*

Lemma 4.1.1 *Given any positive integer d, an agent A located at a node v of a bi-colored edge-labelled graph (G, λ, p), can construct the truncated view $T_v^d(G, \lambda, p)$ of node v, regardless of the presence of other (possibly identical) agents.*

Proof : To construct the view, agent A simply does a depth first traversal of the graph G, up to depth of d, starting from v. During the traversal it remembers the labels of edges traversed and adds them to the view that it constructs. When constructing the view in this way, an agent may have to make a maximum of $2.\Delta^d$ moves, where Δ is the maximum degree of a node in G. Here, the agent does not require the use of whiteboards or any other marking device, when constructing the view. ■

Lemma 4.1.2 *Given a bi-colored edge-labelled graph (G, λ, p), each agent can construct the bi-colored surrounding $T_v^*(G, \lambda, p)$ of any node v, if the size n of G is known and Symmetricity of (G, λ, p) is equal to 1.*

Proof : Due to Lemma 4.1.1, an agent can construct the view $T_v^{2n}(G, \lambda, p)$ up to depth $2n$. Since the symmetricity is equal to 1, each node in G can be uniquely identified by looking at its view up to depth n. Thus, the agent can identify the multiple occurrences of any node in the view and hence it can obtain the bi-colored surrounding $T_v^*(G, \lambda, p)$. ■

4.2 Constructing the minimum base

When the symmetricity of the network is greater than 1, then the agents cannot construct a map of the graph G, even if they know the size of the network or the number of agents. In this case, there exists a label-preserving automorphism on the labelled graph (G, λ, p). In this section, we show how the agents can construct the *minimum-base* of such a network. Intuitively, the minimum-base of a network is the smallest labelled digraph which represents a family of graphs that are all similar to (G, λ, p). The concept of minimum-base was introduced by Boldi and Vigna[28] for computation in anonymous networks and it is based on the notion of graph coverings. We explain these concepts below.

4.2.1 Symmetric Directed Graphs

We define a *digraph* D as the tuple $(V(D), A(D), s_D, t_D)$, where V(D) is a set of vertices, $A(D)$ is a set of arcs, and, s_D and t_D assign to each arc in $A(D)$, two elements of $V(D)$: a source and a target (in general, the subscripts may be omitted). Notice that we allow a digraph to have self-loops (i.e. arc whose source and target are same) as well as parallel arcs (i.e. two arcs sharing the same source and the same target). A digraph D is called *strongly connected* if for all vertices $u, v \in V(D)$, there exists a path from u to v. A *symmetric* digraph D is a digraph endowed with a symmetry, that is, an involution $Sym : A(D) \to A(D)$ such that for every $a \in A(D), s(a) = t(Sym(a))$. A bidirectional network can be represented by a strongly connected symmetric digraph, where each edge of the network is represented by a pair of symmetric arcs. In particular, we consider digraphs D where the vertices and the arcs are labelled by symbols from the recursive set L and $\mu_D \colon V(D) \cup A(D) \to L$ is the labelling function. The label on any arc a should be a pair (x, y) and the labelling μ_D should satisfy the property that if $\mu_D(a) = (x, y)$ then $\mu_D(Sym(a)) = (y, x)$, for every arc $a \in D$.

A self-loop in the digraph D is an arc a, such that $s(a) = t(a)$. A self loop is called symmetric if $Sym(a) = a$ and otherwise it is asymmetric. Notice that for a symmetric self-loop a, $\mu_D(a)$ is of the form (x, x) but for asymmetric self-loop a', $\mu_D(a')$ is of the form (x', y') with $x' \neq y'$. A pair of arcs a and a' are called parallel, if $s(a) = s(a')$ and $t(a) = t(a')$.

A digraph homomorphism γ between the digraph D and the digraph D' is a mapping $\gamma \colon V(D) \cup A(D) \to V(D') \cup A(D')$ such that if u, v are vertices of D and a is an arc such that $u = s(a)$ and $v = t(a)$ then $\gamma(u) = s(\gamma(a))$ and $\gamma(v) = t(\gamma(a))$. A homomorphism from (D, μ_D) to (D', μ'_D) is a digraph homomorphism from D to D' which preserves the labelling, i.e., such that $\mu'_D(\gamma(x)) = \mu_D(x)$ for every $x \in V(D) \cup A(D)$.

We represent the network (G, λ, p) by a labelled digraph (D, μ_D), such that (i) $V(D) = V(G)$, (ii) $\mu_D(v) = p(v)$, $\forall v \in V(G)$ and (iii) for each edge $e = (u, v)$ of G there are two arcs $a_1 = (u, v)$ and $a_2 = (v, u)$ in $A(D)$, where $\mu_D(a_1) = (\lambda_u(e), \lambda_v(e))$ and $\mu_D(a_2) = (\lambda_v(e), \lambda_u(e))$.

4.2.2 Digraph Coverings and their Properties

We now define the notion of graph coverings, based on the terminology used in [28]. A *covering projection* is a homomorphism φ from D to D' satisfying the following: (i) For each arc a' of $A(D')$ and for each vertex v of $V(D)$ such that $\varphi(v) = v' = t(a')$ there exists a unique arc a in $A(D)$ such that $t(a) = v$ and $\varphi(a) = a'$. (ii) For each arc a' of $A(D')$ and for each vertex v of $V(D)$ such that $\varphi(v) = v' = s(a')$ there exists a unique arc a in $A(D)$ such that $s(a) = v$ and $\varphi(a) = a'$.

If a covering projection $\varphi : D \to D'$ exists, D is said to be a *covering* of D' via φ and D' is called the base of φ. A symmetric digraph D is a *symmetric covering* of a symmetric digraph D' via a homomorphism φ if D is a covering of D' via φ such that $\forall a \in A(D), \varphi(Sym(a)) = Sym(\varphi(a))$. A digraph D is *symmetric-covering-minimal* if there does not exist any graph D' not isomorphic to D such that D is a symmetric covering of D'.

Property 4.2.1 ([28]) *Given two non-empty strongly connected digraphs D, D', each covering projection φ from D to D' is surjective and $\forall v' \in D'$, $varphi^{-1}(v')$ has the same cardinality. This cardinality is called the number of sheets of the covering.*

The notions of coverings extend to labelled digraphs in an obvious way: the homomorphisms must preserve the labelling. Given a labelled symmetric digraph (H, μ_H), the *minimum base* of (H, μ_H) is defined to be the labelled digraph (D, μ_D) such that (i) (H, μ_H) is a symmetric covering of (D, μ_D) and (ii) (D, μ_D) is symmetric covering minimal.

Property 4.2.2 ([28]) *If (H, μ_H) is a covering of (D, μ_D) via φ, then any execution of an algorithm $\mathcal{P}$ on (D, μ_D) can be lifted up to an execution on (H, μ_H), such that at the end of the execution, for any $v \in V(H)$, v would be in the same state as $\varphi(v)$.*

As mentioned before, a network (G, λ, p) can be represented by a labelled digraph (H, μ_H). If (D, μ_D) is the minimum-base of (H, μ_H), then we say that (D, μ_D) is the minimum-base of the network (G, λ, p). The minimum-base of a network can be constructed from the view $T^{2n}(v)$ from any node v (truncated to depth $2n$), where n is the size of the network. To construct the minimum-base, first every edge in the view should be replaced by two opposite arcs with symmetric labelling, as mentioned before. Those nodes which have the same view should be merged into a single vertex. Any two parallel arcs with the same labels should be merged into a single arc. The labelled digraph thus obtained would be the minimum-base of the network.

4.3 Distributed Traversal Algorithm

Most of the problems that we consider require the agents to traverse the network (for example to gather information or to meet other agents). In this section we look at a generic algorithm

for the traversal of an arbitrary graph by multiple agents. The traversal algorithm given in this section would be used as a preliminary step in our protocols for solving some other problems.

We want the agents to explore the graph collectively, in such a way that the total number of edge traversals is minimized. Each agent can traverse an area around its homebase, while avoiding the parts being explored by the other agents. During the exploration, the agent needs to remember the path to its homebase, so that it does not get lost. Each agent stores in its memory the sequence of labels (in order) of edges traversed by it, starting from the homebase. We call this sequence of labels as the *Exploration Path.* When the edge $e = (u, v)$ is traversed by the agent from u to v, then the label $l_v(e)$ is appended to the *Path.* This enables the agent to return back to the previously visited node (i.e. u) whenever it wants to. When it does so, the agent is said to have backtracked the edge e and the label $l_v(e)$ is deleted from the *Path.* So, at all times during the traversal, the *Path* contains the sequence of labels of the links that an agent has to traverse (in reverse order) to return to its homebase from the current node.

Each agent on wake-up, starts traversing the graph, from the homebase and it marks the visited nodes if they are previously unmarked[1]. The agent also builds a partial *Map* of the territory that it marks. The edges are marked as 'T' or 'NT', where 'T' edges belong to the territory and are included in the Map, whereas, 'NT' edges are not included in the Map. The algorithm executed by each agent is the following:

Algorithm EXPLORE

1. Set *Path* to empty ;
   Initialize the *Map* as single-node graph consisting of the homebase;

2. While there is another unexplored edge e at the current node u,
   - mark link $l_u(e)$ as 'T' and then traverse e to reach node v;
   - If v is already marked,
     - return back to u and re-mark the link $l_u(e)$ as 'NT';
   - Otherwise
     - mark the link $l_v(e)$ as 'T', and mark v as explored;
     - Add link $l_v(e)$ to *Path*;
     - Add the edge e to *Map*;

3. When there are no more unexplored edges at the current node,
   If Path is not empty then,
   - remove the last link from Path, traverse that link and repeat Step 2;

   Otherwise, Stop and return the *Map*;

We make the following observations about the effects of this algorithm.

[1] Recall that the homebases are already marked.

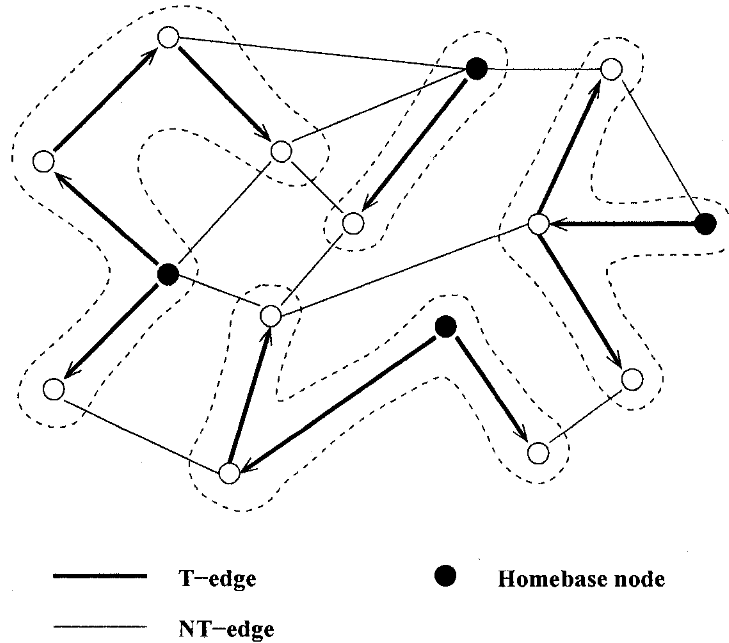

Figure 4.2: After executing algorithm EXPLORE, each agent obtains a disjoint territory.

Lemma 4.3.1 *On termination of the algorithm EXPLORE,*

(a) *If a node has been marked by an agent A, every edge incident that node would have been traversed by agent A.*

(b) *Each node in the graph would have been marked by exactly one agent.*

(c) *If two nodes are marked by the same agent then there exists a consecutive sequence (i.e. a path) of 'T' edges joining them. On the other hand, if two nodes are marked by two different agents, then every path joining them would contain at least one 'NT' edge.*

(d) *There is no cycle consisting of only 'T' edges.*

Proof : Part(a): During the algorithm EXPLORE, if an agent A marks a node u, then each link at the node u is either marked by A, or remains unmarked. (Thus even if an edge $e = (u, v)$ is traversed from v to u by another agent B, the link $l_u(e)$ at u, could be marked only by agent A.) Whenever there are unexplored (i.e. unmarked) links at the current node u, agent A would traverse one of the unexplored edges and whenever an agent leaves a node u through an unexplored link, it would eventually return to u at some time during the traversal. Thus, each link incident to u would be traversed by A before it stops.

Part(b): From part(a), we know that an agent visits all neighbors of a node that it marks.

Also, we know that whenever an agent visits any unmarked node it marks it. To begin with, an agent always marks its homebase, and whenever a node is marked all its neighbors would also be marked eventually. Thus, each node would be marked by some agent. Note that due to the mutual exclusion property of whiteboards, a node cannot be marked by more than one agent.

Part(c): This follows immediately from the algorithm.

Part(d): A 'T' edge is marked whenever it is traversed for the first time. If $e = (u, v)$ is a 'T' edge which was initially traversed in the direction from from u to v, then it is always true that u was marked earlier than v, and e was marked after u but before v. This precedence relationship implies that the 'T' edges cannot form a cycle. ■

Lemma 4.3.2 *The total number of edge traversals made by the agents in executing algorithm EXPLORE, is at most* $4m$*, irrespective of the number of agents.*

Proof : During the algorithm EXPLORE, each 'T' edge is traversed twice—once in the forward direction and once while backtracking. Each 'NT' edge is traversed four times, twice from each side. As there are $(n - k)$ 'T' edges, the total number of moves (i.e. edge traversals) made by the agents is $(4m - 2n + 2k)$. ■

Note that for this algorithm, we do not require much memory at the whiteboards of the nodes. In fact one bit per whiteboard is sufficient—for marking the node as explored.

When an agent A finishes executing algorithm EXPLORE, A would have obtained a map of the territory marked by it. This territory is a tree consisting of all the nodes marked by it and the 'T' edges connecting these nodes; the edges leaving the territory are marked 'NT'. The territories marked by different agents are all disjoint, and together they span the whole graph. So, the distributed traversal of the graph by multiple agents creates a spanning forest of the graph. Two adjacent trees in the forest would be connected through one or more 'NT' edges, each joining a node of one tree to that of another.

Notice that some of the 'NT' edges could possibly be connecting nodes of the same tree (i.e. they are *back edges*) while the other 'NT' edges connect nodes from two distinct trees. The agents cannot, in general, distinguish between these two types of 'NT' edges and from the point of view of an agent, all the 'NT' edges, it encounters, seem to be going out of its territory. Thus, the maps constructed by algorithm EXPLORE provide an incomplete view of the graph to the agents and constructing the complete map of the graph does not simply involve joining these partial maps together.

Chapter 5

Rendezvous in the Token Model

In this chapter we study mobile agent computations in the token model (i.e. when whiteboards are not present). Recall that the token model is much more restrictive than the whiteboard model. Each agent has only a single token which can be placed on any node of the network, and all tokens are identical. We show how to solve rendezvous in this model.

5.1 Solving Rendezvous in the Token Model

As mentioned in Chapter 2, any solution to the leader election problem can be used for solving rendezvous. Yamashita and Kameda [104] have determined the exact conditions for solving leader election in the message-passing model (S.A. model). Barriere *et al.* observed in [17] that any mobile agent algorithm can be simulated in the S.A. model when each node v in the network is provided with the value $p(v)$. Based on these results, we obtain the following results:

Lemma 5.1.1 *When the agents have no knowledge of the network topology (except its size), the rendezvous problem is solvable in an arbitrary network* (G, λ, p) *only if the symmetricity* $\sigma(G, \lambda, p) = 1$ *(i.e. if each node has a unique bi-colored view).*

Lemma 5.1.2 *When the agents have a map of the network, the rendezvous problem is solvable in an arbitrary network* (G, λ, p) *only if the closed-symmetricity* $\sigma^*(G, \lambda, p) = 1$ *(i.e. if each node has a unique bi-colored surrounding).*

We can use a simple strategy for solving rendezvous in the token model. Each agent can traverse the graph to compute its view up to a depth of $2n - 1$ and thus obtain the views (up to depth $n - 1$) of all other agents. Since there exists a total order on views (of the same depth), it is possible to order the agents based on their views and thus elect a leader among them. The leader agent would simply wait at its homebase and all other agents would move

to the homebase of the leader to solve rendezvous. The following algorithm implements such a strategy. Since we construct the bi-colored view, each homebase needs to be marked. Each agent can mark its homebase using a token (we call a node that is marked with a token, a *black* node). The following algorithm can be executed using tokens only (without using whiteboards).

Algorithm *Compare-View*

Put token on homebase;
$h \leftarrow 2n - 1$;
Call Construct-View(h) to get the bi-colored view T_u^h of the homebase u;
$i \leftarrow 1$; $A[i] \leftarrow T_u^{n-1}$;
for each vertex v in T_u^h, which is at level less than n, **do**
  if $T_v^{n-1} \neq A[j]$ for any $1 \leq j \leq i$, **then**
    $i \leftarrow i + 1$; $A[i] \leftarrow T_v^{n-1}$;
if $i \neq n$, **then** terminate with failure;
Sort the array A based on a fixed total order (assuming black vertices precede white ones);
if $A[1] = T_u^{n-1}$, **then** become LEADER;
else go to nearest node v such that $A[1] = T_v^{n-1}$;

Procedure *Construct-View*(h)

(To construct the view up to level h, for current node u)
Add the current node u to T_u^h,
if $h = 0$, **then** return T_u^h;
else,
  for $i \leftarrow 1$ to degree(u), **do**
    traverse link i to reach node v_i;
    add the traversed edge and its labels to T_u^h;
    $T_{v_i}^{h-1} \leftarrow$ Construct-View($h - 1$); add $T_{v_i}^{h-1}$ to T_u^h;
  Return T_u^h;

Theorem 5.1.1 *The algorithm* Compare-View *is an* effective *algorithm for rendezvous.*

Proof : If the rendezvous problem is solvable for a given instance, then the view T_v^{n-1} for each homebase v would be unique (by Lemma 5.1.1 and Property 4.1.1). So, the agent whose bi-colored view is first in the lexicographical order would be elected as LEADER by the algorithm *Compare-View* and all other agents would reach the LEADER's homebase, thus solving rendezvous. On the other hand, if rendezvous is not solvable for the given instance, then we would not have n unique views for the n nodes of the graph, and thus the algorithm would terminate with failure notification. ∎

Remark 1 *The correctness of algorithm* Compare-View *is based on the assumption that when an agent A visits the homebase of another agent B, at that time agent B must have already started executing the algorithm (i.e. it must have released the token). If this is not the case, the visiting agent A can simply wake-up the sleeping agent B and wait for it to start executing the algorithm.*

Theorem 5.1.2 *The total number of agent moves in an execution of algorithm* Compare-View *is $O(k \cdot \Delta^{2n})$ for a graph of size n with maximum degree Δ, and k agents.*

Proof : In algorithm *Compare-View*, all computation is done locally except the call to *Construct-View(h)* at the beginning and the move to the LEADER's homebase at the end. The latter takes only $O(k \cdot n)$ moves. Using the procedure *Construct-View*, each agent would be making $O(\Delta^h)$ moves to compute the view up to level h. So, the total number of moves made by the agents is $O(k \cdot \Delta^{2n-1})$. On the other hand, the algorithm requires just one bit of node memory (to mark the color on the node). ∎

Even though the algorithm *Compare-View* is an *effective* protocol for solving rendezvous, it requires an exponential number of agent moves. It is not known whether there exists more efficient algorithms for rendezvous in arbitrary networks using identical tokens, when the network topology is unknown. However if the network topology is known to the agents, it is possible to solve rendezvous in arbitrary networks, in an efficient way.

Theorem 5.1.3 *There exists an effective algorithm for rendezvous (in the token model) in arbitrary networks with knowledge of a map, such that the total number of moves made by all agent combined, during any execution of the algorithm, is $O(n \cdot k)$.*

Proof : The proposed algorithm simply computes the bi-colored surrounding for each agent and solves rendezvous by comparison of the surroundings (instead of view-comparisons as in the previous algorithm). Note that a single traversal is sufficient for constructing the surroundings of each agent, provided that the homebases are marked by tokens. Since every agent knows a map of the graph, each agent can traverse the graph in $O(n)$ moves. Thus the total number of moves performed during the algorithm is $O(n \cdot k)$. ∎

5.2 Rendezvous When Tokens Fail

We now consider the scenario where the tokens used by the agents are not reliable. After an agent places a token on a node and leaves that node, the token may unexpectedly disappear (or, fail). We assume the faults to be permanent i.e. once a token disappears, it may not appear again.

We make the following general assumptions:

[A1] At most $(k-1)$ tokens may fail;

[A2] A failed token never reappears;

[A3] A token either fails immediately on release or never fails.

5.2.1 In Ring Networks

We consider a ring network consisting of n nodes with bi-directional channels connecting each node to its two neighbors. The ring is not oriented, i.e. there is no universal concept of left and right.

Conditions for Solving Rendezvous

When solving rendezvous using tokens that may fail arbitrarily, we have the following result:

Lemma 5.2.1 *In presence of token failures, there is no terminating algorithm for solving the rendezvous problem in an asynchronous anonymous ring, if the value of k is unknown to the agents.*

Proof : Note that as long as no agent puts down its token, rendezvous is not possible. Let agent A be the first agent to put down its token at a node v and suppose agent A then starts traversing an edge e. Since the system is asynchronous, agent A may take a long time to traverse the edge e. Meanwhile if the token disappears then the other agents cannot distinguish this scenario from the one in which agent A was not present. Since k is not known, the other agents cannot determine whether to terminate after (k-1) agents have gathered or to wait for the extra agent. ■

For solving the *Rendezvous* problem, as before, each agent would put its token at its starting location, to mark its homebase. While traversing the ring, an agent can count the distances between successive tokens on the ring and thus obtain a sequence of inter-token distances. The sequence of inter-token distances S obtained by the agent (in the absence of failures) is of the form $(d_1,d_2,d_3, \ldots d_k)$ where for $1 \leq i < k$, d_i is the distance between ith and $(i+1)$th token and d_k is the distance between kth and the first token.
We define the following two operations on the sequence $S = (d_1,d_2,d_3, \ldots d_k)$

- The reversal operation gives us the sequence $Rev(S) = (d_k,d_{k-1}, \ldots d_1)$, called the reverse of S.

- For any $1 \leq i < k$, the rotation operation gives the sequence $Rot_i(S) = (d_{i+1}, d_{i+2}, \ldots d_k, d_1, \ldots d_i)$ which is called the i-th rotation of S.

The sequence $S = (d_1, d_2, d_3, \ldots d_k)$ is called rotation-reversal free (or RR-free) if for every $0 < i < |S|$, $Rot_i(S) \neq S$ and $Rev(Rot_i(S)) \neq S$. Notice that if $Rot_i(S) = S$ for some $1 \leq i < |S|$, then the sequence is periodic with period i and thus i divides both $k = |S|$ and n. On the hand if $Rev(Rot_i(S)) = S$ for some $1 \leq i < |S|$, then the (bi-colored) ring is symmetrical.

The sequence S is RR-free only if it is not periodic and the bi-colored ring is not symmetrical. In this case, there exists a unique location in the ring defined by Min-Point(S) as the location of the i-th token such $Rot_i(S)$ or $Rev(Rot_i(S))$ is the lexicographically minimum sequence among all sequences obtained by applying rotation and reversal operations on S. For any $1 \leq i \leq |S|$, we define $Sum_i(S)$ as the sum of the first i elements of S.

Theorem 5.2.1 *If there are k agents in a ring of n nodes with the initial location of each agent marked by a token, and S is the sequence of inter-token distances, then the rendezvous problem is solvable if and only if one of the following holds:*

1. *S is RR-free, or,*
2. *S is aperiodic and $S = Rev(Rot_i(S))$ for some $1 \leq i < |S|$ such that $Sum_i(S)$ or $n - Sum_i(S)$ is even.*

Proof : If S is RR-free, then the bi-colored symmetricity is equal to one and we can solve rendezvous. On the other hand, if S is periodic then the bi-colored symmetricity is greater than one and we cannot solve rendezvous. So, we are left with the case when S is aperiodic but the bi-colored ring is symmetrical. In this case, we can solve rendezvous if and only if there is at least one node of the ring which lies on the axis of symmetry. This will be satisfied if at least one of $Sum_i(S)$ or $n - Sum_i(S)$ is even. ∎

The above conditions are necessary and sufficient for solving rendezvous in a ring when the tokens do not fail. However, we shall show that even when f tokens fail for $1 \leq f < k$, the above conditions are still sufficient for solving rendezvous. In the next section we solve rendezvous for the simpler case when n and k are co-prime. In a later section, we present a solution that is *effective* i.e. one which solves rendezvous whenever the conditions of Theorem 5.2.1 are satisfied.

When n and k are co-prime

In this section we consider the rendezvous problem in a ring of size n with k agents initially, such that $\gcd(n, k) = 1$. In this case, the conditions of Theorem 5.2.1 would always be satisfied. Thus, rendezvous is solvable for any instance, when no tokens fail. However $0 \leq f \leq k$ tokens may fail. We use the following strategy to deal with the faulty tokens. Whenever the token of an

agent A disappears, then agent A becomes a LOSER agent. A LOSER agent remains stationary at some node acting like a *virtual* token. We define the Token-Count of a node v as the number of tokens plus the number of LOSER agents present at node v. When an agent traverses the ring it can compute a weighted sequence S_W of the form $((c_1, d_1),(c_2, d_2),\ldots (c_r, d_r))$ where the c_i's are Token-counts at the respective locations and the d_i's are the distances between them. We define Token-count(S_W) as follows:

$$\text{Token-count}(S_W) = \sum_{i=1}^{r} c_i$$

Similarly, Edge-Count(S_W) would denote the sum of the inter-token distances (i.e. d_i's) in S_W. Notice that Edge-Count(S_W)$= n$.

Definition 5.2.1 *Given a sequence S (or a weighted sequence S_W) of inter-token distances in a ring, we define the* rendezvous-point *or, RV-point(S) as follows. If S is RR-free, then RV-point(S) is simply the Min-Point i.e. the location of the ith token such that $Rot_i(S)$ is lexicographically smaller than $Rot_j(S) \forall j \neq i$ and $1 \leq j \leq |S|$. Otherwise if S satisfies the conditions of Theorem 5.2.1 clause-(2), then RV-point(S) is the middle node in the largest*[1] *even segment among the two segments defined by $S(1, i)$ and $S(i + 1, k)$. In all other cases, RV-point(S) is undefined.*

The following result immediately follows from the above definition.

Lemma 5.2.2 *If the sequence S satisfies the conditions of Theorem 5.2.1, then RV-point(S) returns the same location as RV-point($Rot_i(S)$) for all $i \in [1, k]$.*

We now present the algorithm for solving the Rendezvous problem when $\gcd(n, k) = 1$. For the following algorithm, we assume that both the value of n and k are initially known to the agents.

Algorithm *RVring1*

1. Put Token at the homebase, travel for n steps in any choosen direction to compute the sequence of inter-token distances S and return to homebase.

2. If Token at homebase has disappeared, then become LOSER and go to the next node having a token. A LOSER agent remains at this node until it receives instructions from another agent.

3. Else if $|S| = k$, then go to RV-point(S) and stop;

[1] largest in terms of edge-count and then Token-count.

4. Else re-traverse the ring to compute the weighted sequence of inter-token distances S_W, returning to homebase and then execute the following steps.

5. While (Agent-Count(S_W) $< k$) do
   Wait at homebase until another LOSER agent arrives at this node, then recompute S_W by traversing the ring again.

6. When Agent-Count(S_W) $= k$, compute RV-point(S_W), and go round the ring to communicate S_W to every LOSER and waiting agent. Let f be the number of LOSER agents.

7. If your homebase is guarded by a LOSER agent, then become an ACTIVE agent. Otherwise wait at the homebase.

8. If ACTIVE, go to the rendezvous-point v and wait for all other ACTIVE agents to arrive at v. (Note that there are exactly $k_1 = |\{c_i \in S_W : c_i > 1\}|$ ACTIVE agents.) Once all ACTIVE agents have arrived at v, one of them (or, maybe two of them), is(are) chosen as LEADER, depending on the distance to the agent's homebase. (If there are two LEADER agents, they stick together for the rest of the algorithm acting as single unit.)

9. If LEADER, then go round the ring to collect all waiting agents (agents that are not LOSER or ACTIVE). At each node u that contains a token but no LOSER agent, the LEADER waits for the token-owner to return, and collects it before moving to the next node.

10. When $(k - f)$ agents have gathered at the rendezvous-point, the LEADER goes round the ring again to collect all LOSER agents.

Lemma 5.2.3 *If no tokens fail, then algorithm RVring1 terminates correctly after Step 3, achieving rendezvous of all the k agents.*

Proof : If no tokens fail, then the sequence S computed by each agent would be of size k and would represent the actual inter-agent distances at the beginning of the algorithm. Also, the sequences by computed by distinct agents would rotations of one-another. Due to Lemma 5.2.2, the rendezvous-point computed by each agent would be the same. Thus, after executing step-3, all the agents would have gathered at the unique rendezvous location. ■

Theorem 5.2.2 *When $1 \leq f < k$ tokens fail, the algorithm RVring1 correctly solves rendezvous, after no more than $f \cdot n$ moves by any agent.*

Proof : Whenever a LOSER agent goes the next token and becomes stationary, the ACTIVE agent corresponding to that token, re-computes the weighted sequence S_{DT}. Let agent A be the last among the f LOSER agents to become stationary, say at node u. When the token owner at node v (say agent B), re-traverses the ring, it would obtain the final weighted sequence S_W having an agent-count of k. Thus there is at least one ACTIVE agent. Due to the assumption that $\gcd(n, k) = 1$, RV-point(S_W) is well-defined and unique location. Thus, all ACTIVE agents would gather at that location and then the LEADER agent would collect all other agents, thus solving Rendezvous.

Notice that no agent makes more than f traversals of the ring. Each makes a constant number of ring traversals, plus one extra traversal for each LOSER agent that comes to its homebase. ■

Theorem 5.2.3 *For algorithm RVring-1, the total number of moves made by the agent is $O(k \cdot n)$ and the memory requirement for each agent is $O(k \log n)$.*

Proof : In the above algorithm, each agent performs at least one traversal of the ring. For every token that is lost, one extra traversal of the ring is performed by some agent and a partial traversal of the ring is made by the LOSER agent. Thus, the total number of moves is less than $3k \cdot n$. The memory required to store the weighted sequence of inter-token distances is $O(logk + r \log n)$ where $r \leq k$ is the number of remaining tokens. ■

Effective Solution

The algorithm RVring1 from the last section solves the rendezvous only if $\gcd(n, k) = 1$ and thus, it is not an effective solution. An algorithm for rendezvous in a ring is *effective* if it always solve rendezvous whenever the conditions of Theorem 5.2.1 are satisfied. We now present such an algorithm called *RVring2* which is obtained by some minor modifications to the previous algorithm.

Algorithm *RVring2*

1. Put Token at the homebase, travel for n steps in any choosen direction to compute the sequence of inter-token distances S and return to homebase.
2. If Token at homebase has disappeared, then become LOSER and go to the next node having a token. A LOSER agent remains at this node until it receives instructions from another agent. A LOSER agent also remembers the distance to its original homebase.
3. Else if $|S| = k$, then go to RV-point(S) and stop;

4. Else re-traverse the ring to compute the weighted sequence of inter-token distances S_W, returning to homebase and then execute the following steps.

5. While (Agent-Count(S_W) $< k$),
   Wait at homebase until another LOSER agent arrives at this node, then recompute S_W by traversing the ring again.

6. When Agent-Count(S_W) $= k$, compute the sequence S of original inter-token distances, by going round the ring and gathering information from LOSER agents about their original locations.

7. If your homebase is guarded by a LOSER agent, then become an ACTIVE agent. Otherwise wait at the homebase.

8. If ACTIVE, go to RV-point(S) (say, node v) and wait for all other ACTIVE agents to arrive at v. (Note that there are exactly $k_1 = |\{c_i \in S_W : c_i > 1\}|$ ACTIVE agents.) Once all ACTIVE agents have arrived at v, one of them (or, maybe two of them), is(are) chosen as LEADER, depending on the distance to the agent's homebase. (If there are two LEADER agents, they stick together for the rest of the algorithm acting as single unit.)

9. If LEADER, then go round the ring to collect all waiting agents (agents that are not LOSER or ACTIVE). At each node u that contains a token but no LOSER agent, the LEADER waits for the token-owner to return, and collects it before moving to the next node.

10. When ($k - f = |S_W|$) agents have gathered at the rendezvous-point, the LEADER goes round the ring again to collect all LOSER agents.

Theorem 5.2.4 *Algorithm RVring2 is an effective algorithm for rendezvous in a ring network. The algorithm makes $O(n \cdot k)$ moves in total.*

Proof : Notice that if S is the original sequence of inter-token distances then RV-Point(S) returns a unique location if and only if the conditions of Theorem 5.2.1 are satisfied. Thus whenever these conditions are satisfied, the algorithm succeeds in solving rendezvous (this follows from the correctness of algorithm *RVring1*). Notice that the number of moves made by this algorithm cannot be more than that of algorithm *RVring1*. ∎

5.2.2 In Arbitrary Networks

We now extend the results of the previous section to networks of arbitrary topology. In this section, we assume that each agent has a map of the network. This means that the agents can do the following:

- Each agent can traverse the network G in $O(n)$ moves without marking any nodes.
- Each agent can compute the *surrounding* of any node in the network.

The following algorithm solves Rendezvous in arbitrary networks using tokens when $1 \leq f < k$ tokens fail.

Algorithm *RVgeneral*

1. Place Token at the homebase and traverse the network. For each node v in the network, compute the bi-colored *surrounding* of node v. Rank the nodes based on a fixed ordering on the bi-colored surroundings.
2. If (there exists k tokens in the network) do
   (i) If there are more than one nodes with the same rank, then terminate with failure;
   (ii) Otherwise the node with smallest rank is the rendezvous location. Go to the rendezvous location; Terminate the algorithm.
3. If the token at the homebase has failed, then traverse the network (remembering the labels on the path traversed) until a node containing a token is reached. Become LOSER and wait at this node.
4. Else, traverse the network and at each node, compute the Token-Count of the node and obtain information from LOSER agents (if any) about their homebases. Finally, return to the homebase.
5. While (sum of Token-Counts is less than k) do
   (i) Wait until another LOSER agent arrives at this node;
   (ii) If another LOSER agent arrives at this node, then traverse the network again to recompute the Token-Counts of the nodes; For each LOSER agent encountered during the traversal, obtain information about the location of its homebase.
6. If (sum of Token-Counts is k) do

(i) Compute the bi-colored surrounding of each node based on the information collected; Rank the nodes based on the bi-colored surroundings; Let $r = |\{v \in G : \text{TokenCount}(v) > 1\}|$.

(ii) If there are more than one nodes with the same rank, then terminate with failure;

(iii) Otherwise the rendezvous location is the node with smallest rank; If there are no LOSER agents at your homebase, then become WAITING and wait at the homebase. Else, traverse the network and at each node v do the following: If there is a token but no LOSER agent at v, then wait for the token-owner to return to v; Communicate the rendezvous location to every LOSER or WAITING agent. Finally (after the traversal), go to the rendezvous location.

(iv) If there are r agents at the rendezvous-location, then the last agent to arrive[2], makes another traversal of the network to instruct each WAITING and LOSER agent to move to the rendezvous location.

7. A LOSER agent that received instructions to move to the rendezvous location, waits until the agent A that owns the token at this node arrives. The LOSER agent then communicates the rendezvous-location to agent A and then moves to the rendezvous-location.

8. When all k agents have arrived at the rendezvous-location, each agent terminates with success.

Theorem 5.2.5 *Algorithm* RVgeneral *is an effective algorithm for rendezvous in arbitrary networks, when a map is available.*

Proof : Recall that when a map is available, an algorithm for rendezvous in arbitrary networks is *effective* if it always solves rendezvous whenever the closed-symmetricity of the network is 1. The effectiveness of the algorithm follows from the following observations: (i) There is always at least one active agent. (ii) There exists an active agent A such that the sum of the Token-counts computed by agent A is k. (iii) If the sum of the Token-counts computed by an agent A is equal to k, then agent A can correctly compute the bi-colored surrounding of every agent in the network. ∎

Theorem 5.2.6 *The total number of moves made during Algorithm* RVgeneral *is* $O(n \cdot k)$.

Proof : During the algorithm at most $2k$ complete traversals of the graph are performed in total. Each agent performs one traversal in the beginning and one traversal at the end

[2]if multiple agents arrive through multiple ports, at the same time then the winner is decided based on the labels of the ports

(unless the agent becomes a LOSER). For each LOSER agent, one extra traversal of the graph is performed by exactly one active agent. Notice that each traversal of the graph takes $O(n)$ moves, since the agents have a map of the graph. ■

5.3 Conclusions

In this chapter, we presented solutions to the rendezvous problem in the token model, when each agent contains a single identical token. We first considered algorithms for the fault-free scenario. In this case each agent can mark its homebase using the token, and thus we can solve rendezvous by simply comparing the bi-colored views of the agents. This algorithm makes $O(k \cdot \Delta^{2n})$ in total. When the map of the network is known, $O(n \cdot k)$ moves are sufficient to solve the rendezvous problem.

We also considered solutions to rendezvous using faulty tokens, i.e. tokens which may disappear unexpectedly. We showed if at least one of the tokens is non-faulty, we can still solve rendezvous in $O(n \cdot k)$ moves, when the map of the network is known.

Chapter 6

Rendezvous in the Whiteboard Model

In this chapter, we present solutions to the Rendezvous problem using whiteboards for communication among agents. As mentioned in Chapter 3, previous solutions for Rendezvous in the whiteboard model were presented by Barrière et al. in [17], under the additional assumption of the presence of *sense of direction.* We do not make such an assumption in this chapter.

6.1 Conditions for Solving Rendezvous

Recall that it is not always possible to solve the rendezvous problem in anonymous networks. The following impossibility results were presented in [59] and [17] for rendezvous in the ring networks and in arbitrary networks with *sense of direction*, respectively. We re-state them for the general case.

Lemma 6.1.1 *In an arbitrary graph of n nodes with k agents, the Rendezvous problem is not solvable if the agents know neither the value of n nor k.*

Lemma 6.1.2 *In an arbitrary graph of n nodes with k agents,, the Rendezvous problem is not solvable, in general, if* $\gcd(n,k) > 1$ *i.e. if n and k are not co-prime.*

The above result is due to the fact that when n and k are not co-prime, an adversary which decides the initial positions of the agents, may decide to place the agents in exactly symmetrical positions with respect to each other (provided that the graph itself is symmetrical; e.g. a ring), such that, no deterministic algorithm can break the symmetry among the agents and achieve rendezvous. However, when n and k are co-prime, we can be sure that it is possible to rendezvous, irrespective of the graph topology or the initial locations of the agents.

Notice that that Lemma 6.1.1 holds even when $\gcd(n,k) = 1$. In the next section, we present a solution to the Rendezvous problem in anonymous networks, assuming that $\gcd(n,k) = 1$ and the agents have prior knowledge of the value of at least one of n and k. Under these conditions, we show that it is possible to solve the *Rendezvous* problem, using just $O(\log n)$ node memory and making $O(k \cdot m)$ moves. The moves complexity can be decreased to $O(m \log k)$ when both n and k are known.

6.2 Solving RV when n and k are co-prime

We present an algorithm that solves Rendezvous by solving the Labelled Map Construction problem (recall that the problems are equivalent). We make use of the procedure EXPLORE from Chapter 4 which partitions the network into disjoint territories owned by each agent. In this section, we show how the agents can merge their territories, construct a spanning tree of the graph and then finally use it to solve Rendezvous. Recall that the task of merging the territories of the agents is complicated by the fact that the territory constructed by two agents may look exactly similar and it is difficult to distinguish 'NT' edges connecting two distinct trees from those (cyclic) NT-edges which connect two nodes of the same tree.

As mentioned in Chapter 3 there exists a well-known distributed algorithm for minimum spanning tree construction(MST) given by Gallager, Humblet and Spira(GHS) [67], where the spanning tree is constructed by repeatedly joining adjacent trees using the unique edge of minimum weight connecting them. Such an approach is unfortunately not applicable in our setting, since neither the edges nor the nodes have unique labels, making it impossible for the agents to agree on a unique edge for joining two trees.

Thus, in our setting, we need a much more complicated protocol for merging the maps and building the spanning tree. Such a distributed algorithm, MERGE-TREE, is described in the following.

6.2.1 Algorithm MERGE-TREE

The algorithm proceeds in phases, where in each phase, some agents become passive i.e. they stop participating in the algorithm. Agents communicate by writing certain symbols on the whiteboards. Two special symbols would be used which we call the 'ADD-ME' symbol and the 'DEFEATED' symbol. An agent can be in one of three states: *Active*, *Defeated* or *Passive.* Each agent is *active* at the time it starts the algorithm, but it may become *defeated* and subsequently *passive*, during some phase of the algorithm. When an agent becomes *passive* during a phase, it keeps waiting at its current location till the end of the algorithm. At the time an agent starts the algorithm, it knows the value of either n or k.

Algorithm *MERGE-TREE*

Phase 0 : Each agent on startup executes procedure EXPLORE and when it finishes, it has a map of the territory marked by it and also a count of the number of nodes marked. Each agent maintains a *LABEL* which is of the form (Ph, Nc, Ac) where Nc (Node-Count) is the count of the number of nodes marked by it, Ac (Agent-Count) is the number of agents in its territory (initially set to 1) and Ph is the phase number which is also initially set to 1. Now the agent can begin the first phase.

In phase i, $1 \leq i < k$, an agent A (if *active*), executes the following steps:

STEP 1 – 'WRITE-LABEL' : Agent A does a depth-first traversal of its territory using the map; recall that a territory is a tree. During the traversal the agent writes its LABEL on the whiteboard[1] of each node in its tree.

STEP 2 – 'COMPARE LABEL' : During this step, the agent compares its LABEL with the LABEL's in adjacent trees. Agent A starts a depth-first traversal of its territory. During the traversal, whenever it finds an 'NT' edge $e = (u, v)$ incident to some node u in its territory, it traverses the edge e to reach the other end v, compares its LABEL with the LABEL at v, and takes an appropriate action, before returning back to u. If it does not find any LABEL at node v (or, finds a LABEL from the previous phase $i - 1$), it waits till the LABEL for phase i is written at v. On the other hand, if it finds a LABEL from phase $i + 1$ at node v, it ignores that LABEL, goes back to u and continues with the traversal.

Two LABEL's from the i-th phase, $T_1 = (i, N_1, K_1)$ and $T_2 = (i, N_2, K_2)$, are compared as follows. LABEL T_1 is said to be larger than LABEL T_2 if either $N_1 > N_2$, or $N_1 = N_2$ and $K_1 > K_2$. The two LABEL's are said to be equal if both $N_1 = N_2$ and $K_1 = K_2$. Otherwise, LABEL T_1 is smaller than LABEL T_2.

After the LABEL comparison, the agent takes one of the following actions:

[$<$] If the LABEL at the other side is larger, it writes a 'ADD-ME' symbol on the whiteboard of node v and returns to node u. It remembers[2] the node u, (as the *terminal* node) and the edge e (as the *bridge* edge). It then does a complete traversal of its territory writing 'DEFEATED' symbols on each node in its territory. It now becomes *defeated*. (The actions taken by a *defeated* agent are described below.)

[$=$] If the LABEL at the other side is equal to its own LABEL, it ignores the LABEL, returns to its own tree and continues with its traversal.

[1] Any previously written LABEL or symbol is deleted from the whiteboard.

[2] The agent remembers a node by marking in its map.

[>] If the LABEL at the other side is smaller, it waits at node v till it finds a 'DEFEATED' symbol. On finding a 'DEFEATED' symbol, it goes back to u and continues with the traversal.

If agent A becomes *defeated* then it takes the following actions. It continues with the traversal and LABEL comparisons — whenever it finds a LABEL which is smaller or equal to its LABEL, it takes the same action as an *active* agent; but, when it finds a LABEL that is larger than its LABEL, it ignores it. (So, a *defeated* agent never writes any 'ADD-ME' symbol.) After completing the traversal, the *defeated* agent A returns to the *terminal* node u and marks the *bridge* edge e as a 'T' edge. It then traverses the edge e to reach the other end, say v. It adds the edge e to its map and designates the vertex corresponding to node v, as the *junction point*, in the map. At this stage, the agent A becomes *passive* and goes to sleep.

During the traversal, whenever an *active* (or *defeated*) agent A finds an 'ADD-ME' symbol at some node w in its tree, it takes the following action. It deletes the 'ADD-ME' symbol and waits at node w till the agent B (which had written the message), returns back to w. Agent A then acquires all the information available in agent B's memory, including B's LABEL, its map and all other LABEL's and maps acquired earlier by agent B. Agent A also remembers the vertex corresponding to node w, as the location where it acquired this new information. (This vertex is called the *acquisition point.*)

STEP 3 – 'UPDATE LABEL' : If the agent A completes the second step without becoming *passive*, it extends its territory and updates its LABEL, before starting the next phase. The agent adds together the Node-count and Agent-count values respectively, from all the acquired LABEL's, including its own LABEL, to get the new values of Node-count Nc, and Agent-count Ac. The new phase number is obtained by incrementing Ph by one. The agent also constructs a new map by merging the acquired maps with its own map. Note that the agent has the information about how to merge the maps[3]. (While merging the maps, the agent may have to relabel some of the vertices of the maps, to ensure unique labelling of the vertices.) The resulting map constructed by the agent defines its new territory.

On updating the LABEL, if the agent finds that the new node-count is equal to n (or the agent-count is equal to k), then it reaches the termination condition. Otherwise, it proceeds with the next phase.

Phase k : An agent which reaches this phase terminates the algorithm after sending failure notification to all agents in its territory.

When an agent A reaches the termination condition, it becomes the leader agent; at this stage, it has a spanning tree of the whole graph. Finally, it executes the following procedure:

[3]The maps are disjoint except for the joining vertex.

Procedure GATHER

1. The leader agent executes a depth-first traversal of the spanning tree, writing node labels on the appropriate whiteboards, starting with its homebase which is labelled 1. In a subsequent traversal, all the non-tree edges are added to the map.
2. The leader agent traverses the graph, waking up each passive agent and communicating the full map to all the agents.
3. Every passive agent that is woken, moves to the smallest labelled node on the map.

In the next section, we show that algorithm always works correctly, under the assumption that n and k are co-prime. We shall also analyze the complexity of the algorithm and then some improvements to the algorithm.

6.2.2 Analysis of the Algorithm

Throughout this section, we shall use the following notations. G_{iA} denotes the subgraph of G that corresponds to the territory of agent A at the time when it reaches the end of phase i. If A becomes *passive* in phase i, then $G_{iA} = \phi$. We denote by Γ_i the set of all agents which start phase i, in *active* state. We say that the algorithm reaches phase i, if there is at least one agent that starts phase i.

Whenever an agent A becomes *defeated* on comparing its LABEL with the LABEL of an agent B, during phase i, we say that agent A was defeated by agent B in phase i. In that case we know that B was *active* at the start of phase i and B's LABEL in phase i was larger than A's LABEL, in phase i.

The following facts imply that there is no deadlock in the algorithm MERGE-TREE.

Lemma 6.2.1 *(a) An (*active*) agent that starts phase i, either completes the phase or becomes* passive *during the phase. (b) At the end of every phase i reached by the algorithm MERGE-TREE, there is always at least one* active *agent.*

Proof : Part(a): We show that there cannot be any cyclic waiting among the agents. Suppose, for the sake of contradiction, that there is a group of agents $A_1, A_2, \ldots, A_t$ such that for each $1 \leq j \leq t-1$, A_j waits for A_{j+1}, and A_t waits for A_1. We represent these agents as vertices of a graph and we draw directed (colored) arcs to denote who waits for whom. (The color of the edge denotes the type of waiting.) There are three situations when an agent A, in phase $i \geq 1$ has to wait at a node v for some agent B:

1. Agent A found no LABEL, or a LABEL from phase $(i-1)$ at node v and it is waiting for the LABEL for phase i to be written, by agent B. [Denoted by *Blue* arc.]

2. Agent A is waiting at node v, after finding an 'ADD-ME' symbol written by B. [Denoted by *Yellow* arc.]

3. Agent A found agent B's LABEL (at node v) to be smaller than its own LABEL. This indicates A is waiting for agent B to write a 'DEFEATED' symbol at v. [Denoted by *Red* arc.]

Note that in first case above, agent B is either in a lower phase than A, or B is yet to complete step-1 of phase i. But in the other two cases, both A and B have to be in step-2 of the same phase i and also B would have a smaller LABEL than A in that phase. Also note that an agent can be waiting only if it is in step-2 (i.e. the COMPARE-LABEL step) of some phase.

So, each $A_j, 1 \leq i \leq t$, is in step-2 of some phase and for any $1 \leq j, k \leq t$,

- there is a blue arc from A_j to A_k if and only if A_k is in smaller phase than A_j.
- there is a red or yellow arc from A_j to A_k if and only if A_k is in the same phase as A_j but has a smaller LABEL than A_j.

Let us first consider the case when at least one of the arcs in the cycle is blue. Let A_j be the first node with a blue arc in the cycle (connecting A_j to A_{j+1}). Let A_j be in phase i. By definition of red and yellow arcs, we then know that any A_k with $k < j$ are in the same phase i. If $j = t$, we immediately have a contradiction because, by definition of blue arc, A_1 would have to be in a phase smaller than i. Let us then assume that $2 \leq j \leq k - 1$. Also in this case we have a contradiction because, by definition of blue arc, $A_{j+1}, A_{j+2}, \ldots, A_t$ must be in a smaller phase than phase i. So, there can neither be a blue nor a red (or yellow) arc from A_t to A_1. Thus, we cannot have a cycle containing any blue arc.

We now consider the case when the cycle is composed by yellow and red arcs only. In this case, all the agents in the cycle would be in the same phase i, and each agent would have a smaller LABEL than the agent on its left — which is not possible!

So, we conclude that there can not be any cyclic waiting among the agents. Each agent in phase i, either reaches the end of the phase or becomes *passive.*

Part(b): Note that an agent A can be defeated by an agent B during phase i, only if B's LABEL in phase i is larger than A's LABEL, in phase i. So, an agent A having the largest LABEL in phase i cannot be defeated in phase i. Thus, agent A remains *active* at the end of phase i. ■

We shall show next that the algorithm MERGE-TREE terminates in finite time, and whenever $\gcd(n, k) = 1$, there is exactly one leader agent on termination and the map constructed by the leader agent is a spanning tree of the graph G.

Lemma 6.2.2 *The following holds for any phase i that is reached by the algorithm:*

1. *For each $A \in \Gamma_i$, G_{iA} is a tree.*

2. *For any $A, B \in \Gamma_i$, if A and B are distinct, then $G_{iA} \cap G_{iB} = \phi$.*

3. *$H_i = \bigcup_{A \in \Gamma_i} G_{iA}$, is a subgraph of G having the same vertex set as G.*

Proof : For phase $i = 0$, the territory obtained by each agent in phase i is the same as obtained from the EXPLORE algorithm. Thus, the given conditions hold in phase $i = 0$, as proved in section 4.3. We assume that these conditions hold at some phase $i = r$ that is reached by the algorithm and we show that each of these conditions would continue to hold at phase $i = r+1$, if the algorithm reaches phase $r + 1$.

For any agent A which reaches phase $r + 1$, the following holds: If agent A does not find any 'ADD-ME' symbol during phase r, then $G_{(r+1)A} = G_{rA}$. On the other hand, if agent A read an 'ADD-ME' symbol in phase $r+1$, then it acquires the territory of some *defeated* agent B that wrote the 'ADD-ME' symbol in phase $r + 1$. The territory of such a *defeated* agent B consists of G_{rB} combined with a single edge e that connects a node in G_{rA} to a node in G_{rB} (where G_{rA} and G_{rB} are disjoint trees, by our assumption). Thus, the new territory of A at the end of phase $r + 1$ would still be a tree.

Agent A acquires some territory from agent B in phase $r + 1$, only if it defeats agent B in phase $r + 1$. Note that an agent can be defeated only once and by only one agent. Thus, if A and C are two agents that reach the end of phase $r + 1$, then both of them could not have acquired the territory of the same agent B. This implies that the territories of the agents A and C would remain disjoint at the end of phase $r + 1$. Thus, $G_{(r+1)A} \cap G_{(r+1)C} = \phi$ for any two agents A and C that reaches the end of phase $r + 1$.

Note that whenever an agent becomes *passive* in phase $r + 1$, its territory is acquired by the agent that defeated it. So, each node that was contained in the territory of some *active* agent at the end of phase r, would be contained in the territory of some *active* agent at the end of phase $r + 1$. In other words $H_{r+1} \supseteq H_r$. Thus, the third condition also holds for phase $i = r + 1$. ■

For an agent $A \in \Gamma_i$, we define $NodeCount(G_{iA})$ to be the number of nodes in G_{iA}. Note that, this is equal to the Nc part in the LABEL of agent A for phase $i + 1$. Similarly, $AgentCount(G_{iA})$ is defined to be the number of nodes in G_{iA} that are agent homebases. This is equal to the Ac part in the LABEL of agent A in phase $i+1$. We have the following corollary as a consequence of the above lemma:

Corollary 6.2.1 *For any phase i reached by the algorithm, we have*

$$\sum_{A \in \Gamma_i} NodeCount(G_{iA}) = n \quad and \quad \sum_{A \in \Gamma_i} AgentCount(G_{iA}) = k$$

Two agents A and B are said to be neighbors in phase i, if G contains an edge $e = (u, v)$ such that G_{iA} contains the vertex u and G_{iB} contains the vertex v. Note that the edge e is not included in either G_{iA} or G_{iB} (as $G_{iA} \cap G_{iB} = \phi$), so e would remain to be marked as an 'NT' edge at the end of phase i.

Lemma 6.2.3 *If* $\gcd(n, k) = 1$ *then, for any phase* $i \geq 1$ *with* $|\Gamma_i| \geq 2$, *at least one agent* $B \in \Gamma_i$, *becomes* passive *during phase* i.

Proof : If $t = |\Gamma_i| \geq 2$ then t agents would have reached the end of phase $i - 1$. Note that all the t agents cannot have the same node-count and agent count at the end of phase $i - 1$ (because then we would have $\gcd(n, k) \geq t \geq 2$). So, there must be two (neighboring) agents A and B, with different LABEL's and thus, one of them would defeated in the LABEL comparison during phase i. ■

So, if the condition $\gcd(n, k) = 1$ is satisfied, then in each phase, at least one of the *active* agents, becomes *passive*, until in some phase i, there is only a single *active* agent left. The territory of this agent A, would be the tree G_{iA} containing all the nodes of G (due to Lemma 6.2.2), and the node-count and agent-count of A would equal n and k respectively. Thus, agent A would reach termination condition and the algorithm would terminate. Notice that the algorithm always terminate within k phases, irrespective of the values of n and k.

Lemma 6.2.4 *If an agent A reaches phase $i = k$, then* $\gcd(n, k) > 1$.

Proof : Due to Lemma 6.2.3, if $\gcd(n, k) = 1$, only one of the k agents can reach phase $k - 1$ and in that case, the territory of this agent in phase $k - 1$ would contain all the n nodes of G and thus, this agent would terminate at the end of phase $k - 1$. ■

Hence, the algorithm fails only if $\gcd(n, k) > 1$.

Theorem 6.2.1 *The algorithm MERGE-TREE terminates after at most k phases, and if* $\gcd(n, k) = 1$ *then exactly one agent A reaches the termination condition; when this happens, G_{iA} represents a spanning tree of G.*

Theorem 6.2.2 *After the execution of the procedure GATHER, every agent reaches the home-base of the unique leader agent.*

This proves the correctness of our algorithm.

Theorem 6.2.3 *The number of edge traversals made by the agents during algorithm MERGE-TREE is in $O(k \cdot m)$ where m in the number of edges in the graph G.*

Proof : During procedure EXPLORE, the agents perform at most $4m$ moves. During each phase of the algorithm MERGE-TREE, each 'T' edge is traversed twice in step-1 and twice in step-2, during the depth-first traversals; Each NT edge is traversed at most four times in step-2. This accounts for $4m$ moves per phase and thus a total of $O(k \cdot m)$ moves. Other than that each defeated agent does one extra traversal of its tree to write 'DEFEATED' messages. These extra moves would account for at most $O(k \cdot n)$ edge traversals. Finally, the procedure GATHER takes $O(m)$ edge-traversals. ■

As for the memory requirement, only $O(\log n)$ bits of whiteboard memory are needed per node of the graph, which is sufficient for writing a LABEL on any whiteboard. Note the marking of the T-edges and NT-edges does not have to be done by explicitly writing on the whiteboards. Each agent can use its own map to locate the T-edges in its territory and distinguish them from the NT-edges incident to the territory.

6.2.3 Reducing the number of Phases

When the values of n and k are co-prime, then the algorithm will never fail to elect a leader. In case the agents have the prior knowledge that n and k are co-prime, then we can simplify the algorithm and reduce its complexity by allowing only those agents which defeat some other agent to proceed to the next phase. This new algorithm is described below. As before the algorithm proceeds in phases and during the algorithm an agent can be one of the states – *Active*, *Passive*, or *Leader*. Every agent begins in state *Active* with the initial knowledge of the value of both n and k (or else the value of $\gcd(n, k)$ along with one of the values n or k).

Algorithm Explore-&-Capture

If $\gcd(n, k) > 1$, terminate the algorithm with failure notification. Else proceed with the first phase. In each phase $i \geq 1$, an agent A (if *Active*) executes the following steps:

STEP 1: Agent A executes procedure EXPLORE using the phase number i as a tag for all marks that it makes. During the execution of EXPLORE, if agent A at some node v, finds a mark with phase number $j < i$ then v is considered to be unmarked. (In this case, agent A overwrites such marks with its own mark.) On the other hand, if node v is marked with $j > i$, then agent A aborts the execution of EXPLORE and changes to *Passive* state.

The territory obtained by an *active* agent A at the end of procedure EXPLORE is denoted by T_{iA}. Let n_i be the number of nodes and k_i be the number of homebases in T_{iA}. The LABEL for agent A in this phase would be $Q_A = (i, n_i, k_i)$.

If $n_i = n$ (or equivalently $k_i = k$) then agent A changes to state *Leader* and executes procedure GATHER. Otherwise, it continues with the next step.

STEP 2: An active agent A performs a depth-first traversal of the territory T_{iA} and writes the LABEL Q_A on the whiteboard of each node in the territory.

STEP 3: At the start of this step, agent A initializes its *Win-Count* to zero and then starts another depth-first traversal of its territory. During the traversal, whenever it finds an outgoing edge $e = (u, v)$ (where $u \in T_{iA}$ but $v \notin T_{iA}$), agent A visits the node v and reads the LABEL written at node v. If agent A finds no LABEL's from phase i or higher, at node v, then the agent waits until such a LABEL, (say Q_B) is written and then takes the following action, before continuing with the traversal:

(i) If $(Q_B > Q_A)$ agent A writes 'ADD-ME' at the node v, traverses its territory writing 'DEFEATED' on each node in T_{iA} and then changes to passive state.

(ii) If $(Q_B < Q_A)$ then agent A waits at node v, until it finds a 'DEFEATED' symbol written at node v.

During this step, whenever agent A finds an 'ADD-ME' symbol written in any node of T_{iA}, it deletes the 'ADD-ME' symbol and increments its *Win-Count*. When agent A completes this step, if the *Win-Count* of agent A is still zero then it changes to passive state.

Any agent that becomes passive in this phase returns to its homebase and waits until it receives the map from the *Leader* agent. Those agents that did not become passive, continue with the next phase. Finally an agent that becomes *Leader*, executes the procedure GATHER to terminate the algorithm.

In the following, we show the correctness of this algorithm. Notice that any active agent has to wait only in STEP-3 of the algorithm. It can be easily shown that there is no deadlock due to this waiting.

Lemma 6.2.5 *For any active agent A executing STEP-3 of some phase i of the algorithm, the following holds: (a) Agent A does not wait forever in STEP-3 (b) Agent A either itself becomes passive or causes another agent from phase i to become passive.*

Proof : Part(a): First notice that if agent A is in STEP-3 of phase i, then there is at least one other agent in phase i, and all those agents which are neighbors of agent A must have at least reached STEP-1 of phase i. Now, suppose agent A is waiting to find a LABEL from phase i at node v (which belongs to the territory of agent B). So, agent B (which is in phase i) must write its LABEL at node v during STEP-2 (unless it became passive in STEP-1, in which case there is an agent in higher phase which would reach and mark node v) and then agent A would stop waiting. In the other case of waiting, suppose agent A found a smaller LABEL Q_B at

node v and is waiting to find the 'DEFEATED' symbol. In that case, agent B is at least in STEP-2 of phase i, so in STEP-3 it will find a larger LABEL and write 'DEFEATED' on all nodes in T_{iB}. Thus, agent A stops waiting.
Part(b): If agent A did not became passive in STEP-3, then it must have seen at least one 'ADD-ME' symbol. Notice that the agent that wrote this 'ADD-ME' symbol must be in STEP-3 of phase i and thus it would become passive during phase i. ■

Lemma 6.2.6 *If* $\gcd(n, k) = 1$ *then, for any phase* $i \geq 1$ *with* $|\Gamma_i| = r \geq 2$*, the following holds: (a) At least* $r/2$ *agents become* passive *during phase* i*, and (b) At least one agent reaches phase* $i + 1$.

Proof : Part(a): Suppose $r' \leq r$ agents complete STEP-3 of phase i without becoming passive, then each of these r' agents must have caused some other agent from this phase to become passive (due to Lemma 6.2.5). So, $r' \leq r/2$ and thus, $(r - r') \geq r/2$ agents become passive in phase i.
Part(b): If $\gcd(n, k) = 1$, then among the r agents $\in \Gamma_i$, there would be two neighboring agents A and B, such that LABEL(A) > LABEL(B). So, during STEP-3 of phase i, agent B would find a larger LABEL and thus write 'ADD-ME' symbol at some node v. The agent whose territory in phase i contains node v would reach phase $i + 1$. ■

Theorem 6.2.4 *Algorithm* Explore-&-Capture *terminates after at most* $\log k$ *phases, electing a unique leader agent and constructing a uniquely labelled map of* G *if and only if* $\gcd(n, k) = 1$.

Proof : If $\gcd(n, k) \neq 1$ then every agent terminates before starting the first phase. If $\gcd(n, k) = 1$, then due to Lemma 6.2.6, in some phase i there would be only one agent $A \in \Gamma_i$. Thus, in phase i, the territory of agent A, T_{iA} would be a spanning tree of the graph G and agent A would become leader and execute procedure GATHER to obtain a uniquely labelled map of G. Again due to Lemma 6.2.6, $k/2^{i-1} > |\Gamma_i| = 1$ which implies that $i \leq \log k$. ■

Theorem 6.2.5 *The total number of edge traversals made by the* k *agents during algorithm* Explore-&-Capture *is* $O(m \cdot \log k)$ *where* m *in the number of edges in the graph* G*. The algorithm requires* $O(\log n)$ *memory at each node.*

Proof : Using arguments similar to those for the previous algorithm, it can be shown that each edge is traversed a constant number of times in each phase of algorithm *Explore-&-Capture*. Thus, there are $O(m)$ edge-traversals per phase of the algorithm, adding up to a total of $O(m \log k)$ traversals for the whole algorithm. Notice that this algorithm requires the same amount of whiteboard memory as the previous algorithm. ■

6.3 Designing Effective Solutions

In the previous section, we have considered only those instances of the problem where $\gcd(n, k) = 1$, because we know that rendezvous (or the agent election) is unsolvable in general, when n and k are not co-prime. However, when $\gcd(n, k) > 1$, there are still some specific instances of the problem which are solvable. For example, if the graph is a tree with an odd number of nodes, then it is always possible to elect a leader, regardless of whether n and k are co-prime or not. We shall now present an *effective* solution to the rendezvous problem, i.e. an algorithm that is able to detect if the problem is solvable in a given setting and then solves it whenever it is possible to do so. The algorithms presented in this section simply elect a leader agent; a solution to rendezvous can be obtained by executing procedure GATHER at the end.

6.3.1 Partial-Views

First notice that an effective algorithm for agent election can be obtained by simply using the views of the agents for comparison among them. However, such a solution would be very inefficient (recall from Chapter 4 that constructing the view requires an exponential number of moves). In order to reduce the complexity of solving the agent election problem, we can use the following approach. Each agent can explore only a part of the graph and construct a *partial* view of the part explored. The agents can then communicate with each-other to find out the partial-view constructed by other agents and use this for comparison.

For the initial exploration, we again use the procedure EXPLORE. During procedure EXPLORE, each agent A can label the nodes in its territory $\mathcal{T}_A$ by marking them with numeric identifiers, i.e. numbering them (in the order that they are visited). Thus, within the territory of an agent, each node could be uniquely identified. However, the nodes on two different trees may have the same label. So, once an agent traverses an 'NT'-edge to reach another marked node, it is generally not possible for the agent to determine if that node belongs to its own territory or that of some other agent.

Based on the territory of an agent, we define the *Partial-View* PV_A of an agent A having territory $\mathcal{T}_A$, as the finite rooted tree, such that:

(i) The root corresponds to the homebase v_0 of agent A.

(ii) For every other node v_i in $\mathcal{T}_A$, there is a vertex x_i in PV_A.

(iii) For each edge (v_i, v_j) in $\mathcal{T}_A$, there is a edge (x_i, x_j) in PV_A.

(iv) For each outgoing edge $e = (v_i, u_i)$ such that $v_i \in \mathcal{T}_A$ but $e \notin \mathcal{T}_A$, PV_A contains an extra vertex y_i (called an external vertex) and an edge $\hat{e} = (x_i, y_i)$ that joins x_i to it.

(v) Each edge in PV_A is marked with two labels, which are same as those of the corresponding edge in G.

(vi) Each vertex in PV_A is colored black or white depending on whether the corresponding node in G is a homebase or not.

(vii) Each vertex is also labelled with the numeric identifier assigned to the corresponding node of G.

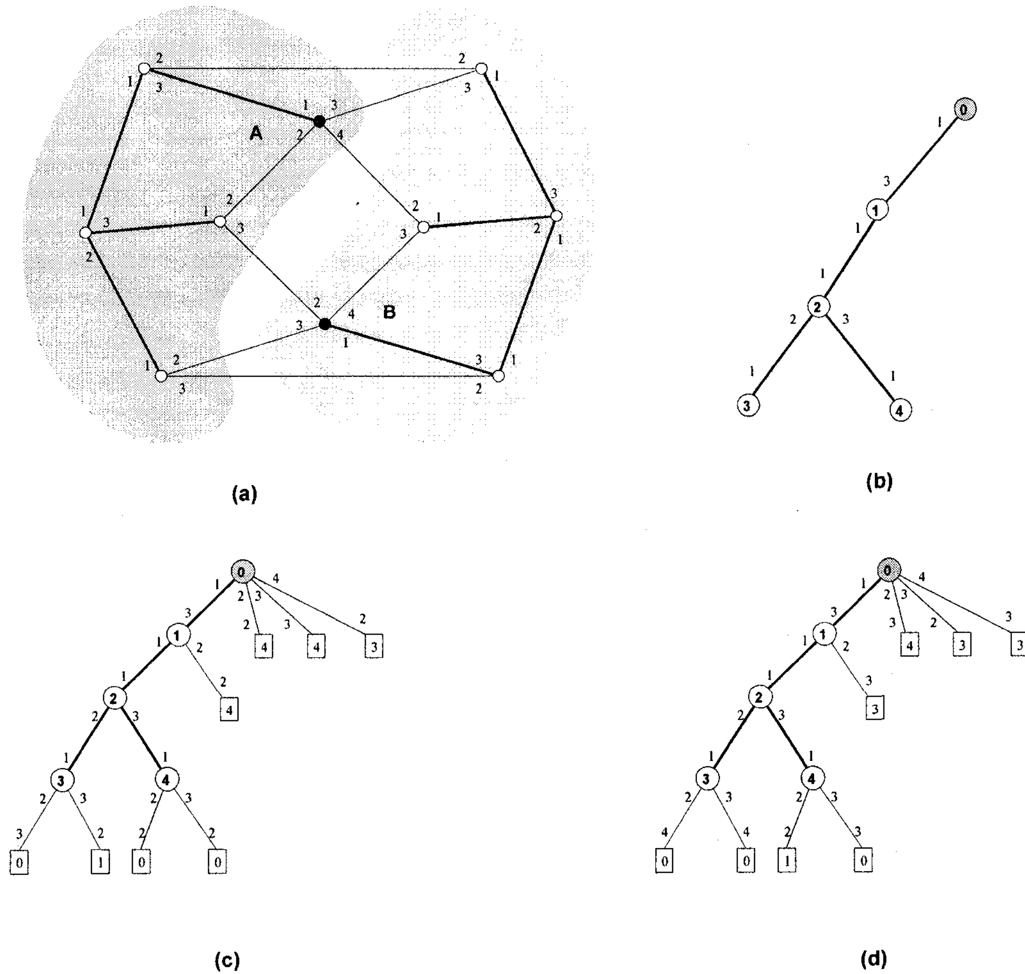

Figure 6.1: Territories and Partial-Views: Figure (a) shows a graph G with 10 nodes and two agents A and B (whose territories are marked by **bold** edges). The territory of each agent looks exactly the same, as shown in figure (b). However, the *Partial-View* PV_A for agent A (figure (c)) is different from the *Partial-View* PV_B for agent B (figure (d)).

During the execution of algorithm EXPLORE, each agent can build its Partial-View. We have the following important result:

Lemma 6.3.1 *If the agent election problem is solvable for an instance (G, λ, p) then, after the execution of algorithm EXPLORE on (G, λ, p), there would be at least two agents having distinct Partial-Views.*

Proof : Suppose every agent has an identical Partial-View. In this case we can obtain the complete bi-colored view of an agent to any required depth d as follows. Take the Partial-View PV and recursively replace each external vertex having label x, with a copy of PV rooted at the internal node labelled x. This means that the bi-colored views of the agents would be identical if their Partial-View are identical, and so by Lemma 5.1.1 the AEP problem is unsolvable in this case. ■

The above result implies that we can use the Partial-Views to distinguish between some of the agents. We fix a particular encoding for the partial-views so that we can compare among the PV's of the agents.

Encoding of Partial-View

We can encode the Partial-View of a node v, layer by layer, as follows. Each layer is encoded as the concatenation of the encodings of the vertices in that layer, ordered according to the labelled path from the root v to that vertex. The encoding of a vertex u consists of the degree and the color of u, followed by the labels $l_u(e), l_w(e)$ (ordered according to the $l_u(e)$ values) for each outgoing edge $e = (u, w)$ that leads to a child vertex w. Each external vertex is encoded by three numbers viz the degree(0), the color and the numeric identifier (as assigned by algorithm EXPLORE).

More formally, the encoding of a Partial-View PV_A is given by

$$\mathcal{E}(PV_A) = (L_0, L_1, ...L_h)$$

where h is the height of the tree PV_A and each L_j(i.e. encoding of level-j) is given by:

$$L_j = (\gamma(v_1), \gamma(v_2), ...\gamma(v_r))$$

where $v_1, v_2, ...v_r$ are the vertices at level-j of PV_A, ordered according to the labelled path from the root to that vertex. For an internal vertex v, the encoding of v is given by:

$$\gamma(v) = (d = Degree(v); Color(v); \lambda_v(e_1), \lambda_v(e_2), ...\lambda_v(e_{d-1}); \lambda_{w_1}(e_1), \lambda_{w_2}(e_2), ...\lambda_{w_{d-1}}(e_{d-1}))$$

where $w_1, w_2, ...w_{d-1}$ are the child vertices of v, that are connected to v through edges $e_1, e_2, ...e_{d-1}$, such that $\lambda_v(e_1) < \lambda_v(e_2) < ... < \lambda_v(e_{d-1})$. For an external vertex v, the encoding of v is given

by:

$$\gamma(v) = (Degree(v) = 0; Color(v); Label(v))$$

We can define a fixed total order on the PV's based on this encoding and thus we can do comparisons between agents using the encoded PV's. We shall show how such comparisons can be used for an effective leader election algorithm given below.

6.3.2 Algorithm *Agent-Elect*

The following algorithm proceeds in phases, where in each phase, agents compete with each other to become the leader. An agent can be in one of the following states: *Active*, *Passive*, *Leader*, *Follower* or *Fail.* Each agent starts the algorithm in *active* state and knows the value of k at start. (We shall show later how to execute the same algorithm if the value of n is known instead of k.) The (encoded) Partial View of an agent A in phase i is denoted PV_{iA} and Agent-Count(A) denotes the number of homebases in the current territory of agent A.

```
Algorithm Agent-Elect
Execute EXPLORE to construct the territory T_A;
PV_0A ← COMPUTE-PV(T_A) ;
for phase i ← 1 to k {
   if (AgentCount(A) = k ) {
      State ← Leader ;
      SEND-ALL("Success"); Exit();
   }
   SEND-ALL(PV_iA, i);
   S ← RECEIVE-PV (i);
   State ← COMPARE-PV(PV_iA, S);
   if (State = Passive) {
      SEND-MERGE(i);
      SEND-ALL("Defeated", i);
      Return to homebase and execute WAIT();
   }else {
      RECEIVE-MERGE(i);
      execute UPDATE-PV() and continue;
   }
}
SEND-ALL("Failure"); Exit();
```

- **Procedure** SEND-ALL(M): During this procedure, agent A simply writes the message M on the whiteboard of each node in its territory.

- **Procedure** RECEIVE-PV(i): During this procedure, agent A visits each vertex u in its territory and traverses each NT-edge $e = (u, v)$ incident at u. On reaching the node v at the other end of the edge e, agent A waits till it finds the pair (PV_{iX}, i) written at node v (where PV_{iX} is the encoded Partial-View of some agent X which may or may not be same as agent A). Each Partial-View PV_{iX} that is read is added to the set S, which is returned at the end.

- **Procedure** COMPARE-PV(PV_{iA}, S): During this procedure, agent A compares its Partial-View PV_{iA} with those in the set S. If it finds any $PV_{iX} > PV_{iA}$, agent A changes its State to *Passive*. Else, for every PV_{iY} that is less than PV_{iA}, agent A stores the corresponding node v (where it was found) to the *Defeated-List*. The procedure returns the current state of the agent (Active or Passive).

- **Procedure** SEND-MERGE(i): During this procedure, the agent A returns to the node v where it found some Partial-View PV_{iX} that is greater than its own Partial-View PV_{iA}. On reaching node v, it writes (MERGE,i,$\lambda_v(e)$) on the whiteboard of node v, where e is the NT-edge joining v to T_A.

- **Procedure** RECEIVE-MERGE(i): During this procedure, agent A visits every node v in the *Defeated-List* and waits till it finds ("Defeated",i) on the whiteboard at node v. Finally agent A visits every node u in its territory and if it finds (MERGE,i,l) written at u, then the edge e having $\lambda_u(e) = l$ is re-marked as a T-edge (from both sides). In this case, we say that the territories at the two ends of edge e are merged.

- **Procedure** UPDATE-PV(): During this procedure an active agent updates its Partial-View and its territory as follows. For every edge e that it re-marked as T-edge during this phase, it finds the corresponding edge in its Partial-View and replaces the external node v incident on this edge with the Partial-View PV_{iX} that it read at node v. The new Partial-View obtained at the end of this procedure is called $PV_{(i+1)A}$ and the internal nodes in this partial view represents the new territory of agent A.

- **Procedure** WAIT(): This procedure is executed by a Passive agent A. The agent A simply waits at its homebase until it finds either "Success" or "Failure" written on the whiteboard. If it finds "Success", then State is changed to *Follower*; otherwise the State is changed to *Fail*, and the agent terminates the algorithm.

We have the following results showing the correctness of the above algorithm.

Definition 6.3.1 *(a) Γ_i denotes the set of agents that reach phase i of the algorithm in active state. (b) We say that the algorithm reaches phase i if at least one agent reaches phase i in active state.*

Lemma 6.3.2 *(a) During any phase i of the algorithm, the territories of the agents $\in \Gamma_i$ form a spanning forest of the graph G where each territory is a connected component of G. (b) For any phase i, if $|\Gamma_i| \geq 2$, and none of the agents in Γ_i become passive during phase i, then the AEP problem is unsolvable for the given instance (G, λ, p). (c) If at least one agent becomes passive in phase i then at least one agent reaches phase (i+1) in active state.*

Proof : The parts (a) and (c) follow directly from the description of the algorithm and the previous discussion. We prove part (b) here. Suppose no agent was defeated in some phase i. This would be possible only if all agents had exactly the same Partial-View PV_i at phase i. In that case, it is possible for each agent to construct the complete bi-colored view (to any depth) by repeatedly replacing each external node x in PV_i with an isomorphic copy of PV_i rooted at the node labelled same as the external node x. Hence, we can conclude that the view of each agent is identical in this case and thus the AEP problem is unsolvable for this instance. ∎

Theorem 6.3.1 *Given any instance (G, λ, p) of the AEP, algorithm AGENT-ELECT succeeds in electing a unique leader agent whenever (G, λ, p) is a solvable instance, and otherwise terminates with failure notification.*

Proof : Due to Lemma 6.3.2(b) and (c), if the instance (G, λ, p) is a solvable instance of the problem then in every phase i of the algorithm, (having $\Gamma_i > 1$) at least one of the agents would become passive and at least one agent would reach the next phase in active state. So, in some phase of the algorithm there would be only one active agent remaining say agent A. Due to Lemma 6.3.2(a), the territory of agent A would be a spanning tree of G and thus agent A would become the leader. Notice that no other agent can become the leader because the territories are disjoint and only the agent whose territory spans G can become leader. ∎

Theorem 6.3.2 *The algorithm AGENT-ELECT requires $O(m \cdot k)$ agent moves, in total, for any given instance (G, λ, p) where m is the number of edges in G and k is the number of agents.*

Proof : The procedure EXPLORE when executed by each agent requires a total of $O(m)$ agent moves while COMPUTE-PV() does not require any extra moves. Now let us determine the number of moves per phase of the algorithm. Each execution of SEND-ALL() requires $O(n)$ agent moves in total. During procedure RECEIVE-PV(), each NT-edge in G is traversed twice from each end, while each T-edge may be traversed twice. This accounts for $O(m)$ agent moves. Similarly RECEIVE-MERGE requires $O(m)$ moves while SEND-MERGE() requires $O(n)$. The other operations are executed locally. So, each phase of algorithm Agent-Elect requires $O(m)$ agent moves and there are at most k phases in the algorithm. ∎

Theorem 6.3.3 *The algorithm AGENT-ELECT requires $O(m \log n)$ memory at each node of the network, where m and n are the number of edges and nodes respectively.*

Proof : The memory available at the nodes should be sufficient for writing the label of that node and also for writing the partial view of an agent. An encoding of a partial view takes at most $O(m \log n)$ bits; $O(m \log \Delta)$ bits for the internal vertices (and edges), and $2(m-n)\log n$ bits for the external ones. Other than that $\log n$ bits are required for writing the label of a node. ■

For this algorithm, the value of k was used to detect termination(in case of failure), by comparing the phase number with k. In case the value of k is not known, but the value of n is known instead, then we can use the following equivalent termination condition. If an agent has at least n/r nodes in its territory (for some $r > 1$) and its territory has not changed for the last r phases, then it can terminate with failure notification to other agents. It is easy to show that in this case too, the algorithm would terminate after at most k phases.

We have the following lower bound on the cost of an effective election algorithm for mobile agents.

Lemma 6.3.3 *Any deterministic algorithm for effective leader election among k anonymous agents, dispersed in an arbitrary anonymous graph $G(V, E)$ with n nodes would require $\Omega(k \cdot n)$ edge traversals in the worst case, irrespective of the amount of memory available at the nodes.*

Proof : Consider the graph G that is a ring of n nodes with a bi-coloring b that places the k agents equidistant from each-other on the ring (i.e. k divides n, in this case). Let λ be a local orientation that labels each edge as '1' in the clockwise direction and as '2' in the anti-clockwise direction. Clearly, the agent election problem is not solvable for instance (G, λ, p) and thus any *effective* election algorithm $\mathcal{A}$ should detect this and terminate. We can show that algorithm $\mathcal{A}$ would make a total of $\Omega(k \cdot n)$ moves before it terminates. Notice that, since all the agents are in a symmetrical situation at the start of the algorithm, there exists an execution of $\mathcal{A}$, where this symmetry is maintained at every step of the algorithm and in such an execution, each agent would have to make $\Omega(n)$ moves, before it can obtain any information about the farthest node (at a distance $n/2$) from its homebase, in order to decide whether election is solvable or not for the given instance. ■

Thus from Theorem 6.3.2 and Lemma 6.3.3, we can say that the algorithm AGENT-ELECT is optimal in terms of agent moves, for sparse graphs (where $m \simeq n$).

6.3.3 Reducing the Size of the Whiteboards

Even though the number of agent moves used by algorithm AGENT-ELECT is quite efficient in terms of the number of agent moves; the memory requirements for the algorithm is much larger than that of algorithm *Compare-View* which uses constant memory. We would like to reduce the amount of memory required, without making an exponential number of agent moves. In

the following we propose some modifications to our algorithm, in order to reduce the amount of memory required at the whiteboards of the nodes. As a result the number of agent moves made by the algorithm is slightly increased, even though it is still polynomial in the size of the graph.

Algorithm Agent-Elect-2

We propose the algorithm *Agent-Elect-2* which is a modified version of the previous algorithm (AGENT-ELECT) and works as follows:

Algorithm *Agent-Elect-2*
Execute EXPLORE-2 to construct the territory T_A;
$PV_{0A} \leftarrow$ COMPUTE-PV(T_A) ;
for phase $i \leftarrow 1$ to k {
  if (AgentCount(A) $= k$) {
    State $\leftarrow$ Leader ;
    SEND-ALL("Success"); Exit();
  }
  SEND-ALL("Begin", i);
  $S \leftarrow$ RECEIVE-PV2 (i);
  State $\leftarrow$ COMPARE-PV(PV_{iA}, S);
  if (State = Passive) {
    SEND-MERGE(i);
    SEND-ALL("Defeated", i);
    Return to homebase and execute WAIT();
  }**else** {
    RECEIVE-MERGE2(i);
    execute UPDATE-PV() and continue;
  }
}
SEND-ALL("Failure"); Exit();

- The procedure EXPLORE-2() is similar to procedure EXPLORE with the only difference that instead of marking the edges as T-edge and NT-edge, at each node v a parent-link would be stored which would point to the parent of node v in the territory tree.
- The procedure RECEIVE-PV2() is different from RECEIVE-PV() in the following way. When an agent A executing RECEIVE-PV-2() reaches an external node v to read the Partial-View, it traverses the territory T_B that contains v (using the Parent-links), and computes the Partial-View PV_{iB}.

- The procedure RECEIVE-MERGE2() is again similar to RECEIVE-MERGE() with the following changes. Whenever an agent A has to merge its territory T_A with the territory T_B at other end of an external edge (u,v), agent A sets the Parent-link at node v to point to node u and then updates (i.e. reverses) the Parent-Link for each edge on the path from v to the root of T_B.

Lemma 6.3.4 *The output of procedure RECEIVE-PV2 is identical to the output of procedure RECEIVE-PV.*

Proof : During every phase $i > 0$ of the algorithm, each node contains the information *Parent*(v) which specifies the link to its parent node. Due to the presence of these child-to-parent links, once agent A enters a node x in the territory of B, it can completely traverse the tree containing x and thus obtain a map of the territory $\mathcal{T}_B$ of agent B. It can also traverse each non-tree edges connected to $\mathcal{T}_B$ and check the label of the node on the other side. Also, agent A can identify the root of $\mathcal{T}_B$, as the only (active) homebase inside the territory. Thus, the partial-view PV_{Bi} constructed by agent A would be same as that constructed by agent B in algorithm AGENT-ELECT. ■

The only difference between the two algorithms is that in the modified algorithm, an agent computes the Partial-View of its neighboring agents, instead of reading it from the whiteboard, as in the original algorithm AGENT-ELECT. Thus, the correctness of the algorithm *Agent-Elect-2* follows from the correctness of AGENT-ELECT, due to the above lemma.

Theorem 6.3.4 *The algorithm* Agent-Elect-2 *performs* $O(k \cdot m^2)$ *agent moves in total and requires* $O(\log n)$ *bits of node memory, in a network of* n *nodes and* m *edges, with* k *agents.*

Proof : As before, the algorithm terminates after at most k phases. In Step-1 of a phase, $O(n)$ traversals are made. In Step-2 of each phase, each tree $\mathcal{T}_A$ (and the non-tree edges connected to it) are traversed r times where r is the number of non-tree edges connected to $\mathcal{T}_A$. Thus, each edge of G may be traversed at most $m - n$ times. So, $O(m^2)$ edge traversals are required in each phase of the algorithm, amounting to a total of $O(k \cdot m^2)$ agent moves. As for the memory requirement, writing *Label*(v), *Parent*(v) and the phase number on the whiteboard requires $\log n$, $\log \Delta$, and $\log k$ bits of node memory, respectively. ■

Algorithm Agent-Elect-3

In order to further reduce the memory requirement of our algorithm, we can make the following modifications to algorithm *Agent-Elect-2*:

1. The procedure EXPLORE-2 is replaced by procedure EXPLORE-3, with the change that EXPLORE-3 would not explicitly write the labels of the nodes on the whiteboard of the nodes.

2. The procedure RECEIVE-PV2() would be replaced by procedure RECEIVE-PV3() where an agent A that is computing the Partial-View of a neighbor B and needs to read the label of a node x *external* to T_B, computes the label[4] by traversing the tree containing x.

3. During the execution of the algorithm, whenever an agent A has to write the phase number i on the whiteboard, it will write ($i \mod 3$) instead.

This new algorithm is called *Agent-Elect-3.*

Theorem 6.3.5 *The algorithm* Agent-Elect-3 *performs* $O(k \cdot n \cdot m^2)$ *agent moves in total and requires* $O(\log \Delta)$ *bits of node memory, in a network of* n *nodes,* m *edges, and maximum degree* Δ*, with* k *agents.*

Proof : In algorithm *Effective-Elect-2*, each non-Tree edge may be traversed at most $(m - n)$ times during a single phase. During any phase of algorithm *Effective-Elect-3*, corresponding to each traversal of a non-tree edge e in algorithm *Effective-Elect-2*, there is an extra traversal of the tree connected to e. Thus, at most $(n \cdot m^2)$ extra edge-traversals are made in a single phase of algorithm *Effective-Elect-3*, accounting for a total $O(k \cdot n \cdot m^2)$ agent moves. The algorithm only requires sufficient memory for writing the information $Parent(v)$ on the whiteboard of a node; thus $O(\log \Delta)$ bits suffice. ■

6.4 Summary of Results and Discussions

In this chapter, we first presented algorithms for the rendezvous problem in the whiteboard model, that solve the problem when $\gcd(n, k) = 1$. Our solution for this case requires $O(\log n)$ whiteboard memory and makes $O(m \log k)$ moves in total.

Next, we removed the restriction for n and k to be co-prime and design an algorithm for solving leader election (and thus Rendezvous), irrespective of the values of n or k, whenever the problem is solvable. Thus, we presented *effective* algorithms for solving leader election in the mobile agent model. Note that the extra cost for executing procedure GATHER, does not result in any increase in the asymptotic moves-complexity. Table 6.1 below shows the costs associated with the *effective* algorithms proposed here in comparison with the algorithm Compare-View from the previous chapter.

As for the complexity of the problem, our algorithms give an upper bound of $O(m \cdot k)$ agent moves for any *effective* solution to the agent election problem. A trivial lower bound for the

[4] The label assigned to a node v belonging to tree $\mathcal{T}$, is uniquely determined as the rank of the path to v from the root of $\mathcal{T}$, when compared with the paths to other nodes belonging to the same tree $\mathcal{T}$.

Algorithm	**Cost**		**Assumption**
	Edge-traversals	Node-Storage	
Agent-Elect	$O(k\ m)$	$O(m \log n)$	one of n or k is known
Agent-Elect-2	$O(k\ m^2)$	$O(\log n)$	
Agent-Elect-3	$O(k\ n\ m^2)$	$O(\log \Delta)$	
Compare-View	$O(k\ \Delta^{2n})$	$O(1)$	n is known

Table 6.1: Comparison of the cost (moves vs storage) for the proposed *effective* algorithms. Here, n, m and Δ are the number of nodes, number of edges and maximum degree of the graph and k is the number of agents.

same problem is $\Omega(n \cdot k)$ agent moves. Thus, it would be interesting to see if this gap between the bounds can be closed or at least narrowed, in future.

The analysis of the algorithms presented in this paper suggests that there is a possible trade-off between the two cost measures of agent moves and node memory, when solving the agent election problem. Using larger memory at the nodes allows the agents to communicate more information with each-other, thus reducing the number of moves they need to make. Another open question is whether it is possible to optimize both the number of agent moves and the amount of node memory. For instance, if we are allowed to use only a constant amount of memory at the nodes, is it still possible to achieve *effective* leader election with sub-exponential number of agent moves? If an efficient solution to the problem can be found that uses only a constant amount of whiteboard memory, then it may be possible to apply such an algorithm even in other computation models, where whiteboards are not available.

Chapter 7

Grouping and Near-Gathering

As seen in the previous chapters, it is not always possible to rendezvous all the agents in a network. In such cases, we may take two possible approaches. One approach could be to attempt the rendezvous of as many agents in the same location as possible. In fact, it may be possible to rendezvous the agent in multiple groups of smaller sizes (instead of a single group) — we call this the *Grouping* problem. Another approach is to try to gather the agents as close to each-other as possible. This problem is called *Near-Gathering.* In this chapter we study the following problems which can be viewed as relaxed versions of the *Rendezvous* problem:

- **Grouping**(w): The objective of this problem is to gather the agents in multiple groups, each of size at least $w \leq k$.
- **Near-Gathering**(w, d): The objective of this problem is to gather $w \leq k$ agents within a distance of d from each other. In a modified version of the problem called Maximal-Gathering(d), we need to gather within a distance of d as many agents as possible.

7.1 The Grouping Problem

Notice that the Rendezvous problem requires us to break the symmetry between the agents. If two agents are in symmetrical situations it is not possible to gather them in the same node unless there is some mechanism to break this symmetry. In case of the *Grouping* problem however we only require partial breaking of symmetry as opposed complete-symmetry-breaking required for solving *Rendezvous.* We show below the relation between Rendezvous and Grouping and also determine the necessary conditions for solving the Grouping problem.

7.1.1 Definitions and Properties

Definition 7.1.1 (Partial-Rendezvous) *The* p-Rendezvous*(w) problem requires* $1 < w \leq k$ *agents to gather at a single node of the network.*

Observation 7.1.1 *Grouping(w) is solvable in the network* (G, λ, p) *only if p-Rendezvous(w) is solvable in the same network.*

Definition 7.1.2 (Rendezvous-Number) *The* Rendezvous-number, $\mathcal{RV}num(G, \lambda, p)$ *of a network* (G, λ, p) *is the maximum value of w for which p-Rendezvous(w) is solvable in* (G, λ, p).

Definition 7.1.3 (Maximal-Rendezvous) *The* Maximal-Rendezvous *problem is to gather as many agents as possible to a single node of the network. In other words, solving maximal-rendezvous in a network* (G, λ, p) *requires solving p-Rendezvous for* $z = \mathcal{RV}num(G, \lambda, p)$ *agents.*

Lemma 7.1.1 *Given the network* (G, λ, p) *whose minimum base is* (D, μ_D) *and whose symmetricity is q, for any algorithm* $\mathcal{P}$*, there exists an execution of* $\mathcal{P}$ *on the network where the following happens: If agents* $A_1, A_2, ..., A_q$ *start from homebases* $v_1, v_2, ...v_q$ *which correspond to the q nodes that map to some vertex* $v \in D$*, then the agents would terminate in distinct nodes* $u_1, u_2, ...u_q$ *all of which map to the same vertex* $u \in D$.

Proof : Notice that for each vertex $v \in D$, there are exactly q nodes in the network that map to v. If v is a homebase and A is the agent starting from v, then let P_A be the sequence of edge-labels of the path traversed by agent A in D, during the execution of protocol $\mathcal{P}$. In the execution obtained by lifting this execution to the network (G, λ, p), each of the q agents corresponding to agent A would traverse the path defined by P_A and thus, they would reach distinct nodes $u_1, u_2, \ldots u_q$ which correspond to the vertex $u \in D$ that is the final destination of agent A. ∎

Lemma 7.1.2 *If q is the symmetricity of the network* (G, λ, p) *having k agents, then p-Rendezvous(w) is not solvable for* $w > (k/q)$ *agents.*

Proof : Suppose w agents are able to gather at a node, by executing some algorithm $\mathcal{P}$. Now, according to Lemma 7.1.1, for each one of these w agents, there are $q - 1$ other agents who would reach distinct nodes in the network. Thus $w \leq (k/q)$. ∎

Thus, we cannot solve Grouping(w) for $w > (k/q)$. However, as we show in the next section, Grouping(w) for $w = (k/q)$ can be solved in any network of symmetricity q. Hence we have the following results:

Theorem 7.1.1 *Grouping(w) is solvable in the network* (G, λ, p) *if only if p-Rendezvous(w) is solvable in the same network. This is possible for any* $w \leq (k/q)$.

Corollary 7.1.1 *The* Rendezvous-number, *$\mathcal{RV}num(G, \lambda, p)$ of a network (G, λ, p) is equal to (k/q), where q is the symmetricity of the network.*

7.1.2 Solution Protocol

We solve the Grouping problem as follows. Each agent first constructs the minimum-base (D, μ_D) of the network using the algorithm given below. Let v be the vertex with minimum label in (D, μ_D); An agent starting whose homebase node maps to vertex $u \in D$, simply moves along the path labelled α, where α is the sequence of edge-labels in the path from u to v in D.

To construct the minimum-base of the network, each agent executes the algorithm *ConstructMB* given below. This algorithm takes the value of k as input (any upper bound or even the value of n can also be used as input).

Algorithm 1: ConstructMB(k)

```
Mark the homebase;
Execute EXPLORE to construct territory T_A;
PV ← COMPUTE-PV(T_A);
h ← Root(T_A);
for i ← 1 to k do
    χ_i ← Encode(PV);
    Write (i, χ_i) on each node in T_A ;
    for each NT-edge e(u, v) in PV do
        Traverse e and read (i, χ'_i) from the other end v.
        if χ'_i < χ_i then
            Add v to Defeated-list;
        else if χ'_i > χ_i then
            Merge-Link ← e;
            State ← Passive ;
    for each node v in the Defeated-list do
        Go to node v and wait until Defeated(i) is written at v
    if State = Passive then
        Write MERGE(i,λ_v(e)) at node v, where e(u, v) is the Merge-Link;
        Write Defeated(i) on every node in T_A;
        Become Passive and wait at the homebase;
    < T_A, PV > ← Update-PV (i);
D ← Construct-Graph (T_A, PV);
Traverse T_A and communicate D to every passive agent;
return D;
```

Lemma 7.1.3 *The Map constructed by procedure Construct-Graph at the end of Algorithm 1 executed on the network (G, λ, p), represents the minimum-base of (G, λ, p).*

```
Procedure Update-PV(i)
  h ← Root(T_A);
  for each node u ∈ T that contains a MERGE(i,l) do
    Mark the edge labelled l as T-edge (from both sides);
  T ← { h } ⋃ { e(u,v) : e is marked T-edge and u ∈ T } ;
  // i.e.  T is the new territory defined by the T-edges.
  Count ← 0;
  for each node v ∈ T do
    if v is a homebase then
      Label(v):= (1,Count)
    else
      Label(v) ← (0,Count)
    Count ← Count + 1;
  Set PV ← T ;
  for each NT-edge e(u,v) incident to some node u ∈ T do
    Add the edge e and node v to PV ;
  return (T, PV);
```

```
Procedure Construct-Graph(T, PV)
  𝒟 ← an empty digraph;
  Vertex-set(𝒟) ← Vertex-set(T) ;
  for each edge e(u,v) ∈ T do
    Add arcs a1(u,v) and a2(v,u) to Arc-set(𝒟);
    Label(a1)← (λu(e), λv(e));
    Label(a2)← (λv(e), λu(e));
  for each NT-edge e(u,w) in PV, where w is an external vertex labelled y do
    Add an arc a(u,v) to Arc-set(𝒟) where v is the vertex labelled y in 𝒟;
    Label(a)← (λu(e), λv(e));
    if u ≠ v then
      Add arc a'(v,u) to Arc-set(𝒟,);
      Label(a')← (λv(e), λu(e));
  return H ← minimum-base of 𝒟 ;
```

The proof of the above lemma follows from the proof of Lemma 6.3.2 and the definition of minimum-base of a network.

7.2 Near-Gathering

Recall that the *Near-Gathering* problem, $\mathcal{G}$(w,d), $1 < w \leq k$, $d \geq 1$ requires w agents to gather within a distance of at most d from each other. We present some necessary conditions for solving the Near-Gathering problem $\mathcal{G}(w,d)$ for different values of w and d. We then show how to solve the problem and determine for what values of w and d, $\mathcal{G}(w,d)$ is solvable.

7.2.1 Necessary Conditions

In the following, (D, μ_D) is the minimum-base of the network (G, λ, p) and ϕ is the corresponding homomorphism that maps the vertices of G to the vertices of D.

Theorem 7.2.1 *Given the network (G, λ, p) with k agents, it is impossible to solve $\mathcal{G}(k, 1)$, unless there exists a vertex u in the minimum-base (D, μ_D) such the vertices $\in \phi^{-1}(u)$ form a clique of size q in G.*

Proof : To solve $\mathcal{G}(k, 1)$, every agent must be within a distance of one from each other. Due to Lemma 7.1.1, q of the agents would reach distinct nodes $u_1, u_2, \ldots u_q \in \phi^{-1}(u)$ for some vertex $u \in D$. Thus these nodes must be within a distance of one from each other, i.e. they must form a clique in the network. ■

The above result is for the gathering of all the agents. For partial gathering, we have the following results:

Theorem 7.2.2 *Given the network (G, λ, p) with k agents, it is impossible to solve $\mathcal{G}(r * (k/q), 1)$, for $r > 1$ unless there exists a vertex u in the minimum-base D such that*

(i) u has at least $r-1$ self-loops in D (i.e. $\exists$ arcs $a_1, a_2, \ldots a_{r-1}$ such that $s(a_i) = t(a_i) = u$), and

(ii) there is a clique of size r in G consisting of nodes all of which map to vertex $u \in D$.

Proof : Notice that out of the $r * (k/q)$ agents that have to gather, at most (k/q) may have homebases that map to distinct vertices in D. So, at least r of the agents have homebases that map to the same vertex, say $h \in D$. Due to Lemma 7.1.1, the final destinations of these r agents would be distinct nodes $u_1, u_2, \ldots u_r \in \phi^{-1}(u)$, for some vertex u in D. But nodes $u_1, u_2, \ldots u_r$ must be at distance of one from each-other. Hence u must have at least $r - 1$ self-loops and the nodes $u_1, u_2, \ldots u_r$ must form a clique in G. ■

Theorem 7.2.3 *Given the network (G, λ, p) with k agents, it is impossible to solve $\mathcal{G}(w, 2)$, for $w > (k/q)$ unless either there is a self-loop in the minimum-base (D, μ_D) or there are at least two parallel arcs in (D, μ_D).*

Proof : Suppose D has no self-loops. As there are more than (k/q) agents that gather, at least two of them (say agent A and agent B) would have homebases $h(A)$ and $h(B)$ that map to the same vertex in D and thus, due to Lemma 7.1.1 agents A and B would terminate in nodes u_1 and u_2 which map to the same node $u \in D$. Nodes u_1 and u_2 in G must be at a distance of at most 2. However, since u has no self-loops, this is only possible if there are a pair of parallel arcs at u. ■

Theorem 7.2.4 *Given the network (G, λ, p) with k agents and having minimum-base (D, μ_D), it is impossible to solve $\mathcal{G}(w, d)$, for $w > (k/q)$ and $d \geq 3$ unless one of the following holds: (i) There is a self-loop in D, (ii) There are at least two parallel arcs in D, or (iii) There is a cycle of length at most d in D.*

Proof : Using a similar logic as in the proof of Theorem 7.2.3, we can show that, in this case, there must exist two node u_1 and u_2 in G which map to the same node $u \in D$ and are at a distance of at most d. Suppose there are no self-loops or parallel arcs in D. Then the above condition can be satisfied only if there is a cycle of length at most d, connecting vertex u to itself, in D. ■

Theorem 7.2.5 *Given the network (G, λ, p) with k agents, if there are no self-loops or parallel arcs in the minimum-base (D, μ_D), then it is impossible to solve $\mathcal{G}(k, d)$ for $d < q * C/2$ where C is the length of the smallest cycle in D.*

Proof : It follows from the proof of theorem 7.2.2 that in this case, there must be q agents that terminate in distinct nodes $u_1, u_2, \ldots u_q \in \phi^{-1}(u)$, for some vertex u in D. Also, due to Theorem 7.2.4, we know that there must exist a cycle in D containing u. Let C be the length of the smallest such cycle in D. Then the smallest distance between the farthest nodes among $u_1, u_2, \ldots u_q$ is $q * C/2$. Hence the result follows. ■

7.2.2 Solution and Sufficient Conditions

The solution of the Near-Gathering problem depends on the structure of the minimum-base (D, μ_D), e.g. the number of self-loops, cycles or parallel arcs present. Any solution algorithm for $\mathcal{G}(w, d)$ would start with construction of the minimum-base (using Algorithm 1) and thereafter every agent would pick a particular vertex $u \in D$ as the *gathering-point* and move to one of the nodes in the network, that map to u. The selection of the vertex u would depend on the

structure of D and the parameters w and d. Based on this general approach, we obtain the following results on solvability of the Near-Gathering problem.

Lemma 7.2.1 *If there is a symmetric self-loop in D (i.e. a self loop with the same labels on both ends), then $\mathcal{G}(2(k/q), 1)$ is solvable.*

Proof : Let a be any symmetric self-loop in D with $s(a) = t(a) = u$, then arc a corresponds to exactly $q/2$ edges in the network which connect the nodes $\in \phi^{-1}(u)$, in pairs. If after constructing the minimum-base using Algorithm 1, every agent moves to one of the nodes that map to u, then at least one of these $q/2$ edges would have a total of $2(k/q)$ or more agents in its two end-points combined. ■

In fact, the above result would hold for any self-loop (either symmetric or asymmetric). When there are more than one self-loop at a vertex of the minimum-base, we get the following result:

Lemma 7.2.2 *If there is a vertex $u \in D$ with exactly r self-loops (i.e. arcs $a_1, a_2, \ldots a_r$ such that $s(a_i) = t(a_i) = u$) then $\mathcal{G}((r+1)(k/q), 2)$ is solvable.*

Proof : If there is a vertex $u \in D$ with exactly r self-loops, then each node in $\phi^{-1}(u)$ would be directly connected r other nodes in $\phi^{-1}(u)$. (The distance between any two of these $r+1$ agents is at most two.) Hence, if the agents pick vertex u as the gathering-point then $\mathcal{G}((r+1)(k/q), 2)$ would be solved. ■

Lemma 7.2.3 *If there is an asymmetric self-loop in D (i.e. a self loop with distinct labels on each end), and $q < 6$ then $\mathcal{G}(k, \lceil q/2 \rceil)$ is solvable.*

Proof : If a is an asymmetric self-loop in D with $s(a) = t(a) = u$, then the corresponding edges in the network would form one or more disjoint cycles containing only the nodes $\in \phi^{-1}(u)$. Since there are q nodes in $\phi^{-1}(u)$, if $q < 6$ there would be exactly one cycle of length q containing all the nodes which map to u and the maximum distance between any two such nodes would be $\lceil q/2 \rceil$. ■

Lemma 7.2.4 *If there are at least two parallel arcs in D and $q < 4$ then $\mathcal{G}(k, q)$ is solvable.*

Proof : If $q < 4$ and there are two parallel arcs joining vertices u and v in D then, the corresponding edges in the network would form a cycle of length $2q$ containing all the nodes of the network that map to either u or v. So, if the agents pick either u or v as the gathering-point then $\mathcal{G}(k, q)$ would be solved. ■

Lemma 7.2.5 *If there is a vertex $u \in D$ with exactly $q-1$ self-loops (i.e. arcs $a_1, a_2, \ldots a_{q-1}$ such that $s(a_i) = t(a_i) = u$) then $\mathcal{G}(k, 1)$ is solvable.*

Proof : If there is a vertex $u \in D$ with exactly $q-1$ self-loops, then the nodes in $\phi^{-1}(u)$ form a clique in the network. Hence, if the agents pick vertex u as the gathering-point then $\mathcal{G}(k,1)$ would be solved. ■

Lemma 7.2.6 *If there are no self-loops or parallel arcs in D and there is exactly one cycle in D then $\mathcal{G}(k, \lceil C.q/2 \rceil)$ is solvable, where C is the length of the only cycle in D.*

Proof : If there are no self-loops or parallel arcs in D and there is exactly one cycle consisting of vertices $u_1, u_2, \ldots u_C$, then this cycle corresponds to a cycle of length $C.q$ in the network, consisting of all the nodes that map to one of the above vertices. Hence, we can solve $\mathcal{G}(k, \lceil C.q/2 \rceil)$ by picking any of the vertices $u_1, u_2, \ldots u_C$ as the gathering-point. ■

Recall that the d-maximal-gathering problem involves solving $\mathcal{G}(w,d)$ for the maximum possible value of w. Based on the previous results on near-gathering, we have the following theorem on 1-maximal-gathering.

Theorem 7.2.6 *It is possible to solve* 1*-maximal-gathering in (G, λ, p) if either of the following holds:*

(i) There is no vertex $v \in D$ with more than one self-loops, or

(ii) There exists a vertex $u \in D$ with exactly $q-1$ self-loops.

Proof : In case (i) there are two possibilities, either there are no self- loops or there are one or more vertices with a single self-loop each. In the former case, at most (k/q) agents can gather at distance of 1, due to Theorem 7.2.3 and this is achieved by algorithm 1. Otherwise, there is a vertex v with a single self-loop and thus we can solve $\mathcal{G}(2(k/q),1)$ due to Lemma 7.2.1, which is maximal. The result for case (ii) follows from Lemma 7.2.5. ■

7.3 Conclusions

In this chapter, we studied the Grouping and Near-Gathering problems for networks with symmetricity $q > 1$. We showed that Grouping(w) can be solved for $w \leq k/q$, where k is the total number of agents. We solved the problem by first constructing the minimum base of the network using Algorithm 1 which is based on the Algorithm Agent-Elect seen previously and has the same complexity.

For the Near-Gathering problem, we do not have a complete characterization of networks where the problem is solvable. We presented various necessary and sufficient conditions for solving Near-Gathering(w,d) for different values of w and d, based on the structure of the minimum-base of the network. However, there are gaps between the necessary and sufficient conditions in most cases. Only for the case of $d = 1$, we know that the necessary and sufficient

condition for solving Near-Gathering$(k, 1)$ is the presence of $q - 1$ self-loops at some vertex of the minimum-base. In case there are no more than a single self-loop at any vertex of the minimum-base, we can only solve Near-Gathering$(2k/q, 1)$. For $d = 0$, Near-Gathering(w, d) reduces to the Grouping(w) problem.

Chapter 8

Rendezvous in Faulty Networks

In this chapter, we consider the problem of solving rendezvous in networks where some of the nodes or channels in the network are faulty. A faulty node is one which is dangerous for the agents, such any agent that visits this node is destroyed. Such a node is called a *Black-Hole.* We also consider faulty channels or edges, such that any agent that traverses such an edge, dies. Notice that a faulty edge is equivalent to a black hole of degree two. We consider networks containing both black holes and faulty edges. We assume that the faulty nodes and edges do not disconnect the network i.e. we assume that the safe part of the network is connected and every agent initially starts from some safe node. Since the location of the fault could be unknown to the agents, we cannot prevent some of the agents from falling into a black hole and dying. Thus, the objective here is to gather as many agents as possible in some safe node of the network.

The only known results for rendezvous in networks containing faults is in the case of the ring network, containing exactly one black hole [52]. In this chapter, we generalize some of these results to networks of arbitrary topology with arbitrary number of faulty nodes and links. We not only determine how many agents can rendezvous in any given network containing black holes, but also compute the exact conditions under which the maximum number of agents can rendezvous. We present algorithms for solving rendezvous in such cases. Our generic results, when applied to the ring networks coincide with the earlier results of Dobrev et al.[52]. This means that the knowledge of topology is not necessary in solving rendezvous in faulty networks, provided that certain parameters (e.g. the size) for the network are known.

8.1 Some Definitions and Properties

We represent a network as (G, λ, p, η) where G, λ, p are as defined before. The function η : $E(G) \rightarrow \{0, 1\}$ denotes which edges are safe/faulty. An edge $e \in E(G)$ is safe if $\eta(e) = 1$ and

faulty otherwise. The faults are permanent, so any edge or node that is faulty at the start of the algorithm remains so until the end and no new faults appear during the execution of the algorithm. If a node v is a black hole then all the edges incident to v are considered to be faulty edges. We assume that the safe part of the network is connected, i.e. there is a path of safe edges between any two safe nodes in the network. The incident links or ports at a (safe) node which lead to a faulty node or edge (i.e. that are dangerous for an agent) are called dangerous-links. The number of dangerous-links is denoted by τ and the number of safe nodes is denoted by n_s.

Given a network (G, λ, p, η) with k agents, when we say that an algorithm $\mathcal{P}$ is being executed on the network, we mean that every agent starting from its homebase, begins performing (not simultaneously but each starting at any arbitrary instance) the steps of the algorithm $\mathcal{P}$, e.g. reading, writing, computing and moving, until the agent either reaches the terminal state or dies. The algorithm is said to have terminated when each surviving agent has reached terminal state. An algorithm for rendezvous may have either local termination or global termination. If each agent terminates after reaching its final destination, without waiting for any other agent to arrive, then we have local termination. However, if an agent terminates only when all its partners have arrived at the rendezvous location, then we have global termination.

We present some definitions for the rendezvous problem in faulty networks:

Definition 8.1.1 Rendezvous*: The rendezvous problem, $\mathcal{RV}(w)$ requires $1 < w \leq k$ agents to gather at a single (safe) node of the network. The* Rendezvous with detect *problem, $\mathcal{RVD}(w)$ requires that at least w agents gather at a safe node, if it is deterministically possible and otherwise all surviving agents terminate within a finite time and report that the problem is not solvable.*

Definition 8.1.2 Rendezvous-Number*: The rendezvous-number, $\mathcal{RV}num(G, \lambda, p, \eta)$ of a network (G, λ, p, η) is the maximum value of w for which $RV(w)$ is solvable in (G, λ, p, η).*

Definition 8.1.3 Maximal-Rendezvous*: The maximal-rendezvous problem is to gather as many agents as possible to a single node of the network. In other words, solving maximal-rendezvous in a network (G, λ, p, η) requires solving $\mathcal{RV}(z)$ for $z = \mathcal{RV}num(G, \lambda, p, \eta)$.*

8.2 Characterizations based on rendezvous problem

In this section we determine the conditions under which the rendezvous problem can be solved in given network (G, λ, p, η) containing faulty nodes and edges.

We define the *extended-view* of the network (G, λ, p, η) as the labelled digraph (H, μ_H) such that, H consists of two disjoint vertex sets V_1 and V_2 and a set of arcs $\mathcal{A}$ as defined below:

- $V_1 = \{v \in V(G) : v \text{ is safe. }\}$;
- $\mu_H(v) = p(v)$, $\forall v \in V_1$;
- For every safe edge $e = (u, v) \in E(G)$, there are two arcs $a_1, a_2 \in \mathcal{A}$ such that $s(a_1) = t(a_2) = u$, $s(a_2) = t(a_1) = v$, and $\mu_H(a_1) = (\lambda_u(e), \lambda_v(e))$, $\mu_H(a_2) = (\lambda_v(e), \lambda_u(e))$.
- For every faulty edge $e = (u, v)$ connecting two safe nodes, there are vertices u' and $v' \in V_2$ with $\mu_H(u') = \mu_H(v') = -1$ and arcs $(u, u'), (u', u), (v, v')$ and $(v', v) \in \mathcal{A}$ with labels $(\lambda_e(u), 0)$, $(0, \lambda_e(u))$, $(\lambda_e(v), 0)$, and $(0, \lambda_e(u))$ respectively;
- For every edge $e = (u, v)$ connecting a safe node u with a faulty-node v, there is a vertex $v' \in V_2$ with $\mu_H(v') = -1$ and arcs (u, v') and $(v', u) \in \mathcal{A}$ with labels $(\lambda_e(u), 0)$, $(0, \lambda_e(u))$ respectively;

We now define the *determinant-graph* D of the network (G, λ, p) in terms of its extended-view. The determinant-graph D of a network (G, λ, p) is the symmetric digraph (D, μ_D) that is the minimum base of (H, μ_H) the extended-view of the network. Note that (D, μ_D) would be a multi-graph, possibly containing self-loops and parallel arcs. We define the quantity $q = |H|/|D|$ to be the *quasi-symmetricity* of the network (G, λ, p).

Lemma 8.2.1 *For any network (G, λ, p) the value of the quasi-symmetricity q is such that $1 \leq q \leq \gcd(n_s, \tau, k)$ and q divides $\gcd(n_s, \tau, k)$.*

Proof : Notice that according to the definition, $n_s = q \times |\{v \in D : \mu_D(v) \neq -1\}|$ where (D, μ_D) is the Determinant graph of the network. Thus, q must divide n_s. For similar reasons q must also divide τ and k. Hence the result follows. ∎

Lemma 8.2.2 *Given the network (G, λ, p) whose extended view is (H, μ_H) and determinant graph is (D, μ_D), and any deterministic (distributed) algorithm $\mathcal{P}$, the following holds:*

1. *Any execution of algorithm $\mathcal{A}$ on the network (G, λ, p) can be simulated on the digraph (H, μ_H), in such a way the outcome of the execution on (H, μ_H) is identical to that of the execution on the original network.*
2. *There exists an execution of algorithm $\mathcal{P}$ on (D, μ_D) (with k/q agents) which corresponds to a particular execution of algorithm $\mathcal{P}$ on (G, λ, p). This means that if in the later execution an agent moves from node v_i to node v_j during any step, then in the former execution during the same step, an agent moves from node $\phi(x_i)$ to $\phi(x_j)$ (where nodes v_i and v_j of the network correspond to the vertices x_i and x_j in H).*

3. *In the above execution of algorithm $\mathcal{P}$ on the network (G, λ, p), if agents $A_1, A_2, ..., A_q$ start from homebases $v_1, v_2, ...v_q$ which correspond to the q nodes that map to some vertex $v \in D$, then the agents would terminate in distinct nodes $u_1, u_2, ...u_q$ all of which map to the same vertex $u \in D$.*

Proof : (1) The simulation works as follows. Initially, each vertex labelled 1 in (H, μ_H), is considered to be hosting an agent process. Whenever during the execution of the algorithm, an agent moves from one safe node v_i to another safe node v_j, in the simulation the corresponding agent at vertex x_i is assumed to have moved to vertex x_j. However, when an agent traverses a link leading to a black hole; in the simulation the corresponding agent moves along the corresponding arc and once it reaches a vertex labelled 0, it is immediately terminated and does not participate in the algorithm anymore. Notice that during the algorithm, if two (or more) agents reach a node simultaneously, only one agent would get access to the whiteboard, while the others would wait. So, the agents at a node execute in some order; thus, in the simulation too, the actions of the agents would be executed in the same order. It is easy to see that the output obtained from such a simulation would correspond to the outcome of the original execution of algorithm $\mathcal{P}$ on the network.

(2) This follows from Property 4.2.2 and point (1) above.

(3) Notice that for each vertex $v \in D$, there are exactly q nodes in the network that map to v. If v is a homebase and A is the agent starting from v, then let P_A be the sequence of edge-labels of the path traversed by agent A in D, during the execution of protocol $\mathcal{P}$. In the execution obtained by lifting this execution to the network (G, λ, p), each of the q agents corresponding to agent A would traverse the path defined by P_A and thus, they would reach distinct nodes $u_1, u_2, \ldots u_q$ which correspond to the vertex $u \in D$ that is the final destination of agent A. ■

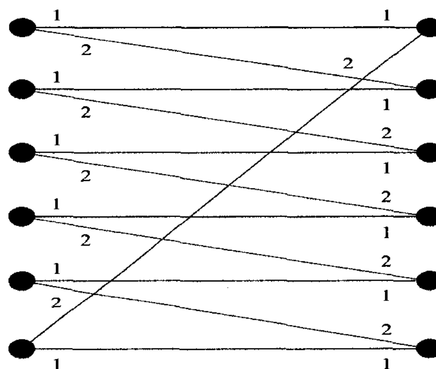

Figure 8.1: A network where $n = k = 4r$ and there are r faulty edges (not shown in the figure)

Lemma 8.2.3 *In a network containing τ dangerous links and k dispersed agents, τ agents may die while executing any algorithm for rendezvous. Thus, it is not always possible to rendezvous more than $k - \tau$ agents even if the network topology is known to the agents.*

Proof : Consider a network of size $4 \cdot r$, $r \geq 1$ as shown in Figure 8.1 (here $r = 3$). Each node of the network contains one agent initially. Suppose there are r faulty edges (i.e. $\tau = 2r$) in the network but the location of the faulty edges is unknown to the agents (thus an adversary can decide which edges are faulty). Suppose all the agents wake-up at the same time; Since all these agents are in the same state initially, they would take the same action. So, either each agent would traverse the incident edge labelled 1 or each agent would traverse the edge labelled 2 (They cannot just wait because then we would have a deadlock!). In the first case, we assume all the faulty edges are labelled $(1,1)$; and in the second case, we assume the faulty edges are labelled $(2,2)$. Thus in both cases, at least τ agents would die. ∎

Theorem 8.2.1 *It is impossible to deterministically solve $\mathcal{RV}(z)$ for $z > (k-\tau)/q$ in a network (G, λ, p) with k agents, where q is the quasi-symmetricity of the network.*

Proof : Due to Lemma 8.2.3 above, τ agents may die while executing the algorithm. Suppose r of the remaining agents are finally able to gather at a node. Now, according to Lemma 8.2.2, for each one of these r agents, there are $q - 1$ other agents who would reach distinct nodes in the network. Hence $r \leq (k - \tau)/q$. ∎

Thus, $\mathcal{RV}num(G, \lambda, p) \leq (k - \tau)/q$ for any arbitrary network (G, λ, p). We shall show in a later section that these many agents can indeed rendezvous, and thus the inequality in the above equation can be replaced by an equality.

8.3 Initial Knowledge Required

We now determine what initial knowledge is required by the agents to solve rendezvous in faulty networks.

Theorem 8.3.1 *It is impossible to solve* Maximal Rendezvous *in a network (G, λ, p, η) if the value of n_s and k are both unknown to the agents.*

Proof : Consider the networks shown in Figure 8.2, where G_1 and G_2 are (distinct) arbitrary graphs, each of size $r > 1$. Each black node contains a single agent initially. The dotted lines represent faulty edges and the edges marked by arrows (in the first network, Figure 8.2(a)) are slow edges. Notice that all three networks are minimal. The size of the first network is $2r + 8$, whereas the networks in (b) and (c) have size $r + 4$ each. However, if no prior knowledge of n_s or k is available to the agents, then the agents can not distinguish between the three networks.

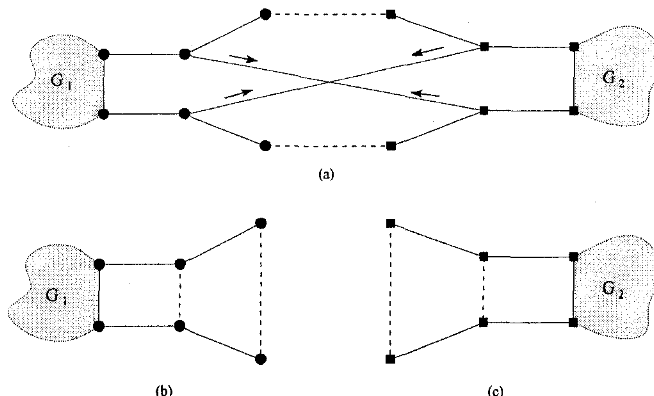

Figure 8.2: The network in (a) cannot be distinguished from the networks in (b) and (c) due the slow edges (marked by arrows)

In networks (b) and (c), the rendezvous of only two agents is possible, so in network (a) too, the algorithm would terminate after rendezvous of only two agents even though rendezvous of $(k - \tau)/q = 8$ agents is possible for the network (a). ■

Theorem 8.3.2 *It is impossible to solve maximal rendezvous in a network* (G, λ, p, η) *if the value of the quasi-symmetricity* q *is unknown to the agents (even if the network topology is known).*

Proof : Consider the two networks shown in Figure 8.3(a) and (b). Each network has the same topology and there are three faulty edges in each (but their locations are different). There are $k = 12$ agents in each network (their homebases are the dark nodes in the figure). Notice that for the first network $q = 2$ (the minimum base is shown in Figure 8.3(c)), whereas for the second network $q = 1$. In the network of Figure 8.3(b) there are two edges (marked by arrows) which are very slow. Since slow edges edges can not be distinguished from faulty edges, the agents would not be able to determine whether they are in the first network or the second. Thus any rendezvous algorithm $\mathcal{P}$ (that terminates within a finite time) must achieve the same result in both networks. Since algorithm $\mathcal{P}$ must fail to achieve rendezvous of $(k - \tau) = 6$ agents in the first network, it must also fail in the second one. However it is possible to solve rendezvous of $(k - \tau) = 6$ agents in the second network. Thus algorithm $\mathcal{P}$ fails to solve maximal-rendezvous in the second network. ■

Theorem 8.3.3 *It is impossible to solve* Maximal Rendezvous *in the network* (G, λ, p) *unless the value of* τ *is known to the agents.*

Proof : Suppose the rendezvous number $\mathcal{RV}num(G, \lambda, p, \eta) = z$. Consider an execution of any rendezvous algorithm $\mathcal{P}$ on the network, such that all agents traversing one particular

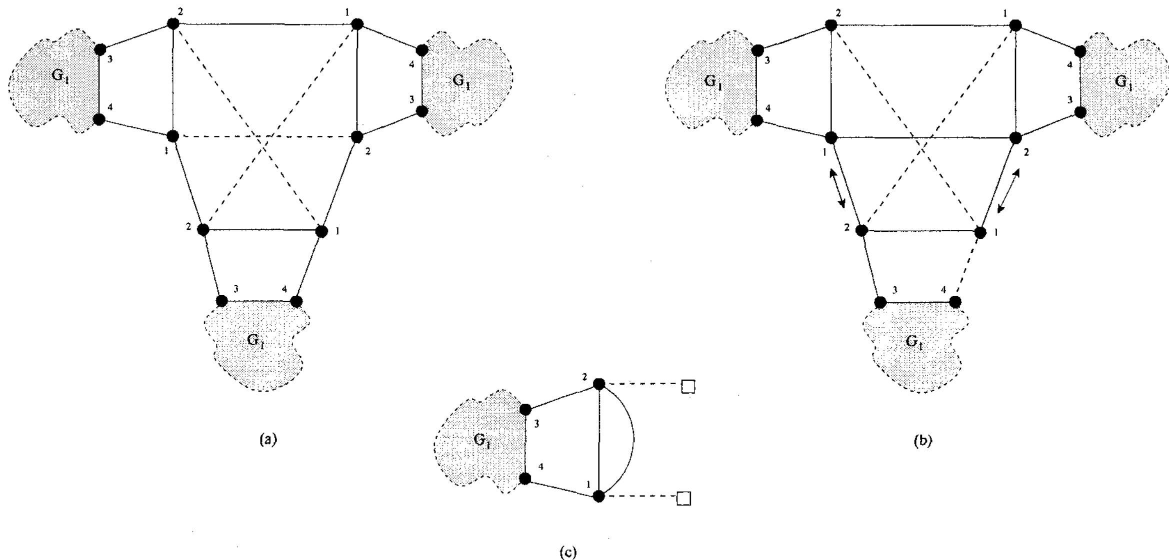

Figure 8.3: The networks in (a) and (b) have same topology but differ in the location of faulty edges (shown by dashed lines). (c) The minimum base for the network in (a).

edge e are blocked by the adversary (i.e. e is a slow edge). So, at most $z-2$ may be able to rendezvous while at least two agents are blocked at edge e. At this point, should the algorithm terminate and declare failure or, should it wait for the blocked agents? If the value of τ is unknown then the algorithm would not be able to take the right decision and thus may not be able to solve maximal rendezvous. ■

8.4 Solution Protocols

The protocol for solving the rendezvous problem that we present below, uses the *cautious-walk* technique as in [53], to prevent two agents from falling into the black hole through the same link. The technique is as follows—whenever any agent traverses an unexplored edge $e=(u,v)$ incident at a node u, it marks e as "dangerous" from node u's end; On reaching the other end v, it immediately returns to node u and re-marks the edge as "safe". By employing the rule that no agent should ever traverse an edge marked as "dangerous", it can be ensured that no more τ agents eventually fall into black holes.

In our algorithm, the edges would be marked as either "dangerous" or, "T-edge" (i.e. tree-edge) or, "NT-edge" (i.e. non-tree edge). Any unmarked edge is considered to be unexplored. Initially each edge starts a depth-first traversal from its homebase, using *cautious-walk* and marking the node and edges traversed, to construct a tree (rooted at the homebase), which

we call the agent's territory. This algorithm is presented as Procedure Cautious-Explore. The algorithm ensures that the territories of the agents are disjoint and together they cover all the safe nodes in the network. The algorithm also constructs the Partial-View of an agent, called PV which is obtained from the territory T, by adding certain external vertices and edges to T. These external vertices and edges represent the dangerous links as well as the NT-edges incident to nodes in T.

During the algorithm RendezVous(w) given below, each agent tries to expand its territory by defeating other agents and acquiring their territories or by exploring new edges. The defeated agents join the territory of the winning agent. The algorithm terminates when at least w agents are present in the same territory.

Procedure `Cautious-Explore`

```
Num ← 1;
Path ← {};
Mark the current node u with Num and increment Num;
PV ← T ← {u} ;
repeat
    while there is an unexplored edge e at current node u do
        Mark e as dangerous ;
        traverse e to reach node v ;
        if v is marked explored then
            Go back to node u and re-mark e as NT-edge;
            add edge e and node v to PV; // These are the external vertices and
                edges.
        else
            Mark v with Num and increment Num;
            add λ(e) as the parent link at node v ;
            go back to node u and re-mark e as T-edge;
            add edge e and v to T and PV; // These are the internal vertices
                and edges.
            return back to node v;
            Add λ(e) to Path ;
    if Path is not empty then
        remove the last edge-label from Path and backtrack on that edge;
until Path is empty;
return (T, PV) ;
```

Theorem 8.4.1 *Algorithm RendezVous(w) always achieves the rendezvous of w agents in the network (G, λ, p, η), if $w \leq \mathcal{RV}num(G, \lambda, p, \eta)$ (i.e. if rendezvous of w agents is possible). Otherwise the algorithm may not terminate.*

Algorithm 5: RendezVous(w)

$< T, PV > \leftarrow$ `Cautious-Explore` () ;
$h \leftarrow$ Root(T);
for $i \leftarrow 1$ *to infinity* **do**
- $\chi_i \leftarrow$ Encode(PV);
- Write $(COMPETE, i, \chi_i)$ on each node in T ;
- **for** *each NT-edge* $e(u, v)$ *in* PV **do**
  - Traverse e and read (S, j, χ') from the other end v.
  - **if** $S = EXPLORE$ *or* $j < i$ *or* $(j = i$ *and* $\chi' < \chi_i)$ **then**
    - Go to root r_v of the tree containing v;
    - **if** r_v *is not marked 'CAPTURED'* **then**
      - Write CAPTURED at r_v;
      - Add (e, r_v) to Defeated-List;
- Return to node h;
- **for** *each entry* (e, r) *in the Defeated-List* **do**
  - Mark e as T-edge;
  - Go to node r;
  - **if** *State(r) is COMPETE* **then** Wait for the owner of node r;
  - Flip the parent links of all nodes on the path from r to e;
  - Wake-up any sleeping agents at r and move them to node h;
- $T_{old} \leftarrow T$;
- $< T, PV > \leftarrow$ `Update-PV` () ;
- **if** *Number of homebases in* $T \geq w$ **then**
  - Wake-up all sleeping agents in T;
  - **return** *"Success"* ;
- **if** h *is marked 'CAPTURED'* **then**
  - Wait for winner agent W;
  - $T \leftarrow$ Tree owned by W;
  - $h \leftarrow$ Root(T);
  - Go to the node h and execute `Check-New-Edges` ();
- **else if** $T = T_{old}$ **then**
  - Write $(EXPLORE, i, 0)$ on every node in T;
  - **if** *there is any unexplored edge* e *incident to* T **then**
    - `Check-New-Edges` ();
  - **else**
    - Sleep until woken-up;
  - $< T, PV > \leftarrow$ `Update-PV` ;

Procedure `Check-New-Edges`

```
if There exists an unexplored edge e incident to T(h) then
    Update T(h) to mark e as Dangerous;
    Traverse edge e to reach node v;
    if v is unexplored then
        mark e as T-edge;
    else
        mark e as NT-edge;
    Go to node h;
    while State(h)=CAPTURED do
        Go to new root r;
        h ← r;
    Update T(h) to change status of edge e;
if State(h)=COMPETE then Sleep until woken-up;
```

Procedure `Update-PV`

```
if homebase-node h is marked 'CAPTURED' then
    Traverse the parent-links from h to reach the new root r;
    h ← r;
    Increment the agent-count at node h;
T ← tree rooted at node h (as defined by the T-edges);
// i.e.  T is the new territory.
Traverse T and re-label the nodes of T;
PV ← T ;
for each NT-edge e(u, v) incident to some node u ∈ T do
    Add the edge e and node v to PV ;
for each dangerous link l incident to some node u ∈ T do
    Add new node x and an edge e(u, x) to PV;
    Label(x) ← −1; Label(e) ← (l, −1);
for each unexplored link l incident to some node u ∈ T do
    Add new node y and an edge e(u, y) to PV;
    Label(y) ← 0 and label(e) ← (l, 0);
return (T, PV);
```

Notice that the algorithm 5 does not require any prior knowledge of the network. However, the algorithm does not terminate for invalid input values. As shown previously, any solution to *Rendezvous with Detect* or *Maximal-Rendezvous* requires the prior knowledge of either n_s or k along with the values of q and τ. When k, q and τ are all known, it is possible to compute the rendezvous-number for the network. Thus, we can solve *Rendezvous with Detect* (as shown in the algorithm 8) when the values of k, q and τ are known to the agents.

Algorithm 8: RVwithDetect(w, k, q, τ)

```
RV-number ← (k − τ)/q ;
if w ≤ RV-number then
|  Execute RendezVous (w);
else
|_ Terminate with Failure;
```

8.5 Conclusions

We studied the problem of rendezvous in networks of arbitrary topology in the presence of both faulty nodes (i.e. black holes) and faulty edges, assuming that the faults do not disconnect the safe part of the network. We determined the maximum number of agents that may rendezvous in any given network, and we present solution protocols for achieving the same. We also determine what conditions are necessary to solve the rendezvous problem; in particular what prior knowledge is necessary for solving rendezvous in arbitrary networks. Some of our results are generalizations of the results presented in [52], extended to arbitrary networks with multiple black-holes instead of just one. The table below provides a comparison of their results (in row-1) with our results (in row-2). Here, t denotes the number of black-holes in the network.

Network Topology	**Possibility Results**		**Assumptions**
	for odd n_s, k	when n_s or k is even	
Ring Network $t = 1$	$\mathcal{RV}(k-2)$	$\mathcal{RV}(k-2/2)$	Topology known n, t known
Arbitrary $t \geq 1$	$\mathcal{RV}((k-\tau)/q)$ [$q = 1, \tau = 2$ **for Ring**]	$\mathcal{RV}((k-\tau)/q)$ [$q \leq 2, \tau = 2$ **for Ring**]	Topology Unknown n_s, q, τ known

Table 8.1: Comparison of results for the rendezvous problem in faulty networks

Chapter 9

Computing with Faulty Mobile Agents

The mobile agent systems that we considered until now were assumed to be completely reliable. In Chapter 8 we studied mobile agent systems where the underlying network was faulty. In this chapter we consider the scenario where the agents themselves may be faulty, i.e. they may crash. An *agent crash* happens when an agent suddenly dies or disappears while moving from one node to another. This kind of fault commonly occurs in many networked environments due to various reasons, e.g. due to network congestion, or unavailability of a host. When studying agent crashes, we will assume that the hosts in the network are reliable (i.e. they do not crash). So, an agent is assumed to be safe once it has safely arrived at a node; however it may die while moving between nodes. In this chapter, we study the effects of agent crashes on the computational power of a mobile agent system. In fact we show that any distributed computation that can be performed in a stationary agent system can be simulated in a mobile agent system even in the presence of agent crashes. Thus, the computational power of such systems is not affected by agent crashes, unless all the agents crash down.

The mobile agent model considered in the chapter is same as the one discussed before (see Chapter 2). However, we make the following additional assumptions:

A1 Each communication link (represented by an edge of the graph G) satisfies the first-in-first-out property; So, if agent B leaves node v through the edge $e(v, w)$ after agent A left through the same edge, then agent B would arrive at node w after agent A arrives there.

A2 An agent can die while traversing an edge or on arrival at a node, but it cannot die while performing some computation at a node.

A3 At most $f \leq k-1$ agents can crash. Once an agent crashes, it may not become alive

again (i.e. agent crashes are permanent).

As mentioned before, the simulation of message-passing algorithms in the mobile agent model has been studied earlier by Chalopin et al. [32]. However the simulation algorithm proposed in that paper is applicable only in fault-free environments where agents do not crash. Further, there is no explicit termination detection for the simulation. (This is due to the absence of information about the network size or the number of agents). In contrast, the algorithms proposed in this chapter terminate explicitly, even if there is no explicit termination detection in the simulated message-passing algorithm.

9.1 Simulating a Distributed Algorithm

In this section, we assume the nodes of the network have distinct identifiers. The case of anonymous networks will be considered separately in the next section.

Consider an arbitrary (distributed) algorithm—we shall call it algorithm Z—being executed on a network (G, λ), in the conventional stationary-agent (S.A.) model. Such an algorithm can be described as follows:

Algorithm Z

If state = "Initiator", then execute the following steps:

Step-1 Initiate the algorithm and do local computation(possibly generating messages to send)

Step-2 Send zero or more messages through some of the ports.

Step-3 Wait for messages to arrive from one or more ports.

On receiving any message m, execute the following steps:

Step-1 Read message m and do local computation(possibly generating messages to send)

Step-2 Send zero or more messages through some of the ports.

Step-3 Wait for messages to arrive from one or more ports.

The algorithm is said to have terminated when every node is in Step-3 (passive mode) and there are no messages in transit.

To simulate such an algorithm in a mobile agent system, we need to execute the active steps (Step-1 and Step-2) at each node. This involves performing local computation at the nodes and delivering messages between nodes. The idea of the simulation is simple – the mobile agents can move from node to node delivering the messages; An agent that delivers a message

m to a node x, can also perform the local computation at node x that results from the receipt of the message m. Using the client-server paradigm, we can think of each node as a client which requires some service (transporting messages, performing local computation etc.) and the agents as mobile servers which move from node to node performing the required services.

For effectively simulating the message-passing computation using mobile agents that can fail at any time, we need to address the following issues:

[FR] Fairness: Every message that is generated at a node should be delivered within a finite time to its destination.

[TD] Termination Detection: Once all messages have been delivered and the execution of the original algorithm Z terminates, then each agent should be able to detect this and stop the simulation.

To ensure fairness, every agent would periodically visit each node to deliver the messages generated at that node. Thus, even if an agent dies, the others would keep servicing the nodes (delivering the messages). Note, however that an agent may die on the way while delivering some message m. In that case, another agent has to take over the task of delivering that particular message. So, we need to keep track of which messages have been delivered successfully. For this purpose, we maintain two message queues at each node– the *To-Be-Delivered*(TBD) message queue and the *Messages-Received*(MR) queue. Any messages generated at a node v, is added to the TBD queue at v, along with the port number $p = \lambda_v(e)$ of the edge e through which the message needs to be sent. A message can be removed from the TBD queue when it is known that it has been received successfully and written to the MR queue at the receiving end. A message in MR queue can be removed when it is read and the actions corresponding to the receipt of this message are executed. We keep such messages in a third queue called *Messages-Executed*(ME) queue. Each message m generated at a node v and to be sent through port p would be assigned an identifier, called MID(m); The TBD queue would store the original message m along with the (MID(m),p) pair as key, while the MR queue would store the message with (MID(m),q) as key, where q is the label of the port through which this message was received. The ME queue only needs to store the identifier (MID(m),q) and not the contents of the original message.

The simulation of the message-passing algorithm Z would be started at those nodes which are in the special state "*Initiator*". Recall that the original algorithm may not have an explicit termination detection mechanism. So, in order to ensure that the agents can detect when the computation has terminated, we would maintain a dynamic forest-like structure among the nodes of the graph, using an idea of Shavit and Francez (see [98]). The initiator nodes would be the root nodes and every other (active) node x would contain a link to its parent node i.e.

the node y which send the first message to node x to activate it. We would maintain at each node x a child-list containing the links to all its (active) children, i.e. all those nodes which were activated by messages from node x and are still active. The leaf nodes would be those that did not send any messages or sent messages to already active nodes. Once a leaf node completes its local computation and it does not have any messages to send, then it is removed from the child-list of its parent node and becomes inactive. Once a root node has no children and no further messages to send, it becomes inactive too. If all the root nodes become inactive then the simulation is terminated.
The simulation algorithm executed by each agent A is given below:

Algorithm *DisSimulate*

I Explore the network G and construct a traversal path P_A that starts and ends at the home-base of A, and visits every other node at least once.

II Walk through path P_A and at each node v which is in the state *Initiator* do the following -

1. Initiate the algorithm Z at node v and perform all local computation steps until the first time a message needs to be sent or received.
2. Update the state of node v by writing to the whiteboard the current state of execution. In particular, set *node-state* to "Processing" if a message needs to be sent, and to "Waiting" if a message needs to be received.
3. If a message m has to be sent through port p, then add (MID(m),p) to the TBD queue.

III Walk through path P_A and at each node x do the following -

1. If the TBD queue is not empty, then execute procedure *Deliver-Message.*
2. If node-state = "Processing", then continue with the local computation at current node until a message needs to be sent or received. Update the state of node x by writing to the whiteboard.
3. While the MR queue is not empty, execute procedure *Receive-Message.*
4. If there are no messages neither in the TBD queue nor in the MR queue, the Child-List is empty, and the node-state is "waiting", then write 'TERM' (for "terminated") on the whiteboard of x. Then go to the parent node y(if any) and delete x from the Child-List at node y. Now return back to node x.

IV If all the nodes visited in the previous step had a 'TERM' symbol written on their white-board then terminate the algorithm. Else repeat previous step.

where the procedures *Deliver-Message* and *Receive-Message* are as follows:

Procedure *Deliver-Message*

1. Let (m_i, p) be the first entry in the TBD queue. Leave the current node x through port p to reach node y (say, through port q), where $e = (x, y)$, $\lambda_x(e) = p$ and $\lambda_y(e) = q$.
2. If the message (m_i, q) is not present in the MR queue(or, the ME queue) at node y, then do the following:
   Add the message (m_i, q) to the MR queue,
   Delete any 'TERM' symbol (if present) from the whiteboard of node y,
   If the parent-link of node y is not set, set it to q.
   Set result to *Success*.
   Else set result to *Fail*.
3. Return back to node x and delete message (m_i, p) from the TBD queue (if present).
   If the parent-link of node y was set to q, then add the link $p = \lambda_x(e)$ to the Child-list of node x.
4. If the result is *Fail* and the TBD queue is not empty, then go back to the first step.
   Otherwise, return the result.

Procedure *Receive-Message*

1. Let (m_i, q) be the first entry in the MR queue. Remove the message (m_i, q) from the MR queue and perform the local computation at the current node that results from receiving this message from port q.
2. If a message m has to be sent through port p, then add $< MID(m), p >$ to the TBD queue. Update the state of the current node by writing to the whiteboard.
3. Add (m_i, q) to the ME queue

We shall now show the above algorithm satisfies the Fairness [FR] and Termination Detection [TD] properties. We denote by L_A the length of the traversal path P_A constructed by in step-(I) by an agent A, and we define L to be the maximum length of such a path; i.e.,

$$L = max_A\{L_A\}$$

Lemma 9.1.1 *An agent starting at any node v of G can construct a path of length $O(n)$ that starts and ends at v and visits each vertex of G at least once.*

If the nodes of the network are uniquely identifiable, each agent can construct the required path by a simple depth first traversal of the graph. Thus, in this case $L \leq 2n$.

Lemma 9.1.2 *Each message generated at some node x and added to the TBD queue at position r is delivered to its destination after at most $3L \cdot r$ moves by any single agent.*

Proof : To deliver a message successfully an (alive) agent A makes two moves (to and back). So, during a single traversal of the path P_A, if agent A finds some message in the queue at each node it visits, then it would make $3L$ moves. If node x appears only once in the path P_A, then after r such traversals, the message at position r would be delivered. So, as long as there is an alive agent A, it would succeed in delivering that message after $3L \cdot r$ moves. (Note that the other $k - 1$ agents may have died, without helping agent A in its task.) ∎

Due to assumption [A3], at least one agent remains alive. So, every message would be delivered within a finite amount of time. This ensures the fairness of message delivery.

Lemma 9.1.3 *Every message is delivered and executed exactly once and no message is repeated.*

Proof : Consider a message m that has to be sent from node u to node v on the edge $e(u, v)$, where $\lambda(e) = (p, q)$. When the message (m, q) is delivered at node v, it is written on MR queue. Due to the mutual exclusion property of the whiteboards, only one agent can write the message (m, q) to the MR queue. If any other agent reads the message (m, p) from the TBD queue of node u and attempts to deliver it, the agent would find the message (m, q) in either the MR queue or the ME queue of node v. When *Receive-Message*(m, q) is executed by an agent at a node v, the agent also deletes the message (m, q) from the MR queue of node v. An agent would obtain access to the whiteboard at the start of procedure *Receive-Message*, it would relinquish the whiteboard only after the message (m, q) has been deleted from the MR queue of node v. Hence every message is delivered and executed exactly once. ■

Lemma 9.1.4 *When any agent A terminates the simulation, then the following conditions hold: (i) No node has any more messages to send and (ii) there are no messages in transit.*

Proof : We define a node to be in *active* state if either there are some messages in its TBD queue or MR queue, or if its child-list is not empty. When a node u is marked with 'TERM', then none of the above holds and the node is said to be *inactive*. Any node v, other than the initiator nodes, becomes active only on receiving a message from another node u and in that case it becomes the child of node u. Thus, every active node is either an initiator (i.e. a root node in the spanning forest) or belongs to the tree rooted at some initiator. Note that once an initiator node becomes inactive, then it can never become a root node again. So, if an agent A on traversing the graph finds all nodes (including the initiators) are inactive, then no node can ever become active again. However, if any of the conditions of the lemma is false then there must be at least one active node. Hence the lemma holds. ■

Lemma 9.1.5 *Once the execution of the original algorithm Z terminates (i.e. when all messages have been delivered and executed), then every agent that is alive would terminate after at most $L \cdot n$ moves.*

Proof : Consider the instance when the last message (in the given execution) is received and executed at a node v. At this instance, all TBD and MR queues at every node would be empty and each node would be in state "waiting". During the next L moves of any agent A (at least one such agent exists due to assumption [A3]), all the nodes which do not have

any child-links(i.e. *leaf* nodes) would have 'TERM' written on their whiteboards. In every subsequent L moves, the nodes at the next higher level(i.e. the new *leaf* nodes) would have 'TERM' written on their whiteboards. So, after at most $L \cdot n$ moves by agent A, every node would have 'TERM' written on its whiteboard and thus the agent would terminate. ■

Lemma 9.1.6 *During any execution of the algorithm DisSimulate, the size of the TBD queue at any node v would be at most 1 and the size of the MR queue would be at most $(k \cdot L)$. The size of ME queue at any node v would be at most the degree of the node.*

Proof : An agent can add at most one message to the TBD queue at node v during each visit to node v. Whenever any agent A adds a message to the TBD queue, either the TBD queue was empty before this message was added or, agent A had succeeded in removing one message from the TBD queue in the previous step. This means that the contribution from each agent is one, when the queue is empty and it is zero in all other cases. So, the size of the TBD queue can be at most 1.

Each time an agent A visits node v it removes all pending messages from the MR queue of that node. Between two consecutive visits to a node v, the agent may make at most L visits to other nodes and if each such visit results in a message being sent to node v, then the total contribution of a single agent, to the MR queue of a node v, would be at most L. Hence the result follows. Whenever a message received from port p is added to the ME queue of node v, it can replace the previous message (if any) that was received from the same port. This is possible because whenever a message m_i is received on edge e, it implies that the previous message m_{i-1} send on the same edge, has already been deleted from the TBD queue at the source. The FIFO property of the channels (assumption [A1]) ensures this. ■

Notice that, if we can ensure that each node appears exactly once in the traversal path P_A of any agent A, then the maximum size of the MR queue would decrease to $k \cdot d$. In particular, if r is the maximum number of times a single node v appears in the traversal path of some agent A, then the size of the MR queue is never more than $(k \cdot d \cdot r)$.

Theorem 9.1.1 *The result obtained (i.e. the final state of the nodes) in any possible execution of algorithm* DisSimulate, *would be exactly identical to the result of some possible execution of the original algorithm Z in the S.A. model.*

Proof : By Lemmas 9.1.2-9.1.5 it follows that algorithm *DisSimulate* successfully and fairly delivers and executes every message corresponding to algorithm Z. For any execution of *DisSimulate* on the network (G, λ), consider the timing sequence in which messages are generated and received by the algorithm. Notice that such a sequence of operations satisfies the causal order of sending and receiving messages. Now, consider the execution of algorithm Z in an

equivalent S.A. system (G, λ). Since the system is asynchronous, messages may be sent and received in any possible order (satisfying the causal relationship). In particular, one of these possible ordering of the send and receive operations would correspond to the order in which the messages are generated and delivered during the execution of algorithm *DisSimulate*. So, this particular execution of algorithm Z would be equivalent to the execution of algorithm *DisSimulate*, i.e. the contents of each node would be exactly same during these two executions and thus, the same result would be obtained. ■

Notice that the above theorem holds, irrespective of the number of agents failing or crashing, if at least one agent is alive. So, if there exists a S.A. algorithm for solving any computational problem in a network (G, λ), then the same problem can be solved in the mobile agent system with just one agent. In other words, we can say the following:

Observation 9.1.1 *A mobile agent system with at least one agent in a network of n nodes, is computationally as powerful as a stationary agent system with n agents.*

Let us now measure the cost of the simulation.

Theorem 9.1.2 *During the simulation, the total number of moves made by k agents in a network of size n, is at most $O(k \cdot n)$ per message exchanged in an execution of the original algorithm Z.*

Proof : As a consequence of Lemma 9.1.2, r messages are delivered by an agent, using $3r \cdot L$ agent moves. Now, in the worst case, only one agent (say agent A) may be delivering all the messages while the other agents just keep following agent A, (always arriving at each node just after agent A). Thus, in this case, each agent would make $O(L)$ moves in delivering a message and the total agent moves would be $O(k \cdot L)$ per message. ■

Let us analyze the memory overhead required for our simulation algorithm. Notice that each agent needs to store, in its local memory, a copy of the traversal path P_A. Thus the amount of additional memory required by each agent for the simulation is $O(L \log \Delta)$, where Δ is the maximum degree of the graph.

The amount of additional memory required at the nodes (other than what is required for storing the state of the algorithm Z) must be enough for storing the message queues.

9.2 Computing in Anonymous Graphs

In the previous section we considered distributed systems having unique identifiers for each node of the network. In this section, we consider the simulation of distributed algorithms in anonymous networks, where the nodes do not have distinct labels. The same simulation

algorithm of the previous section can possibly be executed on anonymous networks too. Notice that even if the graph is anonymous, it is still possible for an agent to find a traversal path in the graph that visits every node, provided that the agent knows the size n of the graph or at least an upper bound on n.

Lemma 9.2.1 *An agent A starting at a node v of an anonymous graph G, can construct a path P(represented by a sequence of edge labels) of finite length that starts and ends at v and is guaranteed to visit each node at least once.*

Proof : Let V_i be the set of all nodes which can be reached from v by a path (not necessarily simple) of length i. (Note that $V_0 = \{v\}$.) The agent A simply traverses each outgoing edge from V_i to construct V_{i+1}. Once the agent has constructed V_{n-1}, it would have traversed every path of length less than n starting from its homebase and thus every node of the graph would have been visited (This follows from the fact that the graph is connected). This gives us the required traversal path. ■

Notice that the traversal path constructed in the proof above is of length $\Theta(\Delta^n)$. However in most cases, it is possible to construct much shorter traversal paths. In particular, if the agent can map the graph, then it can always construct a traversal path of length $O(n)$ (i.e. linear in the size of the graph).

In general however, it is not always possible to find a traversal path of linear (or even polynomial) length, visiting all the nodes of an arbitrary anonymous graph.

In fact, given any arbitrary anonymous graph G with k agents placed among the nodes of G, there is no known (deterministic) algorithm that will enable the agents to construct a path of length polynomial in n, visiting all the vertices of G. Thus, we have the following corollary of Theorem 9.1.2.

Corollary 9.2.1 *The algorithm* DisSimulate *when executed on anonymous networks, has an exponential overhead, in terms of agent moves, for delivering each message.*

When it is not possible to construct efficient traversal paths, we can use a different approach for simulating a message-passing computation on a graph G. We partition the graph G among the k agents and each agent would be responsible for the subgraph of G that it owns (we call this the territory of the agent). However, as some of the agents may die, the task of a dead agent must be taken over by some other (alive) agent. So, in our algorithm, each agent simulates the computation within its territory, while periodically checking if any of its neighboring agents are dead and annexing the territory of any dead agent. Initially, the agents mark their territories in the graph G using the procedure EXPLORE (see chapter 4), thus partitioning G into k disjoint trees.

The algorithm given below, simulates a given message-passing algorithm Z, on the graph G, using k mobile agents. Initially, each agent knows the steps of the algorithm Z and each initiator node is marked as such. The following fact will be used in the algorithm:

Fact 1 *Given any starting node v of G, every edge $e \in G$ can be uniquely identified by using the sequence of edge-labels for the path from v to e. We use the notation $\mathbf{id}_v(e)$ to denote such an identifier for an edge e at node v.*

ALGORITHM *AnSimulate*:

Phase 0: Each agent A executes procedure EXPLORE to obtain its territory T_A. The territory T_A is a tree rooted at the homebase, and all the other nodes contain a link marked as *home-link* which connects this node to its parent in the tree T_A. Let n_A be the size of T_A. Agent A then traverses its territory T_A and at each node v that it visits — if v in the state "initiator", then agent A initiates the algorithm Z at node v performing all local computation steps until the first time a message needs to be sent or received; Agent A then updates the state of node v and if any message m is to be sent through the port p, then the pair (m, p) is added to the *To-Be-Delivered*(TBD) queue at node v.

Phase $i \geq 1$: Agent A, (if alive) executes the following steps:

STEP 1: Agent A does a depth-first traversal of its territory T_A. During the traversal, for each node u that it visits, agent A does the following:

- If the TBD queue of node u is not empty, then execute procedure *Deliver-Message*;
- If node-state = "Processing", then continue with the local computation at current node until a message needs to be sent or received. Update the state of node x by writing to the whiteboard.
- While the MR queue is not empty, execute procedure *Receive-Message.*
- If there are no messages in both the TBD queue and MR queue, the Child-List is empty, and the node-state is "waiting", then write 'TERM' on the whiteboard of u. Then go to the parent node v(if any) and delete u from the Child-List at node v. Now return back to node u.
- Write DONE(i, n_A) on the whiteboard of u.

If during Step-1, agent A finds an ANNEXED$(j, n_B$,$\mathbf{id}(e'))$ mark in its homebase, then agent A goes to the edge e' and marks this edge as Tree-edge (For the next phase, $n_A \leftarrow n_A + n_B$ and

$T_A \leftarrow T_A + \{e'\} + T_B$.) In this case, agent A skips STEP-2 and jumps to STEP-3 to update its territory.

STEP 2: Agent A starts a depth-first traversal of its territory. During the traversal, for each external edge $e(u,v)$ incident to some node u in its territory, it traverses the edge e to reach the other end v, reads DONE(j, n_B) from whiteboard[1] at v and takes the following actions:

- If ($j < i$) or, ($j = i$ AND $n_B < n_A$), then go to the root-node x of the tree T_B containing v and write ANNEXED(i, n_A,$\mathbf{id}_x(e)$) (only if there is no other ANNEXED mark at node x).
- If successful in writing the ANNEXED mark, then return to e and mark this edge as a Tree-edge.(For the next phase $n_A \leftarrow n_A + n_B$ and $T_A \leftarrow T_A + \{e\} + T_B$.)

STEP 3: Agent A updates its territory T_A to include all territories that it annexed and those annexed by the agents that it defeated. If agent A itself was defeated, then it adds the territory of the agent C that defeated it and all territories annexed by C. The home-links of the nodes in the territory are updated and the value of n_A is modified accordingly. If all nodes in its territory had 'TERM' written on the whiteboards then agent A executes procedure *Termination-Detection* to check if the termination condition has been reached and if so, stops. Otherwise agent A goes to phase $i + 1$.

Procedure *Termination-Detection*

for $r \leftarrow 1$ to k, **do**
    if there is some node in T_A which does not have a 'TERM' mark on its whiteboard, **then**
      return false;
    else write 'TERM(r)' on the whiteboard of every node in T_A.
    for each non-tree edge $e(u,v)$ incident to a node $u \in T_A$, **do**
        Traverse edge e to reach node $v \in T_B$ (say),
        if node v has no 'TERM' mark, then return false;
        else if found a 'TERM(j)' mark and $j < r$, **then**
            go to the root of tree T_B and mark it with 'TERM(r)' (if not already marked so)
            if successful then merge T_A with T_B using the edge e and continue;
if $r = k$, then **return** true;

The procedures *Deliver-Message* and *Receive-Message* are same as before.

In the above algorithm, an agent A annexes the territory of another agent B, if either (i) B has died (or B is slower than A) or, (ii) if during some phase i, B has a smaller territory

[1]If no such mark is found, it reads DONE($j = 0, n_B = 0$).

than A. After agent A annexes the territory of agent B, these two territories are merged and both the agents(if alive) continue the simulation in the bigger territory. The algorithm ensures that the territory of an agent is always a tree. This ensures that each agent completes a single traversal (i.e. a single phase) in $O(n)$ moves.

We give below the proof of correctness of our algorithm and analyze its complexity. Due to assumption [A3], there are some agents which survive till the end of the algorithm—we shall call such agents "alive agents". The lemma below proves fairness of message delivery.

Lemma 9.2.2 *During algorithm AnSimulate, the following always holds:*

1. *The edges marked as tree-edges form a spanning forest of G, containing at most k trees (each rooted at some homebase).*
2. *Every node is visited by an alive agent at least once in every k phases.*
3. *An alive agent completes each phase within a finite amount of time.*
4. *Whenever an alive agent visits a node v, the top-most message in the TBD queue of v is delivered to its destination (unless the queue is empty).*
5. *Every message generated at a node v is delivered within a finite amount of time.*

Lemma 9.2.3 *When an agent A completes procedure* Termination-Detection *with a return value of true, then every message corresponding to the execution of algorithm Z, has been delivered and there are no messages in transit.*

Proof : If procedure Termination-Detection returns true, then every node in agent A's territory has TERM(k) written on it. This implies that nodes in neighboring territories are at least marked with TERM($k-1$), while territories at a distance of two are marked with $(k-2)$ and so on. Since the network is partitioned into at most k territories, every node in the network would have a TERM mark, which implies that no node has any more messages to send and there are no messages in transit. ∎

Lemma 9.2.4 *When every message corresponding to the execution of algorithm Z, has been delivered, every alive agent terminates within at most k phases.*

Proof : Once all messages have been delivered, every node would become inactive. When an alive agent A visits such a node, it would mark it with 'TERM'. So all the nodes in agent A's territory would be marked with 'TERM' and the agent would start procedure Termination-Detection. As there are no nodes in G, which does not have a TERM mark, the procedure would return true after at most k phases and agent A would terminate. ∎

Theorem 9.2.1 *Algorithm* AnSimulate *correctly simulates any given massage-passing algorithm Z, even if up to $f \leq k-1$ agents out of the k agents, crash.*

Proof : The fairness of message delivery is ensured by Lemma 9.2.2, while the lemmas 9.2.3 and 9.2.4 ensure that algorithm *AnSimulate* terminates correctly. As before, the procedures *Deliver-Message* and *Receive-Message* ensure that every message is delivered and executed exactly once and further, the delivery of messages follows the causal ordering of receiving and sending messages. Thus, any execution of algorithm *AnSimulate* would correspond to some possible execution of algorithm Z in the S.A. model. ■

Let us now analyze the cost of the simulation.

Lemma 9.2.5 *During every phase in which no agents crash, at least one message is delivered unless there are no more pending messages.*

Proof : Notice that every vertex v is visited once in every phase i, unless some agent crashed in phase i. Whenever node v is visited, the first message (if any) in the TBD queue of node v would be delivered. Thus, except for those phases (at most $f < k$) during which some agents crash, every phase results in the delivery of at least one message. ■

Lemma 9.2.6 *In each phase of algorithm AnSimulate, the number of moves made by the agents in total is* $O(m \cdot k)$.

The above result follows from the fact that each edge is traversed a constant number (at most six) times by any single agent. However, we can make the following modification to the algorithm to reduce its communication complexity. Whenever an agent traverses an external edge during step-2 of a phase, it can mark the edge such that no other agent follows it and traverses the same edge again in this phase. Thus any non-tree edge would be traversed by at most one agent from each side. However, a tree edge may be traversed by all agents that share that territory. So, the number of moves per phase would be $O(m + n \cdot k')$ for k' alive agents. Thus, we have the following result, due to Lemma 9.2.5.

Theorem 9.2.2 *For the (modified) algorithm AnSimulate, the overhead for delivering each message is* $O(m + n \cdot k')$, *where* k' *is the number of surviving agents and* n *and* m *are the number of nodes and edges respectively, in the graph.*

We would like to remark that the large overhead for message delivery is due to the fact that we have to deal with failures of any number of agents at any time during the simulation. In environments without agent failures, it is always possible to simulate a message-passing computation much more efficiently. For example, the algorithm given by Chalopin et al.[32] for the fault-free environment requires $O(n)$ agent moves per message delivery in the worst case. On the other hand, if we do not require fairness of message delivery, then we can use a much simpler algorithm (based on the idea in [102]) requiring only constant number of moves per message delivery.

9.3 Conclusions

We proposed and studied methods for simulating any message-passing computation in a mobile agent system with faulty agents. In particular, we have shown how to simulate a given message-passing algorithm, in a mobile agent system, while tolerating the crash-fault of any number of agents, provided that at least one agent is alive. Thus, agent crashes do not restrict the computational power of a mobile agent system. Another interesting observation is that a mobile agent system of n nodes with at least one agent is computationally as powerful as a stationary agent system with n agents.

We presented an algorithm *DisSimulate*, for simulating a message-passing computation in a labelled network (or, a network which can be explored efficiently). In this algorithm, the agents work independently of one another, with no communication among the agents and this makes this algorithm very robust. However, one problem is sometimes the workload is not evenly distributed among the agents, i.e. in the worst case, all the messages may be getting delivered by a single agent, even though the other agents may still be alive.

For anonymous networks or networks where node identities are not visible to the agents, we gave another algorithm *AnSimulate*, where the network is partitioned into disjoint parts which are serviced by different groups of agents. Even though the agents may crash at any time, the algorithm ensures that the simulation proceeds flawlessly irrespective of the agent crashes and the system always stabilizes to a state where the workload is almost equally distributed among the remaining agents.

One of the limitations of our results is the assumption that agents can only fail while traversing an edge. We have not considered the possibility of an agent failing while performing computation at a node, because in such a case, the whiteboard of the node may remain locked and thus inaccessible to all other agents, which can create a deadlock. Future studies on this problem should be directed towards finding a way to deal with crashes of agents inside a node, while avoiding the deadlock situation.

Chapter 10

Conclusions and Future Work

10.1 Summary of Results

In this thesis we studied distributed systems where the computational entities are mobile as opposed to the conventional model where computational entities are stationary and communicate by sending and receiving messages. In our model, the environment is modelled as a graph where the entities (agents) can autonomously move from node to node by traversing the edges. The agents may also read and write information at public whiteboards available at each node. This model which corresponds to that of software agents traversing networked environments such as the Internet.

We studied the rendezvous problem in such systems, which requires all the agents to gather together at a single location. The previous deterministic solutions to this problem were mainly restricted to agents with distinct labels in synchronous networks. We studied the problem in much more general (and also computationally difficult) scenario where both the agents and the network nodes are anonymous and the network is asynchronous. The algorithms presented in this thesis are generic protocols and are applicable on any arbitrary network topology. Moreover, these algorithms are designed for the weakest scenario and thus, these algorithms can be easily implemented in other (stronger) models of computation. For any mobile agent algorithm, the detection of termination is a major issue that is often overlooked. In this thesis, we designed algorithms which terminate explicitly, even when a solution to the problem can not be reached.

We showed that solving the *rendezvous* problem in this model is equivalent to solving any of the following problems: Leader election, Graph-labelling or Labelled map construction. Each of these problems require breaking the symmetry between identical and thus indistinguishable agents. The symmetry breaking problem has been previously studied for the Stationary Agent model and the concept of *symmetricity* of a graph has been defined in this context. Using

these results, we show that the rendezvous problem cannot be solved deterministically unless the symmetricity of the bi-colored graph is equal to 1, where the bi-coloring denotes the initial locations of the agents in the graph.

We first studied the rendezvous problem in the token model, where each agent has a single token that can be used to mark a node in the graph. A simple algorithm called *Compare-View* for solving rendezvous in this restricted model was presented. This algorithm is expensive in terms of the number of moves made by the agents. However, if a map of the graph is known to the agents, it is possible to solve rendezvous using $O(n \cdot k)$ moves. This is possible even if some of the tokens fail during the execution of the algorithm. An interesting question for future research is whether these results can be extended to the case when the map of the graph is unknown to the agents. Also it would be interesting to find a lower bound for solving rendezvous using tokens in the absence of any knowledge of network topology. Currently no sub-exponential algorithm is known for this case.

We next considered the whiteboard model for communication between agents. It is possible to simulate the token model using one bit whiteboards and thus any algorithm for the token model is applicable in the whiteboard model too. However, it is possible to design more efficient algorithms for rendezvous when whiteboards are available. In fact our results from Chapter 6 suggests that there is a trade-off between the two cost measures of agent moves and whiteboard memory, when solving the agent election problem. Using larger whiteboards at the nodes allows the agents to communicate more information with each-other, thus reducing the number of moves they need to make. An open question is whether it is possible to optimize both the number of agent moves and the amount of whiteboard memory. In Chapter 6 we gave solutions to rendezvous first for the case when the values of n and k are co-prime. Our solution for this case requires $O(\log n)$ whiteboard memory and makes $O(m \log k)$ moves in total. We also presented algorithms for the general case (i.e. effective algorithms) which solve the problem whenever it is deterministically feasible. The cost of our algorithm is $O(m \cdot k)$ moves, using whiteboards of size $O(m \log n)$. When the whiteboards are of smaller size ($\Omega(\log n)$), our solution has a cost of $O(k \cdot n \cdot m^2)$ agent moves in total. On the other hand we showed that a trivial lower bound for the same problem is $\Omega(n \cdot k)$ agent moves. Thus, it would be interesting to see if the gap between the lower bound and upper bounds can be closed or at least narrowed, in future.

In those settings where the rendezvous problem is not deterministically solvable, we may either attempt the rendezvous of as many agents as possible in the same location (*maximal rendezvous*), or try to gather the agents as close to each-other as possible (*Near-Gathering*). We also introduced the *Grouping* problem which requires the agents to gather in multiple groups of the same size. We studied these problems in Chapter 7 and presented conditions for their solution in graphs of arbitrary topology.

Most previous work on the mobile agent model has been done under the assumption that the environment in which the agents travel is safe. There has been a few attempts at designing fault tolerant solutions for the mobile agent model, but these were restricted to specific topologies (e.g. the ring network). In this thesis, we considered networks where some of the nodes and edges are faulty or un-safe for the agents. An agent that attempts to traverse an unsafe edge, dies (or disappears). A node is unsafe if all edges incident to it are unsafe (such a node is called a *black-hole*). We showed that if there are τ faulty links, i.e. links connecting a safe part of the network to an unsafe part, then we cannot avoid the demise of τ agents. We defined the *Rendezvous-number* of a faulty network as maximum number of agents that may rendezvous in this network. We gave solutions for the rendezvous of w agents, for any w less than the Rendezvous-number of the graph. The results presented in Chapter 8 are generalizations of the previous known results for the case of ring networks.

In Chapter 8, the faults in the network were assumed to be permanent and localized. So, if one particular network edge was faulty, it remains so for the complete duration of the algorithm and every agent traversing this edge dies. However, in many cases faults in the network may be transient, i.e. they occur momentarily in different locations of the network. Each occurrence of such a fault may result in the loss of an agent—we call it an *agent crash*. In Chapter 9 we showed how to perform any distributed computation in a mobile agent system, tolerating such agent crashes. We presented an algorithm for simulating any distributed algorithm (for the stationary agent model) in a mobile agent system while tolerating any number of agent crashes, as long as at least one agent remained alive. Our analysis of the simulation shows that there is a significant cost associated with the translation of a message-passing algorithm to the mobile agent model. This suggests that in designing algorithms for the mobile agent model we should follow an approach distinct from that of the conventional stationary agent model.

10.2 Open Problems

We conclude this thesis by presenting some open problems that arise from our study of the rendezvous problem for anonymous mobile agents. We hope that solutions to some of these problems would be obtained in the near future.

- What is the minimum number of moves required for solving the *Rendezvous with Detect* problem for identical agents in an arbitrary anonymous graph? The current best lower bound is $\Omega(n \cdot k)$, but we believe this can be improved.

- When using tokens or perhaps whiteboards of constant size, is there any effective algorithm for rendezvous whose cost is sub-exponential in terms of the number of moves?

- We presented an algorithm for solving rendezvous in the arbitrary graphs using tokens that fail, when a map of the network is available. Can the same be achieved without a map? What would be the cost in that case?
- What is the minimum size of whiteboards necessary for an efficient (i.e. polynomial cost) and effective solution to rendezvous in arbitrary graphs without the knowledge of a map?

Bibliography

[1] Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. In *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms (SODA '04)*, pages 1063–1071, 2004.

[2] Susanne Albers and Monika Rauch Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29:1164–1188, 2000.

[3] Steve Alpern. Hide and seek games. Seminar, Institut fur Hohere Studien, Wien, July 1976.

[4] Steve Alpern. Rendezvous search: A personal perspective. *Operations Research*, 506(5):772–795, 2002.

[5] Steve Alpern, Vic Baston, and Skander Essegaier. Rendezvous search on a graph. *Journal of Applied Probability*, 36(1):223–231, 1999.

[6] Steve Alpern and Shmuel Gal. *The Theory of Search Games and Rendezvous.* Kluwer, 2003.

[7] Edward J. Anderson and Skander Essegaier. Rendezvous search on the line with indistinguishable players. *SIAM Journal of Control and Optimization*, 33(6):1637–1642, 1995.

[8] Edward J. Anderson and Sándor P. Fekete. Asymmetric rendezvous on the plane. In *Proc. of the fourteenth Annual Symposium on Computational Geometry (SCG '98)*, pages 365–373. ACM Press, 1998.

[9] Hideki Ando, Yoshinobu Oasa, Ichiro Suzuki, and Masafumi Yamashita. A distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transaction on Robotics and Automation*, 15(5):818–828, 1999.

[10] Dana Angluin. Local and global properties in networks of processors. In *Proc. of 12th Symposium on Theory of Computing (STOC'80)*, pages 82–93, 1980.

[11] Midori Asaka, Shunji Okazawa, Atsushi Taguchi, and Shigeki Goto. A method of tracing intruders by use of mobile agents. In *Proc. 9th annual conference of the Internet Society (INET'99)*, 1999.

[12] Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In *Proc. of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 230–240, 1987.

[13] Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152(2):155–172, 1999.

[14] Tucker R. Balch and Ronald C. Arkin. Behavior based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[15] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proc. 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA'02)*, pages 324–332, 2002.

[16] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Can we elect if we cannot compare? In *Proc. 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA'03)*, pages 200–209, 2003.

[17] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems*, 40(2):143–162, 2007. Preliminary version in Proc. 10th Coll. on Structural Information and Communication complexity (SIROCCO'03), pages 17–32, 2003.

[18] Lali Barrière, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M. Thilikos. Searching is not jumping. In *Graph-Theoretic Concepts in Computer Science, 29th International Workshop (WG'03)*, pages 34–45, 2003.

[19] Vic Baston and Shmuel Gal. Rendezvous search when marks are left at the starting points. *Naval Research Logistics*, 48(8):722–731, 2001.

[20] Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil P. Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *Proc. 30th ACM Symposium on Theory of Computing (STOC'98)*, pages 269–287, 1998.

[21] Michael A. Bender and Donna K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proc. 35th Symposium on Foundations of Computer Science (FOCS'94)*, pages 75–85, 1994.

[22] Daniel Bienstock and Paul D. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991.

[23] Lélia Blin, Pierre Fraigniaud, Nicolas Nisse, and Sandrine Vial. Distributed chasing of network intruders. In *Proc. of 13th Coll. on Structural Information and Communication Complexity (SIROCCO'06)*, pages 70–84, 2006.

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

[25] Paolo Boldi, Shella Shammah, Sebastiano Vigna, Bruno Codenotti, Peter Gemmell, and Janos Simon. Symmetry breaking in anonymous networks: Characterizations. In *Proc. 4th Israeli Symposium on Theory of Computing and Systems*, pages 16–26, 1996.

[26] Paolo Boldi and Sebastiano Vigna. Computing anonymously with arbitrary knowledge. In *Proceedings of the 18th ACM Symposium on principles of distributed computing*, pages 181–188. ACM Press, 1999.

[27] Paolo Boldi and Sebastiano Vigna. An effective characterization of computability in anonymous networks. In *Proc. of 15th Int. Conference on Distributed Computing (DISC'01)*, volume 2180 of *LNCS*, pages 33–47, 2001.

[28] Paolo Boldi and Sebastiano Vigna. Fibrations of graphs. *Discrete Math.*, 243:21–66, 2002.

[29] Y. Uny Cao, Alex S. Fukunaga, and Andrew Kahng. Cooperative mobile robotics: antecedents and directions. *Autonomous Robots*, 4:7–27, 1997.

[30] Jérémie Chalopin. Election in the qualitative world. In *Proc. 13th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2006)*, pages 85–99, 2006.

[31] Jérémie Chalopin, Shantanu Das, and Nicola Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. In *20th International Symposium on Distributed Computing (DISC'06)*, volume 4731 of *LNCS*, pages 108–122. Springer, 2007.

[32] Jérémie Chalopin, Emmanuel Godard, Yves Métivier, and Rodrigue Ossamy. Mobile agents algorithms versus message passing algorithms. In *Proc. of the 10th Int. Conference on Principles of Distributed Systems (OPODIS'06)*, volume 4305 of *LNCS*, pages 187–201, 2006.

[33] Jérémie Chalopin and Yves Métivier. A bridge between the asynchronous message passing model and local computations in graphs. In *Proc. of Mathematical Foundations of Computer Science (MFCS'05)*, volume 3618 of *LNCS*, pages 212–223, 2005.

[34] David M. Chess. Security issues in mobile code systems. In *Proc. Conference on Mobile Agent Security*, volume 1419 of *LNCS*, pages 1–14, 1998.

[35] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Solving the robots gathering problem. In *Automata, Languages and Programming, 30th International Colloquium, (ICALP'03)*, pages 1181–1196, 2003.

[36] Colin Cooper, Ralf Klasing, and Tomasz Radzik. Searching for black-hole faults in a network using multiple agents. In *Proc. 10th Int. Conf. On Principles of Distributed Systems (OPODIS'06)*, volume 4305 of *LNCS*, pages 320–332, 2006.

[37] Jurek Czyzowicz, Leszek Gasieniec, and Andrzej Pelc. Gathering few fat mobile robots in the plane. In *Principles of Distributed Systems, 10th International Conference, (OPODIS 2006)*, pages 350–364, 2006.

[38] Jurek Czyzowicz, Dariusz R. Kowalski, Euripides Markou, and Andrzej Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71(2-3):229–242, 2006.

[39] Jurek Czyzowicz, Dariusz R. Kowalski, Euripides Markou, and Andrzej Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability and Computing*, 16(4):595–619, 2007. Preliminary version in Proc. 8th Int. Conf. on Principles of Distributed Systems (OPODIS 2004).

[40] Shantanu Das. Mobile agent rendezvous in a ring using faulty tokens. In *9th International Conference on Distributed Computing and Networking (ICDCN)*, volume 4904 of *LNCS*, pages 292–297, 2008.

[41] Shantanu Das, Paola Flocchini, Shay Kutten, Amiya Nayak, and Nicola Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science*, 385(1-3):34–48, 2007.

[42] Shantanu Das, Paola Flocchini, Amiya Nayak, and Nicola Santoro. Distributed exploration of an unknown graph. In *Proc. 12th Coll. on Structural Information and Communication complexity (SIROCCO'05)*, volume 3499 of *LNCS*, pages 99–114, 2005.

[43] Shantanu Das, Paola Flocchini, Amiya Nayak, and Nicola Santoro. Effective elections for anonymous mobile agents. In *Proc. of 17th Int. Symposium on Algorithms and Computation (ISAAC'06)*, volume 4288 of *LNCS*, pages 732–743, 2006.

[44] Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Fault-tolerant simulation of message-passing algorithms by mobile agents. In *Proc. of 14th Coll. on Structural Information and Communication Complexity (SIROCCO'07)*, volume 4474 of *LNCS*, pages 289–303, 2007.

[45] Shantanu Das, Shay Kutten, and Ayelet Yifrach. Improved distributed exploration of anonymous networks. In *8th International Conference on Distributed Computing and Networking (ICDCN)*, volume 4308 of *LNCS*, pages 306–318, 2006.

[46] Xiaotie Deng, Tiko Kameda, and Christos H. Papadimitriou. How to learn an unknown environment - i: The rectilinear case. *Journal of the ACM*, 45:215–245, 1998.

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

[48] Anders Dessmark, Pierre Fraigniaud, and Andrzej Pelc. Deterministic rendezvous in graphs. In *Proc. 11th European Symposium on Algorithms (ESA'03)*, pages 184–195, 2003.

[49] Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51:38–63, 2004.

[50] Stefan Dobrev, Paola Flocchini, Rastislav Královič, and Nicola Santoro. Exploring a dangerous unknown graph using tokens. In *Proc. 5th IFIP International Conference on Theoretical Computer Science (TCS'06)*, 2006.

[51] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Finding a black hole in an arbitrary network: optimal mobile agents protocols. In *Proc. of 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 153–162, 2002.

[52] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Multiple agents rendezvous in a ring in spite of a black hole. In *Proc. of 7th International Conference on Principles of Distributed Systems (OPODIS'03)*, volume 3144 of *Lecture Notes in Computer Science*, pages 34–46. Springer, 2003.

[53] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48(1):67–90, 2007. Preliminary version in Proc. 15th Int. Symp. on Distributed Computing (DISC 2001).

[54] Stefan Dobrev, Rastislav Královič, Nicola Santoro, and Wei Shi. Black hole search in asynchronous rings using tokens. In *Proc. 6th Italian Conference on Algorithms and Complexity, (CIAC'06), Rome*, volume 3998 of *LNCS*, pages 139–150. Springer, 2006.

[55] Peter Dömel, Anselm Lingnau, and Oswald Drobnik. Mobile agent interaction in heterogeneous environments. In *Proc. First International Workshop on Mobile Agents (MA '97)*, pages 136–148, 1997.

[56] Gregory Dudek, Michael R. M. Jenkin, Evangelos E. Milios, and David Wilkes. Robotic exploration as graph construction. *Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[57] Shay Kutten Ephraim Korach and Shlomo Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Trans. Program. Lang. Syst.*, 12(1):84–101, 1990.

[58] Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Flaminia L. Luccio, Nicola Santoro, and Cindy Sawchuk. Mobile agents rendezvous when tokens fail. In *Proc. 11th International Colloquium on Structural Information and Communication Complexity, (SIROCCO 2004)*, pages 161–172, 2004.

[59] Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Nicola Santoro, and C. Sawchuk. Multiple mobile agent rendezvous in a ring. In *Proc. 6th Latin American Theoretical Informatics Symposium (LATIN'04)*, pages 599–608, 2004.

[60] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005.

[61] Paola Flocchini, Alessandro Roncato, and Nicola Santoro. Computing on anonymous networks with sense of direction. *Theor. Comput. Sci.*, 1-3(301):355–379, 2003.

[62] Pierre Fraigniaud, Leszek Gasieniec, D. Kowalski, and Andrzej Pelc. Collective tree exploration. In *6th Latin American Theoretical Informatics Symp. (LATIN'04)*, pages 141–151, 2004.

[63] Pierre Fraigniaud and David Ilcinkas. Digraph exploration with little memory. In *21st Symp. on Theoretical Aspects of Computer Science (STACS'04)*, pages 246–257, 2004.

[64] Pierre Fraigniaud, David Ilcinkas, G. Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. In *29th Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 451–462, 2004.

[65] Pierre Fraigniaud, David Ilcinkas, Sergio Rajsbaum, and Sébastien Tixeuil. Space lower bounds for graph exploration via reduced automata. In *Proc. of the 12th Int. Colloquium*

on Structural Information and Communication Complexity, SIROCCO'05, pages 140–154, 2005.

[66] Munehiro Fukuda, Lubomir F. Bic, Michael B. Dillencourt, and Jason M. Cahill. Messages versus messengers in distributed programming. *Journal of Parallel and Distributed Computing*, 57(2):188–211, 1999.

[67] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[68] Damianos Gavalas, Dominic Greenwood, Mohammed Ghanbari, and Mike O'Mahony. An infrastructure for distributed and dynamic network management based on mobile agent technology. In *Proc. of IEEE International Conference on Communications (ICC '99)*, volume 2, pages 1362–1366, 1999.

[69] Leszek Gasieniec, Evangelos Kranakis, Danny Krizanc, and X. Zhang. Optimal memory rendezvous of anonymous mobile agents in a unidirectional ring. In *Proc. of 32nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'06)*, pages 282–292, 2006.

[70] Emmanuel Godard, Yves Métivier, and Anca Muscholl. Characterization of Classes of Graphs Recognizable by Local Computations. *Theory of Computing Systems*, 37(2):249–293, 2004.

[71] Fritz Hohl. A model of attacks of malicious hosts against mobile agents. In *Proc. 4th ECOOP Workshop on Mobility: Secure Internet Mobile Computations*, LNCS 1603, pages 105 - 120, 1998.

[72] John V. Howard. Rendezvous search on the interval and the circle. *Operations Research*, 47(4):550–588, 1999.

[73] Ralph E. Johnson and Fred B. Schneider. Symmetry and similarity in distributed systems. In *Proc. fourth Annual ACM Symposium on Principles of Distributed Computing (PODC'85)*, pages 13–22, 1985.

[74] David Jung, Gordon Cheng, and Alexander Zelinsky. Experiments in realising cooperation between autonomous mobile robots. In *Proc. Fifth International Symposium on Experimental Robotics*, pages 609–620, 1997.

[75] Ralf Klasing, Euripides Markou, and Andrzej Pelc. Gathering asynchronous oblivious mobile robots in a ring. In *Proc. of 17th Int. Symposium on Algorithms and Computation (ISAAC'06)*, pages 744–753, 2006.

[76] Ralf Klasing, Euripides Markou, Tomasz Radzik, and Fabiano Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theoretical Computer Science*, 384(2-3):201–221, 2007.

[77] D. R. Kowalski and Andrzej Pelc. Polynomial deterministic rendezvous in arbitrary graphs. In *Proc. of 15th Int. Symposium on Algorithms and Computation (ISAAC'04)*, pages 644–656, 2004.

[78] Dexter Kozen. Automata and planar graphs. In *Fundations Computatial Theory (FCT'79)*, pages 243–254, 1979.

[79] Evangelos Kranakis, Danny Krizanc, and Euripides Markou. Mobile agent rendezvous in a synchronous torus. In *Proc. of 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, pages 653–664, 2006.

[80] Evangelos Kranakis, Danny Krizanc, Nicola Santoro, and C. Sawchuk. Mobile agent rendezvous in a ring. In *Int. Conf. on Distibuted Computing Systems (ICDCS 03)*, pages 592–599, 2003.

[81] Danny B. Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communcations of the ACM*, 42(3):88–89, 1999.

[82] Andrea S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, 1993.

[83] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, 2006.

[84] Antoni Mazurkiewicz. Distributed enumeration. *Inf. Processing Letters*, 61(5):233–239, 1997.

[85] Nimrod Megiddo, S. Louis Hakimi, M. R. Garey, David S. Johnson, and Christos H. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35(1):18–44, 1988.

[86] Horst Müller. Automata catching labyrinths with at most three components. *Elektronische Informationsverarbeitung und Kybernetik*, 15(1/2):3–9, 1979.

[87] Nancy Norris. Universal covers of graphs: isomorphism to depth $n-1$ implies isomorphism to all depths. *Discrete Applied Mathematics*, 56:61–74, 1995.

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

[89] Petrisor Panaite and Andrzej Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33:281–295, 1999.

[90] Torrence D. Parsons. Pursuit-evasion in a graph. In *Theory and Application of Graphs*, volume 642 of *Lecture Notes in Mathematics*, pages 426–441, 1976.

[91] David Peleg Pierre Fraigniaud, Andrzej Pelc and Stephane Perennes. Assigning labels in unknown anonymous networks with a leader. *Distributed Computing*, 14(3):163–183, 2001.

[92] Giuseppe Prencipe. *Distributed Coordination of a Set of Autonomous Mobile Robots.* PhD thesis, University of Pisa, 2002.

[93] Giuseppe Prencipe. On the feasibility of gathering by autonomous mobile robots. In *Proc. 12th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2005)*, volume LNCS 3499, pages 246–261, 2005.

[94] Michael O. Rabin. Maze threading automata. Technical Report Seminar Talk, University of California at Berkeley, October 1967.

[95] Naoshi Sakamoto. Comparison of initial conditions for distributed algorithms on anonymous networks. In *Proc. of the 18th annual ACM Symp. on Principles of Distributed Computing, PODC '99*, pages 173–179, 1999.

[96] Tomas Sander and Christian F. Tschudin. Protecting mobile agents against malicious hosts. In *Proc. of Conference on Mobile Agent Security*, LNCS 1419, pages 44–60, 1998.

[97] Claude E. Shannon. Presentation of a maze-solving machine. In *8th Conf. of the Josiah Macy Jr. Found. (Cybernetics)*, pages 173–180, 1951.

[98] Nir Shavit and Nissim Francez. A new approach to detection of locally indicative stability. In *Proc. of 13th International Colloquium on Automata, Languages and Programming, (ICALP'86)*, volume 226 of *LNCS*, pages 344–358. Springer, 1986.

[99] Wei Shi. *Using mobile agents for black hole search with tokens in multi networks.* PhD thesis, Carleton University, Ottawa, Canada, 2007.

[100] Kazuo Sugihara and Ichiro Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotics Systems*, 13:127–139, 1996.

[101] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *Siam J. Computing*, 28(4):1347–1363, 1999.

[102] Prasoon Tiwari and Michael C. Loui. Simulation of chaotic algorithms by token algorithms. In *1st International Workshop on Distributed Algorithms on Graphs (WDAG'85)*, pages 49–67. Carleton University Press, 1986.

[103] Masafumi Yamashita and Tiko Kameda. Computing functions on asynchronous anonymous networks. *Math. Systems Theory*, 29:331–356, 1996.

[104] Masafumi Yamashita and Tiko Kameda. Computing on anonymous networks: Part I–Characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.

[105] Masafumi Yamashita and Tiko Kameda. Computing on anonymous networks: Part II–Decision and membership problems. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):90–96, 1996.

[106] Xiangdong Yu and Moti Yung. Agent rendezvous: A dynamic symmetry-breaking problem. In *Proc. International Colloquium on Automata Languages and Programming (ICALP'96)*, pages 610–621, 1996.