OUKA
Osaka University Knowledge Archive

Title	A Study on Resource Control and Gossiping for Mobile Agent Systems
Author(s)	鈴木, 朋子
Citation	
Issue Date	
Text Version	ETD
URL	http://hdl.handle.net/11094/833
DOI	
rights	
Note	

Osaka University Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

Osaka University

A Study on Resource Control and Gossiping for Mobile Agent Systems

Submitted to

Graduate School of Information Science and Technology

Osaka University

January 2008

Tomoko SUZUKI

List of Related Publications

Journal Papers

1. Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, and Toshimitsu Masuzawa, "Self-Adaptive Mobile Agent Population Control in Dynamic Networks based on the Single Species Population Model", *IEICE Transactions on Information and Systems*, Vol.E90-D, No.1, pp 314-324, Jan. 2007.

2. Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "Move-optimal Gossiping among Mobile Agents", *Theoretical Computer Science*(to appear).

Conference Papers

3. Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, and Toshimitsu Masuzawa, "Self-Adaptation of Mobile Agent Population in Dynamic Networks: A Biologically Inspired Approach", *Proceedings of 2nd IEEE International Conference on Autonomic Computing (ICAC)*, pp 374-375, Jun. 2005. (Poster)

4. Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, and Toshimitsu Masuzawa, "Biologically Inspired Self-Adaptation of Mobile Agent Population", *Proceedings of 3rd International Workshop on Self-Adaptable and Autonomic Computing Systems (SAACS)*, pp 170-174, Aug. 2005.

5. Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "Bio-inspired Replica Density Control in Dynamic Networks", *Proceedings of 2nd International Workshop on Biologically Inspired Approaches to Advanced Information Technology (Bio-ADIT)*, pp 281-293, Jan. 2006.

6. Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "Optimal Moves for Gossiping among Mobile Agents", *Proceedings of 14th Colloquium on Structual Information and Communication Complexity (SIROCCO)*, pp 151-165, Jun. 2007.

Technical Reports

7. Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, and Toshimitsu Masuzawa, "A Biologically Inspired Approach to Mobile Agent Population Control in Dynamic Networks", *Technical Report of IPSJ*, 2004-MPS-52, Vol.2004, No.130, pp 65-68, Dec.2004 (in Japanese).

8. Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "A Biologically Inspired Approach to Replica Control in Dynamic Networks", *Technical Report of IPSJ*, 2005-AL-102, Vol.2005, No.91, pp 51-58, Sep. 2005.

9. Tomoko Suzuki, Taisuke Izumi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "Move Complexity of Gossiping among Mobile Agents", *Technical Report of IEICE*, COMP2006-50-59, Vol.106, No.566, pp 29-36, Mar. 2007.

List of Unrelated Publications

Journal Papers

10. Koji Inui, Tomoko Suzuki, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "A Multiple Path Based Approach for Achieving Fault Tolerance on Structured Overlay Networks", *IEICE Transactions on Information and Systems* (to appear, in Japanese).

Technical Reports

11. Koji Inui, Tomoko Suzuki, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "A Multiple Path based approach to Fault Tolerance in Chord", *Technical Report of IEICE*, NS2006-138-149, Vol.106, No.418, pp 39-44, Dec. 2006 (in Japanese).

12. Masakazu Morikawa, Tomoko Suzuki, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "A Connected Sensor Cover Algorithm in Sensor Networks", *Technical Report of IPSJ*, 2007-DPS-130, Vol.2007, No.16, pp 357-362, Mar. 2007 (in Japanese).

13. Masahiro Furukawa, Tomoko Suzuki, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "A tool for understanding f-resilient distributed algorithms based on similar executions", *Technical Report of IPSJ*, 2007-MPS-63, Vol.2007, No.19, pp 37-40, Mar. 2007 (in Japanese).

14. Daisuke Kadono, Tomoko Suzuki, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "ACO Routing with GPS for Mobile Ad Hoc Networks", *Technical Report of IPSJ*, 2007-MPS-67, Vol.2007, No.128, pp 227-230, Dec. 2007 (in Japanese).

15. Gen Nishikawa, Tomoko Suzuki, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "A wireless LAN network routing protocol based on attractor-selection", *Technical Report of IPSJ*, 2007-MPS-67, Vol.2007, No.128, pp 231-234, Dec. 2007.

Abstract

A distributed computing system consists of a collection of individual autonomous computers that are connected through a network. As a system model of distributed computing systems, *message passing model*, in which the computers communicate by sending and receiving bounded sequences of bits, is traditionally applied in many systems. However, the design of distributed systems based on the message passing model is growing increasingly complex as the network becomes larger, more dynamic and diverse. *Mobile agent model* provides one of the most promising frameworks to implement distributed systems. Mobile agents are autonomous programs that can migrate from one node to another in a network, and traverse a distributed system to carry out a task at each node. Since adaptability and flexibility of mobile agents simplify the design of distributed systems, mobile agents have been extensively studied for several years by many researchers. As a computational universe, the mobile agent model opens a variety of new challenging problems.

In this dissertation, we focus on *resource control* in mobile agent systems and *gossiping* among mobile agents. In the resource control, we try to control the number of resources, such as mobile agents, files and data, in the system. The resource control is a precious scheme to construct an efficient system since the excessive resources in the network increase the system load (e.g., consumption of processing power, network bandwidth and memory space) and the shortage of resources causes the system performance decrement. We propose the resource control algorithms for dynamic resources (i.e., mobile agents themselves) and static resources (i.e., files and data). First, we consider the mobile agent population control problem (MAPC) that requires adapting the number of agents (i.e., agent population) to a given constant fraction of the current number of nodes in a dynamic network. To achieve the adaptiveness for the dynamic change, we borrow an idea from the single species population model, which is a well-known population ecology model. We propose two algorithms for MAPC: the first algorithm requires global information at each node, while the second one requires only the local information. Simulation results show that the both algorithms realize self-adaptation of mobile agent population in dynamic networks, but the second algorithm attains slightly lower accuracy than the first one.

In the replica density control (RDC), we consider the control of static resource. In this problem, we try to control the numbers of multiple static resources at the same time, meanwhile we control the agent population of only one kind in MAPC. The goal of RDC is to adjust the total number of replicas (e.g., copies of original files) to a constant fraction of the current number of nodes and to adjust the number of replicas of each resource equally. The difference between MAPC and RDC is that the target objects are static or dynamic. While mobile agents can migrate in the network, replicas can not. In addition, we try to control the densities of multiple resources in RDC. We propose two algorithms for RDC based on the single species population model with the same approach for MAPC. In the first algorithm, we try to control the replica density for a single resource. In the case that multiple resources coexist in the system, we can control the replica density of each resource by applying the first algorithm to each resource individually. However, this approach needs high network cost (i.e., migration cost of mobile agents) and the exact knowledge at each node about all resources existing in the network. In the second algorithm, the densities of all resources are controlled by a single algorithm without high network cost and the exact knowledge about all resources. Simulation results show that these two algorithms realize self-adaptation of the replica density in dynamic networks.

In the last part of the dissertation, we consider all-to-all information exchange over agents. All-to-all information exchange is one of the most fundamental schemes for constructing mobile-agent-based distributed systems. In this dissertation, we newly introduce *gossip* problem that requires all-to-all information exchange among agents such that all agents obtain the information each agent initially has. While the algorithms for the rendezvous problem, which requires that all agents rendezvous on a node at the same time, can achieve all-to-all information exchange, it takes excessive cost for our objective. In the gossip, each agent p_i can obtain the information of $p_j (\neq p_i)$ by meeting p_j itself or any agent that already has information of p_j. Thus, the gossip is expected to accomplish the all-to-all information exchange with a smaller number of agents' moves than the rendezvous problem. In this dissertation, we investigate the complexity of the mobile agent gossip problem (MAGP) in terms of the total number of moves performed by agents. For several network topologies, we show the asymptotically tight upper and lower bounds for the move complexity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

A distributed computing system consists of a collection of individual autonomous computers that are connected through a network. The networked computers communicate with each other, cooperate toward common tasks or solution of a shared problem, and act autonomously and spontaneously. In the last two decades, boosted by growth of network infrastructures, the distributed computing system has attracted considerable attention as functional, scalable, and available systems. As a system model of distributed computing system, *message passing model*, in which the computers communicate by sending and receiving bounded sequences of bits, is traditionally applied in many systems. However, the design of distributed systems based on the message passing model is growing increasingly complex as the network becomes larger, more dynamic and diverse.

A *mobile agent model* provides one of the most promising frameworks to implement distributed systems. Mobile agents are autonomous programs that can migrate from one node to another in a network, and traverse a distributed system to carry out a task at each node. That is, after an agent processes a task on a node, it can migrate to one of the neighboring nodes with its computational state, and it can continue its computation from the state on the visited node. Since adaptability and flexibility of mobile agents simplify the design of distributed systems, mobile agents have been extensively studied for several years by many researchers. As a programming paradigm, mobile agent model allows a new philosophy of protocol and communication software design. As a computational universe, the model opens a variety of new challenging problems, and many researchers continue to study the principles and algorithms of the mobile-agent-based distributed systems. For example, the rendezvous problem [8, 18, 29, 35, 36, 38] that requires all agent to rendezvous at a node at the same time, the intruder detection problem

[6, 10, 24, 25, 26] that requires agents to capture an intruder in a network, and the black hole search problem [15, 20, 21] that requires agents to detect a despiteous computer in a network are studied by many researchers. All of them have immediate practical relevance and applicability.

1.2 Overview

In this dissertation, we focus on *resource control* in mobile agent systems and *gossiping* among mobile agents.

The resource is an object that consumes processing power, networking bandwidth, and memory space, such as mobile agent itself and data (e.g., files) maintaining on computers. In the distributed systems, tasks that must be processed by agents and requests for files are generally proportional to the network size (i.e., the number of nodes). Therefore, it is expected that the system will achieve a certain performance regardless of the network size if the numbers of agents and replicas of files are a number proportional to the network size. In the resource control, we try to adapt the number of resources to a constant fraction of the current network size in the system. We propose the resource control algorithms for dynamic resources (i.e., mobile agents) and static resources (i.e., data and files) in dynamic networks that change their topologies with time.

Gossip that achieves all-to-all information exchange over mobile agents is one of the most fundamental communication primitives for constructing mobile-agent-based distributed systems. In this dissertation, we propose *gossip* algorithms in static networks, and prove that the gossip has lower cost of agents' migration than the rendezvous algorithms that is a famous communication primitive among mobile agents [8, 18, 29, 35, 36, 38].

In what follows, we show the brief introduction of our contribution for the resource control and the gossip.

1.2.1 Mobile Agent Population Control (MAPC)

In mobile-agent-based distributed systems, generally, a larger number of agents require shorter time to complete the whole task, but increase more system load (e.g., consumption of processing power and network bandwidth). Therefore, it is indispensable to keep an appropriate number of agents for application on the mobile-agent-based system. One of examples is mobile-agent-based network management systems [13, 41, 44] in which agents traverse the network to collect load information of nodes and links, and adjust their load adequately. It is claimed, for network management system MARP[13], the appropriate number of agents is a half of the number of nodes in the network.

While continuous and significant increase in scale, dynamics and diversity of network environments requires highly-adaptive distributed systems, most mobile-agent-based distributed systems (including MARP) have no facility for adjusting the agent population (i.e., the number of agents) for dynamical network change. In dynamic networks, the appropriate agent population for the application changes with time, since network size varies with time. In addition, it is unrealistic, especially for dynamic networks, to assume that each node or each agent knows the current network size and the current agent population on the network. The reason is the computation of such global informations requires much greater cost since computation should be repeatedly executed to follow dynamic changes of the network. Therefore, adjustment of agent population for dynamical change of network size is not an easy task. Thus, developing a general mechanism for adjusting the agent population is invaluable and requires investigation.

Biological systems inherently have self-$*$ properties, such as self-adaptation, self-configuration, self-optimization, and self-healing, to realize environmental adaptation. Thus, several biologically inspired approaches have succeeded in realizing highly adaptive distributed systems. Successful projects include Bio-Networking project[9] and Anthill project[3]. These projects adopt biologically inspired approaches to provide highly adaptive platform for mobile-agent-based computing[4, 42]. Concerning control of agent population, Kaizar et al.[2] proposed a biologically inspired method in agent-based distance vector routing systems: each agent deposits small data (called *pheromone*) on the node it visits. The agent population is controlled depending on concentration of the pheromone. However the dissertation does not clarify the influence of parameters on the resultant population, and thus, it is not expectable how many agents exist in the network after execution with some parameter setting.

Motivated by the above observation, in this dissertation, we formulate *the mobile agent population control problem* (MAPC) in dynamic networks, and present biologically inspired algorithms for the problem. MAPC requires to adapt the agent population to a given constant fraction of the current network size. We propose two distributed algorithms for MAPC. To realize self-adaptation of the agent population to the network size, we borrow an idea from *the single species population model*, which is a well-known population ecology model. This model considers population of a single species in an environment such that individuals of the species can survive by consuming food supplied by the environment. The model is formulated by *the logistic equation* and shows that the population automatically converges to and stabilizes at some number depending on the amount of supplied food.

In the proposed algorithms, agents are regarded as individuals of a single species, and nodes supply food for agents. The algorithms try to adjust the agent population to a constant fraction of the network size by controlling the amount of food supplied at each node. Thus, the crucial

point of the proposed algorithms is the method to determine the amount of food supplied at each node. The amount of food supplied at each node should be determined from a frequency of agents' visits at the node. To validate the claim, under the assumption that every agent traverses the network by random walk, the proposed algorithms determine the amount of supplied food from the stationary probability of random walk. The simulation results of the first algorithm show that the proposed strategy can adequately adjust the agent population.

A debatable point of the first algorithm is that each node uses the link density of the network to determine the amount of supplied food since the stationary probability of random walk depends on the link density. Distributed algorithms generally prefer to avoid using such global information as the link density. Thus, we propose the second algorithm to tackle this problem: in the second algorithm, each node uses local estimation of the link density that is calculated from information of nodes in its neighborhood. The simulation results show that the second algorithm can sufficiently adjust the agent population using only the local information, but with slightly lower accuracy than the first algorithm.

1.2.2 Replica Density Control (RDC)

Resource replication is a crucial technique for improving performance and availability of resource sharing systems, which is one of the most fundamental distributed applications (a well-known example is file sharing on peer-to-peer networks [14, 30]). In resource sharing systems using replication, replicas of an original resource are distributed over the network so that each user can get the requested resource by accessing a nearby replica. Resource replication can reduce communication latency and consumption of network bandwidth, and can also improve availability of the resources even when some of the replicas are unavailable.

As with the agent population, in systems using resource replication, generally, a larger number of replicas require shorter time to reach a replica of the requested resource, but consume more storage of hosts. Therefore, it is indispensable to adjust the number of replicas appropriately for the resource sharing application. For example, resource searching protocol PWQS, which is a quorum-based probabilistic protocol on peer-to-peer networks, has tradeoff between the reach time and the number of replicas [39]. The PWQS requires replicas of each resource in numbers proportional to the network size to attain good performance. Thus, it is natural to keep the number of replicas to a constant fraction of the network size for guaranteeing QoS to a certain extent regardless of the network size. Therefore, adjustment of the number of replicas for dynamical change of network size requires investigation.

In this dissertation, we formulate *the replica density control problem* (RDC) in dynamic networks, and present algorithms for RDC based on *the single species population model*. In

RDC, we tackle to control the numbers of replicas of multiple original resources at the same time since there are usually one or more files or data in the network. RDC requires to adapt the total number of replicas to a given constant fraction of the current network size and to adapt the number of replicas of each resource equally. The difference between MAPC and RDC is that the target objects are static or dynamic. Mobile agents can migrate between nodes, but replicas can not. In addition, we try to control the densities of multiple resources in RDC, meanwhile we controlled the agent population of only one kind in MAPC.

We propose two algorithms for RDC with the same approach as one for MAPC. In the proposed algorithms, replicas are regarded as individuals of a single species, and agents created by nodes supply food for replicas. The first algorithm focuses on controlling the replica density of a single resource. In the first algorithm, the replica density of a single resource is adjusted by controlling the amount of food supplied by agents. The simulation results of the first algorithm show that the proposed strategy can adequately adjust the replica density. In the case that multiple resources coexist in the system, we can control the replica density of each resource by applying this strategy to each resource, that is, each node supplies different food for different resources, and replicas of resource i are supplied food for resource i.

However, since the first algorithm controls the replica density of each resource independently, the number of agents on the network is proportional to the number of resources. Moreover, it is necessary for each node to have exact knowledge about all resources existing in the network. Thus, in the second algorithm, densities of all resources are controlled without the great number of agents and the exact knowledge about all resources. In the second algorithm, every replica can eat any food supplied by agents. Since the technique for supplying food fairly to replicas of every resource is needed, we limit the amount of food supplied to replicas of each resource. The simulation results show that the second algorithm also realizes self-adaptation of the replica density in dynamic networks.

Improvement of system performance depends on replica allocation, which determines the node each replica is allocated to, as well as the number of replicas. We focus only on control of the number of replicas, meanwhile many other researchers focus on replica allocation [22, 34]. The algorithms we propose for RDC can be combined to these allocation protocols since the algorithm can control the number of replicas independently from replicas' allocation.

1.2.3 Mobile Agent Gossip Algorithm (MAGP)

In the mobile-agent-based systems, multiple mobile agents usually process a given task to improve system performance, and no special agent is used. The reason is the special agent makes the fault-tolerance lower. For example, in a network management system, each agent traverses

Table 1.1: Our Contribution for MAGP

Graph	System model	Sense of direction	Total number of agents' moves	
			Upper Bound	Lower Bound
Ring	synchronous	arbitrary	$O(n)$	$\Omega(n)$
	asynchronous		$O(n \log k + n)$	$\Omega(n \log k + n)$
Tree	asynchronous	arbitrary	$O(n)$	$\Omega(n)$
Complete	asynchronous	without	$O(n \log k + n)$	$\Omega(n \log k + n)$
		with	$O(n)$	$\Omega(n)$
Arbitrary	asynchronous	arbitrary	$O(n \log k + e)$	$\Omega(n \log k + e)$

n, e and k are the numbers of nodes, links and agents respectively.

the network to collect load information of nodes and links and informs each node about the information. In such systems, *gossip* is one of the most fundamental tasks in cooperation among mobile agents. It requires one to accomplish all-to-all information exchange over all mobile agents so that each agent can obtain the information all other agent initially has. By the gossip, a negotiation with other agents and an information collection of a whole network are easily realized in distributed systems; for example, in the network management system, each agent can collect load information of a whole network periodically by the periodic execution of the gossip.

A naive approach to implement the gossip is to use *rendezvous* algorithms [8, 18, 29, 35, 36, 38] where all agents are required to rendezvous on a node at the same time; by exchanging all agents' information at the rendezvous point, the gossip can be achieved. However, in some cases, the use of rendezvous algorithms takes excessive cost to implement the gossip. For example, consider the gossip over k agents in a line network of n nodes. Then, to achieve the gossip, the following scenario is possible; let p be the leftmost agent. The agent p moves to the right end of the line to collect information of all agents, and then, returns to the left end with delivering the information to all agents. As the result, each agent can obtain the information of all the agents. While the rendezvous problem has a trivial $\Omega(kn)$ lower bound on the total number of agent moves, the above scenario takes only $2n$ moves. That is, in this case, rendezvous algorithms are quite costly for the gossip.

Motivated by the above observation, in this dissertation, we formulate the *mobile agent gossip problem* (MAGP), and investigate its solutions. Notice that the network is modeled as the static model, that is, the network topology does not change with time. The goal of this problem is that each agent collects the information every other agent initially has with the

smallest number of moves. Different from the rendezvous, MAGP allows relay of information; an agent p_i can obtain the information of another agent p_j directly from p_j or one that already obtain the information of p_j. Therefore, we expect that MAGP can be solved with a smaller number of total moves than the rendezvous.

In this dissertation, we consider MAGP under the assumption that each agent has prior knowledge of neither the number n of nodes nor the number k of agents. Because it can be easily shown from results in [16, 36] that MAGP can not be solved if agents are anonymous, we assume that each agent has unique identifier. Agents can communicate with each other using *whiteboard*, which is a node's local storage where agents on the node can write and read data. We assume that the whiteboard can store the data for controlling agents' traversal, but cannot store the information the other agents have to collect. There are two reasons for this restriction. The first is to avoid introducing huge space for whiteboard. Since we make no assumption on the size of information each agent initially has, each node needs a large memory space to store the information. The second is security reason. It is insecure to write precious information on a whiteboard since any agent can access it. We prove that the move complexity for MAGP cannot be asymptotically improved even if agents are allowed to write the information on whiteboards.

Table 1.1 summarizes the contribution for MAGP. The property of sense of direction implies that every link is locally labeled in a globally consistent way. "Arbitrary" in the column of sense of direction means that the upper and the lower bounds are satisfied whether sense of direction is assumed or not. For synchronous rings, asynchronous trees, asynchronous complete networks, and asynchronous arbitrary networks, we present MAGP algorithms which are asymptotically optimal in terms of the total number of agents' moves. Interestingly, the move complexities of these algorithms are sublinear in k. Especially, for synchronous rings, asynchronous trees, and asynchronous complete networks with sense of direction, the move complexities of the proposed algorithms are independent of k. Since the trivial lower bound for the rendezvous problem on trees and rings is $\Omega(kn)$, our results imply that MAGP inherently has lower move complexity than the rendezvous problem.

A part of these results derives from the relation between MAGP and node leader election (NLEP) in message passing systems. More precisely, we show that some of the upper and lower bounds for MAGP are obtained from those for NLEP. Some of the upper bounds for MAGP are proved by the reduction of MAGP to NLEP; NLEP is the leader election among nodes in message passing systems. Thus, we present an algorithm of simulating a message passing system in a mobile agent system. The proposed algorithm requires only $O(1)$ agents' moves per message. In contrast, the lower bounds for MAGP are obtained by the reduction of NLEP to MAGP.

1.3 Organization

This dissertation consists of six chapters. In Chapter 2, we describe the model of mobile agent systems. In Chapter 3 and 4, we define the mobile agent population control problem (MAPC) and the replica density control problem (RDC), and propose the algorithms for each problems respectively. In Chapter 5, we deals with the mobile agent gossip problem (MAGP). We conclude this dissertation in Chapter 6.

Chapter 2

Mobile Agent Systems

A network is modeled as an undirected labeled graph $G = (V, E, \lambda)$, where V and E are respectively the node set and the link set in G. A link in E connects two distinct nodes in V. The link between nodes u and v is denoted by l_{uv} or l_{vu}. On each node, each incident link is locally labeled. Let $\lambda_u(l)$ be the label of link l on a node u ($\lambda_u(l) \in \{1, 2, \ldots, deg_u\}$, where deg_u is the number of u's incident links). The numbers of nodes and links in G are respectively denoted by n and e (i.e., $n = |V|, e = |E|$). An agent is an autonomous state machine that can migrate from one node to another in the network. Agents on a node $u \in V$ can migrate to a node $v \in V$ only when link l_{uv} is contained in E.

Each node v is provided with a *whiteboard*, i.e., local storage where agents on v can write and read data. When multiple agents on a node execute their operations, the operations are sequentially executed in an arbitrary order.

An agent p_i on each node v performs a sequence of the following operations;

- *read*(v) : agent p_i reads data written on node v's whiteboard, and executes local computation.
- *write*(v, d) : agent p_i writes data d on node v's whiteboard.
- *move*($v, \lambda_v(l)$) : agent p_i moves to one of v's neighbor nodes through the link labeled $\lambda_v(l)$. If $\lambda_v(l)$ is zero, p_i stays on v.

We assume that these three operations are executed atomically in one action. Agents are said to be *asynchronous* if migration time and local processing time of agents are unpredictable but finite. In contrast, agents are *synchronous* if its execution is partitioned into rounds; in each round, every agent arrives at a node, accesses the whiteboard and executes local computation on the node, and stays on the node or starts migration to one of the neighboring nodes. The agents arrive at the destination nodes by the end of the current round.

A state of an agent is represented by a set of variables the agent has and a set of information the agent has collected. A location of an agent is represented by a node the agent stays or a link the agent migrates on. A state of a node is represented by the state of its whiteboard and a set of variables the node has. A system configuration C is represented by the states of all nodes, the states of all agents, and the locations of all agents. A system configuration is changed by events of agents and nodes. (e.g., read and write on a whiteboard, migration from and arrival at a node). Let C_0 be an initial configuration of a system and Ev_i be a set of events that occur simultaneously at the configuration C_i. An execution of a distributed system is an alternate sequence of configurations and sets of events $EX = C_0, Ev_0, C_1, Ev_1, C_2, \cdots$, such that occurrence of events Ev_{i-1} changes the configuration from C_{i-1} to C_i.

Chapter 3

Mobile Agent Population Control

In mobile-agent-based systems, a larger number of agents generally require shorter time to complete the whole task but increase more system load (e.g., consumption of processing power and network bandwidth). Therefore, it is indispensable to keep an appropriate number of agents for the application on the mobile-agent-based system.

In this chapter, we considers the mobile agent population control problem (MAPC) in dynamic networks: it requires adjusting the number of agents to a constant fraction of the current network size. This chapter is organized as follows. In Section 3.1, we present the model of dynamic systems, and define MAPC. The proposed algorithms are inspired by the single species population model, which is a well-known population ecology model. We describe the single species population model in Section 3.2. In Sections 3.3, we propose the first distributed algorithm for MAPC and show its simulation results. The modified algorithm is presented and simulated in Section 3.4. Section 3.5 concludes this chapter.

3.1 Preliminaries

3.1.1 System Models

In this chapter, we consider *dynamic networks* such that its node set and its link set vary with time. To define dynamic networks, we introduce discrete time and assume that each time is denoted by a non-negative integer in a natural way: time 0 denotes the initial time, time 1 denotes the time immediately following time 0 and so on.

Formally, a dynamic network at time t is denoted by $G(t) = (V(t), E(t))$ where $V(t)$ and $E(t)$ are respectively the node set and the link set at time t. The numbers of nodes and links at time t are represented by $n(t)$ and $e(t)$ respectively. In dynamic networks, agents on node $u \in V(t)$ at time t can start migrating to node $v \in V(t)$ only when link l_{uv} is contained in $E(t)$.

The agent reaches v at time $t + \Delta$ only when the link l_{uv} remains existing during the period from t to $t + \Delta$, where Δ is an integer representing *migration delay* between the nodes. The agent migrating from u to v is removed from the network when the link l_{uv} disappears during the period from t to $t + \Delta$.

In this chapter, we assume that each of nodes and agents has a *local clock* that runs at the same rate as the global time. However, we make no assumption on the local clock values: the difference between the local clock values in the system is unbounded.

As mentioned in Section 2, an agent p on node v can execute some operates on the whiteboard of u. Moreover, in this chapter, node v also executes the following operations:

- *read* : node v reads data written on its whiteboard, and executes local computation.
- *write*(d) : node v writes data d on its whiteboard.

We assume that these operations are executed atomically. Besides the above operations, each agent can execute operations to create new agents and to kill itself and each node can also execute operations to create new agents.

When agents reside on a node, the agents and the node may have operations they can execute. For execution semantics, we assume that the agents and the node execute their operations sequentially in an arbitrary order. We also assume that the time required to execute the operations can be ignored, that is, we consider all the operations are executed sequentially but at an instant time.

Concerning the migration patterns of agents, we assume that each agent makes a *random walk* independently: an agent migrates from one node to one of its neighboring nodes with equal probability. In real agent systems, migration patterns of agents differ from application to application. In this chapter, however, we focus on a certain control of agents rather than the actual applications invoked by the agents. Thus, we adopt random walks as the migration patterns since we do not want to consider a particular application. It is worthwhile to mention that a random walk is one of the most typical migration patterns, and that some real agent systems adopt random walk as their migration patterns[13].

3.1.2 Mobile Agent Population Control (MAPC)

In this chapter, we consider the *mobile agent population control problem* (MAPC). The goal of MAPC is to control the number of agents so that the ratio between the number of agents (called *agent population* hereinafter) and the number of nodes (called *network size* hereinafter) is kept to be a given constant. MAPC is defined as follows. In the definition, let $k(t)$ be the agent population in the network $G(t)$ at time t.

Definition 1 [Mobile Agent Population Control Problem (MAPC)] The mobile agent population control problem is solved iff the agent population $k(t)$ at time t satisfies the following equality for a given constant γ $(0 < \gamma \leq 1)$.

$$k(t) = \gamma \cdot n(t)$$

In this dissertation, we assume that the constant γ is initially given to every node. We consider distributed systems such that mobile agents are distributed over the networks and nodes can leave from or join in the networks. In such environment, it is obviously impossible to keep satisfying the above equation all the time. Thus, our goal is to propose distributed algorithms that realize quick convergence to the target population and stability at the target population if the network does not change.

3.2 Single Species Population Model

In this subsection, we introduce the *single species population model* in the population ecology as the basis of our algorithm. This model considers an environment with a single species such that individuals of the species can survive by consuming food supplied by the environment. The model formulates the population growth of the species in the environment, and shows that the population (i.e., the number of individuals) in the environment automatically converges to and stabilizes at some number depending on the amount of food supplied by the environment.

We present more details of the single species population model. Each individual of the species periodically needs to take a specific amount of food to survive. That is, if an individual can take the specific amount of food then it can survive. Conversely, if an individual cannot take the specific amount of food then it dies. Moreover, in the case that an individual can take a sufficient amount of extra food, then it generates progeny. Consequently, the followings hold: The shortage of supplied food results in decrease in the population. Conversely, the excessive amount of food results in increase in the population.

The single species population model formulates the above phenomena as follows: Let $p(t)$ be the population at time t. The single species population model indicates that the *population growth rate* at time t is represented by the following nonlinear first-order differential equation known as the *logistic equation*[31]:

$$\frac{\Delta p(t)}{\Delta t} = p(t) \cdot g(t) = p(t) \cdot h(f_a(t) - f \cdot p(t)),$$

where $f_a(t)$ is the amount of food supplied by the environment at time t, f is the amount of food consumed by one individual to survive, and h is a positive real constant.

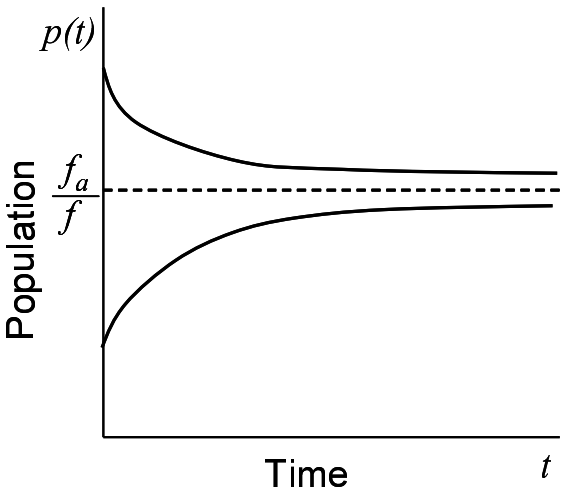


Figure 3.1: Convergence to the equilibrium point in logistic equation

The *per capita growth rate* $g(t)$ at time t is represented by

$$g(t) = h(f_a(t) - f \cdot p(t)).$$

The expression $f_a(t) - f \cdot p(t)$ represents the difference between the amounts of supplied food and consumed food. When the supplied food exceeds the consumed food, $g(t)$ takes a positive value proportional to the difference, that is, the positive per capita growth rate $g(t)$ is proportional to the amount of the surplus food. A scarcity of the supplied food causes a negative value of $g(t)$ proportional to the difference, that is, the negative per capita growth rate $g(t)$ is proportional to the shortage of the supplied food.

The logistic equation has two equilibrium points of the population size $p(t)$: $p(t) = 0$ and $p(t) = f_a(t)/f$. That is, the population remains unchanged, when the population size is at the equilibrium points. The equilibrium point $p(t) = f_a(t)/f$ represents the maximum population that the environment can keep, and is called the *carrying capacity* of the environment.

If the population is larger (resp. smaller) than the carrying capacity then the population decreases (resp. increases). Once the population reaches the carrying capacity, then it remains unchanged (see Fig.3.1). Consequently, the single species population model implies that the population eventually converges to and stabilizes at the carrying capacity. Notice that the carrying capacity depends on the amount of food supplied by the environment.

3.3 Algorithm for MAPC

3.3.1 Algorithm

In this subsection, we present an algorithm for MAPC. The algorithm is inspired by the single species population model: agents are regarded as individuals of a single species, and a network is regarded as an environment. That is, agents need to consume food to survive and the food is supplied by nodes of the network.

The goal of MAPC is to adjust the agent population $k(t)$ to $\gamma \cdot n(t)$. Remind that the single species population model shows that the number of individuals converges to and stabilizes at the carrying capacity. Thus, the algorithm tries to adjust $k(t)$ to $\gamma \cdot n(t)$ by adjusting the carrying capacity to $\gamma \cdot n(t)$.

In the algorithm, we introduce time interval of some constant length denoted by T. Behavior of each node and each agent can be divided into a series of the time intervals. It should be noticed that the start time of the intervals at different nodes or agents are not synchronized: a node or an agent may start a new time interval while another is in the middle of its time interval.

The primary behavior of nodes and agents is simple: each node supplies food every T time units (i.e., at the beginning of each time interval). Each agent traverses the network by a random walk and consumes food the visited nodes have. The agent can survive into the next time interval if it can take a specific amount of food, denoted by AF, during the current time interval. The agent kills itself (i.e., removes itself from the network) if it cannot get food of amount AF during the time interval. In addition, each agent creates new agents by consuming surplus food of amount AF. This idea derives from the fact that the positive per capita growth rate $g(t)$ in the single species population model is proportional to the amount of surplus food.

Figure 3.2 shows the detailed behavior of nodes and agents in the proposed algorithm for MAPC.

In what follows, we explain some technical parts of the node's behavior. Each node supplies an appropriate amount of food every T time units. Now, we consider the amount of food $f(v,t)$ that node v should supply at time t. As stated in the above, the algorithm tries to adjust the carrying capacity to $\gamma \cdot n(t)$. The single species population model shows that the carrying capacity is represented by $f_a(t)/f$. Since $f_a(t)$ corresponds to the total amount of supplied food on the whole network $\sum_{v \in V(t)} f(v,t)$ and f corresponds to the amount of food AF consumed by an agent to survive, the following equation should be satisfied:

$$\sum_{v \in V(t)} f(v,t) \quad = \quad \gamma \cdot n(t) \cdot AF.$$

If each node supplies the same amount of food, $\gamma \cdot AF$, the above equation can be easily

Parameters of the algorithm

AF : the amount of food to be consumed by an agent to survive

T : the time interval of behavior of node and agent

Behavior of node v

Variables of node v

$time_v$: local clock time

/* its value automatically increases at the same rate as the global time */

Data written on whiteboard at node v

$food_v$: the amount of food

$surplus\_food_v$: the amount of surplus food

- at the beginning of each time interval (i.e., when $time_v$ **mod** T =0 holds)

  $surplus\_food_v := food_v$

  $food_v := \frac{deg_v}{2 \cdot e(t)} \cdot \gamma \cdot n(t) \cdot AF$

- on suspicion of the total extinction (i.e., when no agent visits v during the last $\frac{X}{\gamma \cdot n(t) \cdot q(v)}$ time units)

  create one agent

Behavior of agent p

Variables of agent p

eat_food_p : the amount of food that p consumes from *food* of nodes

$eat\_surplus\_food_p$: the amount of food that p consumes from *surplus_food* of nodes

$time_p$: local clock time

/* its value automatically increases at the same rate as the global time */

- when p is created

  $eat\_food_p := 0.0$

  $eat\_surplus\_food_p := 0.0$

  $time_p := 0$

- on arrival at node v

  if $(eat\_food_p < AF)$ **then**

  $y := \min\{AF - eat\_food_p, food_v\}$

  $eat\_food_p := eat\_food_p + y$

  $food_v := food_v - y$

  if $(surplus\_food_v > 0)$ **then**

  $y' := \min\{AF - eat\_surplus\_food_p, surplus\_food_v\}$

  $eat\_surplus\_food_p := eat\_surplus\_food_p + y'$

  $surplus\_food_v := surplus\_food_v - y'$

  if $(eat\_surplus\_food_p = AF)$ **then**

  create one agent

  $eat\_surplus\_food_p := 0.0$

  migrate to one of the neighboring nodes

- at the end of each time interval (i.e., when $time_p$ **mod** T =0 holds)

  if $(eat\_food_p < AF)$ **then** kill itself

  else $eat\_food_p := 0$ /*p survives into the next time interval */

Figure 3.2: Algorithm for MAPC

satisfied. However, since each agent makes a random walk, frequencies of agents' visits vary from node to node and, thus, such uniform allocation of food over the network cannot adjust the agent population appropriately: nodes with greater frequencies face the food shortage and nodes with

less frequencies have surplus food that is never consumed. To resolve this problem, the amount of food each node supplies should be proportional to the frequencies of agents' visits. Theory of Markov chain shows that the *stationary probability* $q(v)$ of an agent at node v is represented by $q(v) = deg_v/(2 \cdot e(t))$, where deg_v is the number of links connecting to v (called *degree* of v hereinafter). Therefore, node v should supply food of amount

$$q(v) \cdot \gamma \cdot n(t) \cdot AF$$

at every T time units.

When a node supplies new food and goes to the next time interval, the node stores the surplus food of the current time interval into variable $surplus\_food$. The surplus food is consumed by agents to create new agents.

Another problem we should consider is the *total extinction* of agents. We consider dynamic networks. When a node v leaves from the network at time t, agents on node v or on links connecting to v at time t are also removed from the network. This may cause the total extinction of agents. To resolve the problem, each node v creates a new agent if v is not visited by any agent during a sufficiently long time period. The length of the period should be also determined from the stationary probability $q(v)$ of an agent. In the algorithm, each node v creates a new agent when no agent visits v during $X/(\gamma \cdot n(t) \cdot q(v))$ time units, where X is a constant.

3.3.2 Simulation Results

In this subsection, we present simulation results to show that the proposed algorithm can effectively adjust the agent population.

In the simulation, we assume that each agent repeatedly executes the following actions: each agent stays at a node for one time unit, and then migrates to one of its neighboring nodes by a random walk. We also assume that the migration delay between any pair of neighboring nodes is two time units. The following values are initialized randomly:

- the initial locations of agents
- the initial values of the local clocks (i.e., $time_v, time_p$)
- the initial amounts of food that nodes have (i.e., $food_v$)
- the initial amounts of food that agents have consumed in the current time interval (i.e., eat_food_p)

The initial amounts of surplus food that nodes have (i.e., $surplus\_food_v$) and the initial amounts of surplus food that agents have consumed in the current time interval (i.e., $eat\_surplus\_food_p$)

are set to 0. To resolve the total extinction of agents, each node v creates a new agent when no agent visits v during $50/(\gamma \cdot n(t) \cdot p(v))$ time units.

We present the simulation results for *random networks* and *scale-free networks*. Random networks are used in evaluations of distributed algorithms by many researchers. Scale-free networks are a specific kind of networks such that some nodes have a tremendous number of connections to other nodes, whereas most nodes have just a handful. The degree distribution follows a power law of k: the number of nodes with degree k is proportional to k^{-r}, where r is a positive constant. A scale-free network is said to be a realistic model of actual network structures [1, 5].

To show the adaptiveness of the proposed algorithm, we also show the difference ratio of the agent population: the ratio is defined by

$$|\gamma \cdot n(t) - k(t)|/(\gamma \cdot n(t)) \tag{3.1}$$

and represents the ratio of difference between the adjusted and the target numbers of agents to the target number.

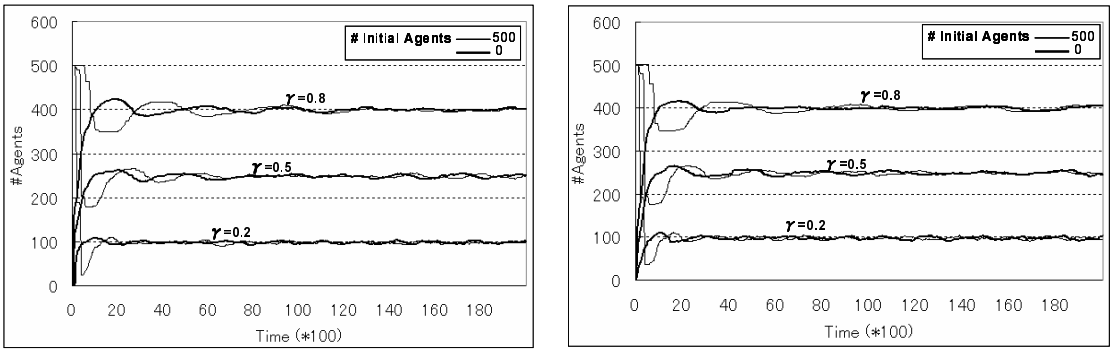
Simulation results for static networks

Figure 3.3 shows the experimental results for "static" random networks and "static" scale-free networks where nodes and links of the networks remain unchanged. In the simulation, the length T of the time interval is set to 200, and the number $n(t)$ of nodes is fixed at 500 during the simulation. Random graphs with n nodes are generated as follows: each pair of nodes is connected with probability of $5.0/(n-1)$. Scale-free networks are generated using the incremental method proposed by Balabasi and Albert [1]. More precisely, starting with 3 nodes, we add new nodes one by one. When a new node is added, three links are also added to connect the new node to three other nodes, which are randomly selected with probability proportional to their degrees.

Figure 3.3 shows transition of the agent population $k(t)$ with time t. It shows the simulation results for six combinations of three values of γ (0.8, 0.5, and 0.2), and two initial agent populations $k(0)$ (500 and 0). These simulation results show that the agent population quickly converges to the equilibrium point, and has small perturbation after the convergence. In addition, the average of the difference ratio is about 0.01.

Simulation results for dynamic networks

Figure 3.4 and Figure 3.5 show the experimental results for "dynamic" random networks and "dynamic" scale-free networks where nodes and links of networks vary with time. When a new node joins in the network, the new node is connected to other nodes with probability



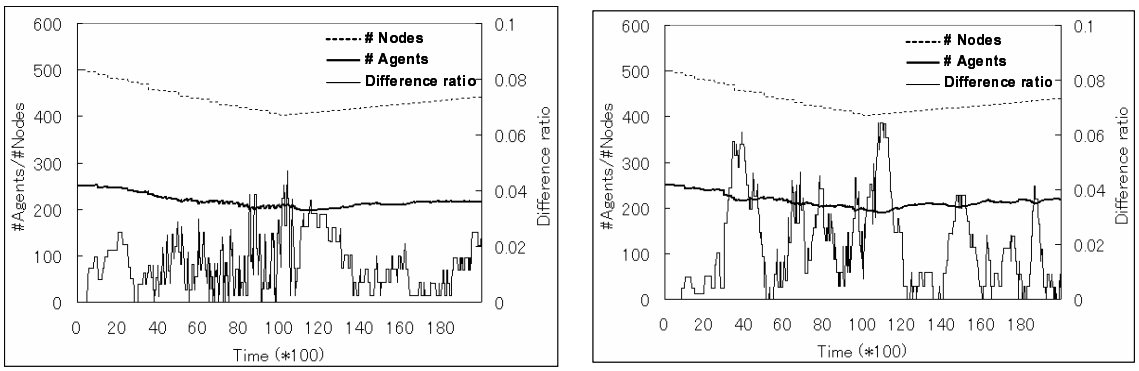
a. random networks ($n(t) = 500$) b. scale-free networks ($n(t) = 500$)

Figure 3.3: Simulation results of the MAPC algorithm on static networks

$5.0/n(t)$ for each other node on random networks, and the node is connected to three other nodes randomly selected with probability proportional to their degrees on scale-free networks. When a node v leaves from the network, the links connecting to v are also removed from the network, and agents on node v or these links are also removed from the network. So, some network partitions may be caused by leave of nodes, but it is not matter for the proposed algorithm. The algorithm can control agent population in a subnetwork; the agents can traverse and eat food on nodes only in the subnetwork. In the simulation, each node leaves from the network with a constant probability for random networks. For scale-free network, the leave of a hub node having tremendous number of connections is undesirable for us since a lot of partitions will be caused. Therefore, for scale-free network, each node leaves with probability inversely proportional to its degree, that is, node v leaves from the network with probability $avg\_deg \cdot p^*/deg_v (0 < p^* < 1)$, where avg_deg is the average degree of the network. So, there is little chance that some hub nodes leave from the network.

Figure 3.4 shows simulation results for dynamic networks with continuous and gradual changes: some nodes join in the network and some nodes leave from the network constantly. In this simulation, the initial network size $n(0)$ is 500, and the following dynamical changes occur every 500 time units. In the first half (from time 0 to time 10,000) of the simulation, two new nodes join in the network with probability 0.05 and each node v leaves from the network with probability 0.01 for random networks and with probability $avg\_deg \cdot 0.01/deg_v$ for scale-free networks. In the second half (from time 10,000 to time 20,000), two new nodes join in the network with probability 1.0 and each node v leaves from the network with probability 0.0001 for random networks and with probability $avg\_deg \cdot 0.0001/deg_v$ for scale-free networks.

In the simulation results of Figure 3.4, the length T is set to 200, the value of γ is set to



a. random networks
$(n(0) = 500, k(0) = 250, \gamma = 0.5)$

b. scale-free networks
$(n(0) = 500, k(0) = 250, \gamma = 0.5)$

Figure 3.4: Simulation results of the MAPC algorithm on dynamic networks with gradual changes

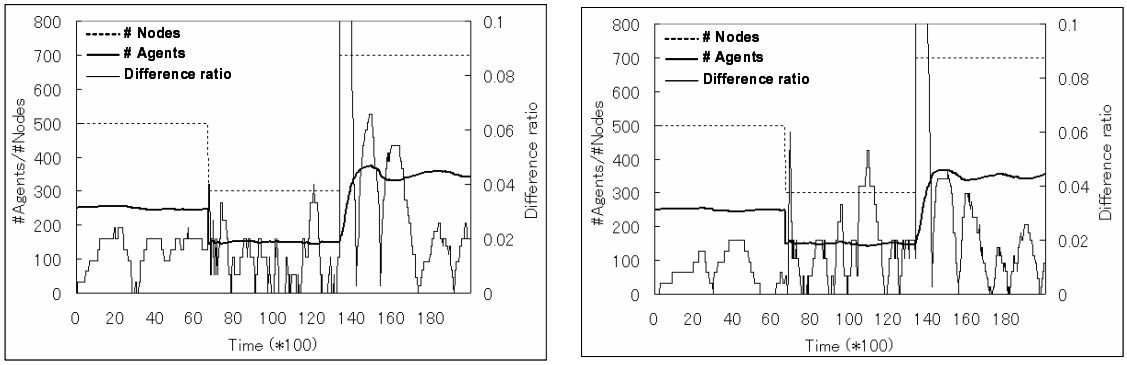
0.5, and the initial agent population $k(0)$ is set to 250. Since the difference ratio is kept to be less than 0.07 and does not widely diverge from 0, the simulation results show that the agent population is adaptively adjusted in response to changes in the network size.

Figure 3.5 shows the simulation results for dynamic networks with drastic changes: in a short term, a large number of nodes leave from the network or join in the network. In this simulation, the initial network size $n(0)$ is 500, and 200 nodes leave from the network at time 7,000 of the simulation, and 400 nodes join in the network at time 14,000 of the simulation. The leaving nodes are chosen randomly.

In the simulation results of Figure 3.5, the length T is set to 200, the value of γ is set to 0.5, and the initial agent population $k(0)$ is set to 250. While the difference ratio widely diverges from 0 immediately after the drastic changes of the network, it quickly converges to the target number. The difference ratio is kept to be less than 0.04 after the convergence.

Simulation results on lifetime and utilization rate of agents

The goal of MAPC is to adjust the agent population to a given ratio of the network size. However, from the point of application view, lifetime and utilization rate of agents are also very important. In real applications, agents traverse the network to complete some tasks such as collection and/or distribution of information. Each task arises on each node and should be processed by agents that visit the node. Thus, such applications require that lifetime of agents should be sufficiently long to complete the task, and that each agent should process many tasks efficiently.



a. random networks
$(n(0) = 500, k(0) = 250, \gamma = 0.5)$

b. scale-free networks
$(n(0) = 500, k(0) = 250, \gamma = 0.5)$

Figure 3.5: Simulation results of the MAPC algorithm on dynamic networks with drastic changes

Lifetime lt_p of agent p is defined to be the time length from its creation to its elimination, i.e., $lt_p = td_p - tb_p$, where td_p is the time when p is removed and tb_p is the time when p is created. Table 3.1 shows the average lifetime of agents of ten trials. To focus on the lifetime of agents after convergence of the agent population to the target value, the initial agent population $k(0)$ is set to the equilibrium point. The simulation results show that lifetime quickly becomes longer when the length of the time interval T becomes longer. Therefore, by setting an appropriate value to T, it is strongly expected that lifetime of each agent becomes sufficiently long.

Next, we show simulation results of utilization rate of agents. Utilization rate of an agent is defined to be the ratio of the actual number of tasks the agent processes to the possible maximum number of tasks that the agent can process. We assume that each agent processes one task on each visit to a node, if the node has tasks. Thus, the maximum number of tasks an agent can process in the interval of T time units is $T/3$ since two thirds of the interval is absorbed by migration. The tasks arise on each node at every T time units. Since the number of agents should be kept to be $\gamma \cdot n(t)$, the number of tasks that can processed by the whole agents during the time interval of T time units is $(\gamma \cdot n(t) \cdot T)/3$. Therefore, the number of tasks that arise on node v at every T time units is determined to be $q(v) \cdot (\gamma \cdot n(t) \cdot T)/3$ by a similar reason to that for the amount of food. Each agent traverses the network and process one task the visited node has. Utilization rate u_p of agent p for a time interval is defined by $w_p/(T/3)$, where w_p is the number of tasks processed by agent p during the time interval.

Table 3.2 shows the average utilization rate of agents of ten trials. The initial agent population $k(0)$ is set to the equilibrium point. The simulation results show that utilization rate are sufficiently high. When the value of γ is set to more than 0.5, almost all average utilization rate

		T		
		100	200	400
n	200	4767	30102	129273
	500	4906	32996	148427
	1000	4954	35446	173367

a. random networks

		T		
		100	200	400
n	200	4708	35581	213076
	500	4703	40152	277297
	1000	4727	41421	366194

b. scale-free networks

Table 3.1: Lifetime of agents

		γ		
		0.2	0.5	0.8
n	200	0.849	0.900	0.916
	500	0.845	0.898	0.915
	1000	0.845	0.898	0.915

a. random networks

		γ		
		0.2	0.5	0.8
n	200	0.857	0.908	0.923
	500	0.859	0.906	0.922
	1000	0.857	0.905	0.921

b. scale-free networks

Table 3.2: Utilization rate of agents

of agent becomes more than 90% .

Besides the simulations on random networks and scale-free networks presented in this section, we did simulations on several other networks such as complete networks, lollipop networks and star networks, and obtained similar results on these networks.

3.4 Modified Algorithm for MAPC with Estimated Link Density

In the algorithm presented in Section 3.3, each node v supplies the amount $q(v) \cdot \gamma \cdot n(t) \cdot AF$ of food every T time units, where $q(v)$ is the stationary probability of an agent at node v and is represented by $q(v) = deg_v/(2 \cdot e(t))$. The algorithm assumes that the amount of food supplied at each node v can be locally computed by v. It seems natural to assume that each node v initially knows the input parameter γ, the constant AF, and its own degree deg_v. However, the numbers of nodes $n(t)$ and links $e(t)$ are global information of the network, and thus, it is unrealistic, especially for dynamic networks, to assume that each node initially knows these values. In static networks, $n(t)$ and $e(t)$ can be computed by a simple distributed algorithm (e.g., the wave algorithm in [43]). However, computation of $n(t)$ and $e(t)$ essentially requires the number of messages proportional to $n(t)$ because $n(t)$ and $e(t)$ are global functions over

the whole network. In dynamic networks, the computation requires much greater cost since computation of $n(t)$ and $e(t)$ should be repeatedly executed to follow dynamic changes of the network.

In this section, to save the cost (e.g., the number of messages) for computing the amount of food supplied by each node, we propose a method for locally estimating the amount of supplied food, and evaluate its accuracy by simulation. We also show simulation results of the mobile agent population control using the estimated values.

We can see that $n(t)$ and $e(t)$ appear in the form of $n(t)/e(t)$ in the formula that determines the amount of food supplied at node v:

$$(deg_v/(2e(t))) \cdot \gamma \cdot n(t) \cdot AF.$$

The form $n(t)/e(t)$ is the inverse of the *link density* of the network. From the observation, we can expect that the amount of food supplied by each node can be locally computed, because the link density $e(t)/n(t)$ is expected to be estimated from local information if the network has some uniformity. In the followings, we propose a method for locally estimating the link density of the network.

3.4.1 Local Estimation of the Link Density

To attain locality of the estimation, we concentrate our attention to the methods such that each node v estimates the link density only from information on its *h-neighbors* for some small constant h: the h-neighbors of v are the nodes at most h away from the node v. Let $Nb_h(v,t)$ be the set of h-neighbors of node v at time t.

The link density of the network $e(t)/n(t)$ is represented by

$$\begin{aligned} e(t)/n(t) &= \sum_{u \in V(t)} deg_u/(2n(t)) \\ &= (1/2) \cdot \sum_{u \in V(t)} deg_u/n(t). \end{aligned}$$

Since $\sum_{u \in V(t)} deg_u/n(t)$ is the average of degrees over all nodes, the equation leads us to the following straightforward local estimation of the link density by node v at time t:

$$(1/2) \cdot \sum_{u \in Nb_h(v,t)} deg_u/|Nb_h(v,t)|.$$

However, when each node determines the amount of supplied food from this estimation, the total amount of the supplied food over the whole network is smaller than the target value. The shortage of supplied food is caused by the tendency that the estimation of the link density tends

to be higher than the actual link density in the whole network. The reason why the estimation demonstrates such tendency can be intuitively explained as follows: the degree deg_v is counted in all the link density values estimated by the h-neighbors of v. Since a node with a higher degree tends to have a larger number of h-neighbors, the above estimation tends to be higher than the actual link density. On the assumption that the number of h-neighbors of each node v is proportional to its degree deg_v, we make correction by multiplying a weight coefficient $1/deg_v$ to each node v. The correction gives us the following estimation of the link density by node v at time t:

$$\begin{aligned} & (1/2) \cdot \sum_{u \in Nb_h(v,t)} (deg_u/deg_u) / \sum_{u \in Nb_h(v,t)} (1/deg_u) \\ = \; & (1/2) \cdot |Nb_h(v,t)| / \sum_{u \in Nb_h(v,t)} (1/deg_u) \end{aligned}$$

We expect that the above formula gives us better estimation especially in large-scale of random networks with small degrees.

Based on the above estimation, the algorithm in Section 3.3.1 is modified so that each node v should supply the following amount of food every T time units.

$$\frac{\sum_{u \in Nb_h(v,t)} (1/deg_u) \cdot deg_v \cdot \gamma \cdot AF}{|Nb_h(v,t)|}$$

3.4.2 Simulation Results using the Estimated Link Density

Now, we present simulation results to show accuracy of the estimation. Table 3.3 shows the value $(s'/s) \cdot 100$ where s (resp. s') is the total amount of supplied food over the whole network by the algorithm in Section 3.3.1 (resp. the modified algorithm). The table shows the simulation results for nine combinations of three values of h (1, 2 and 3), and three network sizes $n(t)$ (200, 500 and 1000). Experimental results in the table are average values of ten different networks, and these networks are created by a similar way to that in Section 3.3.2.

For random networks, the simulation results show that the modified algorithm supplies almost the same amount of food as the algorithm in Section 3.3.1. Larger values of h give closer approximation of the total amount of supplied food, but require larger cost to gather information from h-neighbors. We can conclude from the simulation results that setting $h = 1$ or $h = 2$ gives sufficiently close approximation.

From the simulation results for scale-free networks, we can see that the total amount of supplied food determined from 1-neighbors shows the closest approximation, and that larger value of h cannot improve the approximation. This is because scale-free networks violates the

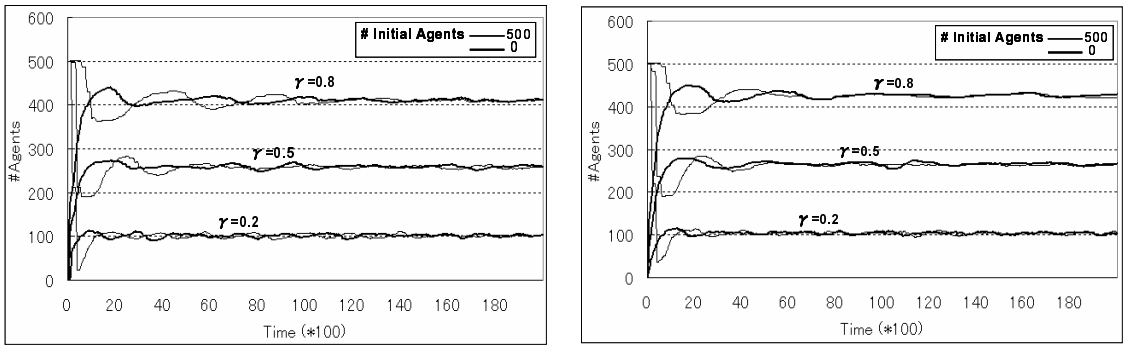
		h		
		1	2	3
n	200	103.40	103.07	107.60
	500	103.36	102.01	103.31
	1000	103.46	101.97	102.01

a. random networks

		h		
		1	2	3
n	200	106.33	120.26	136.18
	500	106.59	118.64	134.76
	1000	106.54	115.24	130.58

b. scale-free networks

Table 3.3: Ratio between the total amounts of food supplied by two MAPC algorithms



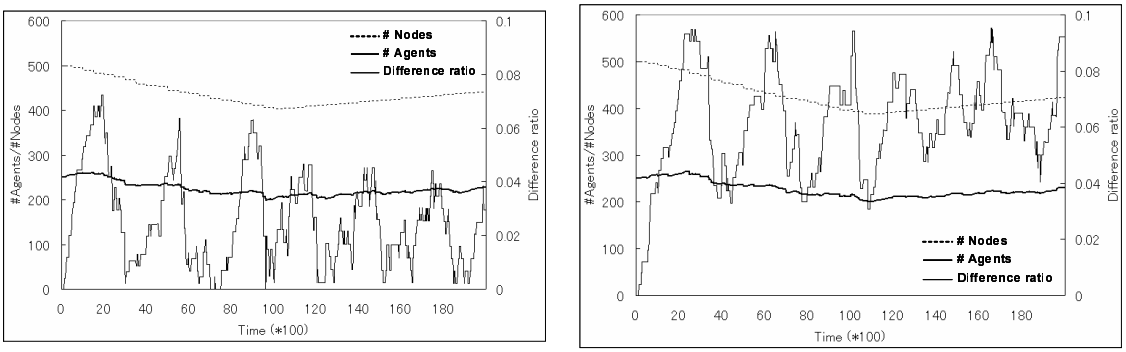
a. random networks ($n(t) = 500$) b. scale-free networks ($n(t) = 500$)

Figure 3.6: Simulation results of the MAPC algorithm with the estimated link density on static networks

assumption that the number of h-neighbors of node v is proportional to the degree deg_v for h of two or more. We can conclude from the simulation results that setting $h = 1$ is appropriate for scale-free networks.

Now, we show simulation results of the modified algorithm. In the simulation, the behavior and the initial locations of agents are determined in the same way as that in Section 3.3.2. Figure 3.6 and Figure 3.7 show the experimental results on the agent population $k(t)$. Figure 3.6 is for "static" networks and Figure 3.7 is for "dynamic" networks. The networks are created by the way presented in Section 3.3.2. In the simulation, the length T of the time interval is set to 200. The network size $n(t)$ is fixed at 500 during the simulation for "static" networks. For "dynamic" networks, networks change in the same way as the simulation in Section 3.3.2. From the above observation on accuracy of the estimated link density, each node uses the link density estimated from 2-neighbors for random networks, and the link density estimated from 1-neighbors for scale-free networks.

The simulation results in Figure 3.6 show that the agent population converges to and stabi-



a. random networks
$(n(0) = 500, k(0) = 250, \gamma = 0.5)$

b. scale-free networks
$(n(0) = 500, k(0) = 250, \gamma = 0.5)$

Figure 3.7: Simulation results of the MAPC algorithm with the estimated link density on dynamic networks

lizes at slightly higher population than the target. The difference is caused by excess of supplied food presented in Table 3.3, and the difference ratio of population is less than 0.02 in random networks and is less than 0.07 in scale-free networks. The simulation results in Figure 3.7 also show that the agent population is adaptively adjusted in response to changes in the network size. The agent population is slightly higher than the target, but the difference ratio of population is still less than 0.1. Consequently, we can conclude that the modified algorithm can also adjust the agent population using only the local information, but its accuracy is slightly lower than the algorithm using the global information $n(t)$ and $e(t)$.

3.5 Concluding Remarks

In this Chapter, we have proposed two distributed algorithms for MAPC that requires adapting the number of agents to a given constant fraction of the current number of nodes in a dynamic network. These algorithms are inspired by the single species population model, which is well-known in the field of the population ecology. The simulation results show that the proposed algorithms can adequately adjust the number of agents in dynamic networks. In addition, from the simulation results, the lifetime of each agent becomes sufficiently long by setting an appropriate value to algorithm parameter T and the utilization rate of agent is also satisfactory.

These two algorithms are different in the information each node requires. That is, the first algorithm requires each node to know the numbers of nodes and links. On the other hand, in the second algorithm, each node needs no global information on networks, but locally estimates the link density. The simulation results have shown that the second algorithm can also adjust

the number of agents but with slightly lower accuracy than the first algorithm.

MAPC is a fundamental problem common to the wide range of mobile-agent-based distributed systems. In the proposed algorithms, on the assumption that each agent makes random walk, each node utilizes the stationary probability of an agent to determine the amount of food it supplies. Our future work is to consider other migration patterns than random walks. The key idea of our algorithms is to adjust agent population by controlling the total amount of food supplied at each node. In the first algorithm, the amount of food supplied at each node is determined from the frequency of agents' visits at the node. Therefore, the first algorithm is expected to control agent population appropriately under other migration patterns, as long as each node can know the frequency of agents' visits at the node. On the other hand, the second algorithm require the assumption that frequencies of agents' visits at each node is inversely proportional to the number of links $e(t)$, which does not generally hold for any migration patterns. Thus, to apply the second algorithm, we have to modify it not to use such assumption.

Chapter 4

Replica Density Control

In Chapter 3, we have proposed the adaptive mobile agent population control algorithms inspired by the single species population model. In this Chapter, we considers the replica density control problem (RDC) in the dynamic networks: the goal of the problem is to adjust the total number of replicas to a constant fraction of the current network size and to adjust the number of replicas of each resource equally. We propose the algorithm for RDC based on the single species population model as with one for MAPC. The difference between MAPC and RDC is that the target objects are static or dynamic. Mobile agents can migrate between nodes and eat food for itself, but replicas can not migrate in the network. In addition, we try to control the densities of multiple resources at the same time in RDC, meanwhile we controlled the agent population of only one kind in MAPC.

This chapter is organized as follows. In Section 4.1, we define RDC. In Sections 4.2, we propose the distributed algorithm for RDC, and show its simulation results. In the first algorithm, we try to control the replica density for a single resource. However, in the system that multiple resources coexist, the first algorithm needs high network cost (i.e., migration cost of mobile agents) and the exact knowledge at each node about all resources existing in the network. Thus, in Section 4.3, we propose the algorithm for RDC that control the densities of multiple resources, and show its simulation results. Section 4.4 concludes this chapter.

4.1 Preliminaries

In this chapter, we consider the *dynamic networks* modeled in Section 3.1.1

4.1.1 Replica Density Control Problem (RDC)

For an application with shared resources, resource replication is a crucial technique for improving system performance. Resources are objects shared by the nodes on the network; files, documents, and so on. Replicas are copies of an original resource. In such systems, generally, a larger number of replicas lead to better performance (e.g., communication latency and consumption of network bandwidth) , but consume more storage of nodes. Thus, it is required to control the number of replicas appropriately for the resource sharing application.

In this chapter, we consider the *replica density control problem* (RDC). Each node has zero or more replicas. We distinguish an original resource from its replicas: an original resource is allocated on its original node and is never deleted unless the original node decides to delete the resource, while a replica of the original resource can be deleted. Each node v can create same replicas from a replica or its original resource on node v and can delete replicas on node v. The goal of RDC is to satisfy two conditions: the first condition is to adapt the total number of replicas to a given constant fraction of the current network size, and the second condition is that all resources have the same number of replicas. Let $r(t)$ be the total number of replicas (including original resources) and $r_i(t)$ be the number of replicas of resource i (including the original resource i) on the network $G(t)$ at time t. The problem is defined as follows.

Definition 2 [Replica Density Control Problem (RDC)] The replica density control problem is solved iff the total number of replicas $r(t)$ and the number of replicas $r_i(t)$ of each resource i at time t satisfy the following equations for a given constant δ.

$$\begin{aligned} r(t) &= \delta \cdot n(t) \\ r_i(t) &= r_j(t) \qquad \forall i, j \end{aligned}$$

It is natural to consider that the system, where the storage cost δ of each node is limited, should keep as many replicas as possible to improve system performance. The first equality represents the adaptation of the current number of replicas to the current network size. That is, the more nodes exist on the network, the more replicas can be kept on the whole network. The second equality represents the restriction on the number of replicas of each resource. That is, when there are many resources on the network, each resource cannot be kept many its replicas since the replicas of the other resources also consume much storage of nodes. On the other hand, when there are not many resources on the network, each resource can be kept many its replicas.

In this chapter, we propose distributed algorithms for RDC. Thus, we assume that the constant δ is initially given to every node. We consider a distributed system such that replicas are distributed over the network and nodes can leave or join the networks at any time. In such

environment, it is obviously impossible to keep satisfying the above equation all the time. Thus, our goal is to propose distributed algorithms that realize quick convergence to the target number and stability at the target number if the network does not change.

4.2 Algorithm for RDC of a Single Resource

In this section, we propose an algorithm for the RDC. This algorithm keeps the replica density of a resource to a given constant fraction of the current network size. We present simulation results to show the proposed algorithm can adjust the replica density.

4.2.1 Algorithm

In this subsection, we present an algorithm for RDC. The algorithm is inspired by the single species population model described in Section 3.2: replicas are regarded as individuals of a single species, and a network is regarded as an environment. That is, replicas need to consume food to survive and the food is supplied by nodes of the network. The food is delivered to replicas by mobile agents.

In the algorithm, we introduce time interval of some constant length denoted by T. Behavior of each node and each replica can be divided into series of the time interval: each node supplies food every the time interval and decides by the food consumption of each replica whether the replica can survive to the next time interval or not. It should be noticed that the start time of the intervals at different nodes need not be synchronized. Moreover, the decisions about survival of different replicas on the same node are made at the different times.

Figure 4.1 and Figure 4.2 show the detailed behavior of nodes and agents in the proposed algorithm for RDC.

The behavior of nodes and agents is simple: each node creates a new agent every T time units (i.e., at the beginning of each time interval). Each agent has a specific amount of food on the initial state and traverses the network with the food. Each agent makes a *random walk* independently: an agent migrates from one node to one of its neighboring nodes with equal probability.When an agent visits node v, node v feeds the replicas on node v with food the agent has. The replica can survive to the next time interval if it can be fed a specific amount of food, denoted by RF, during the current time interval. The replica is deleted if it cannot be fed food of amount RF during the time interval. The original resource behaves the same as a replica except that it is never deleted unless the original node decides to delete the resource: the original resource consumes food just like replicas, but is not deleted even if it cannot be fed food of amount RF. When all the food agent p carries is consumed, the agent p kills itself (i.e.,

Parameters of the algorithm

RF : the amount of food to be consumed by a replica to survive
T : the time interval of behavior of the node and each replica
δ : the target ratio between the number of replicas and the network size

Behavior of node v

Variables of node v

$time_v$: local clock time
/* its value automatically increases at the same rate as the global time */
$eat\_food(h)$: the amount of food that replica h has consumed in the current time interval
$eat\_surplus\_food(h)$: the amount of surplus food that replica h has consumed
$create\_time(h)$: creation time of replica h /* the time at which h is made */

- at the beginning of each time interval of node v (i.e., when $time_v$ **mod** T =0 holds)
  create one agent
- on agent p's arrival at node v
  for each replica h on node v
    if ($food_p > 0$) **then**
      if ($eat\_food(h) < RF$) **then**
        $y := \min\{RF - eat\_food(h), food_p\}$
        $eat\_food(h) := eat\_food(h) + y$
        $food_p := food_p - y$
      else /* the agent is carrying the surplus food */
        $y' := \min\{RF - eat\_surplus\_food(h), surplus\_food_p\}$
        $eat\_surplus\_food(h) := eat\_surplus\_food(h) + y'$
        $surplus\_food_p := surplus\_food_p - y'$
        if ($eat\_surplus\_food(h) = RF$) **then**
          create a new replica of h (let replica j be the created one)
          $create\_time_j := time_v$
          $eat\_food_j := 0$
          $eat\_surplus\_food_j := 0$
          $eat\_surplus\_food(h) := 0$
- at the end of each time interval of each replica h
  (i.e., when $time_v - create\_time(h)$ **mod** T =0 holds)
  if ($eat\_food(h) < RF$) **then** delete h
  else $eat\_food(h) := 0$ /*h survives into the next time interval */

Figure 4.1: Algorithm of node v for RDC

removes itself from the network).

In addition, each node creates a new replica of replica h if the replica h is fed surplus food of amount RF. This idea derives from the fact that the positive per capita growth $g(t)$ in the single species population model is proportional to the amount of surplus food. This scheme is realized in the following way: if all food an agent has is not consumed by replicas after T time units from its creation time, the agent stores the surplus food into variable *surplus_food* and continues a random walk. When an agent carrying surplus food visits node v, node v feeds replicas on node v with the surplus food the agent has. If the total amount of surplus food the replica, say

Behavior of agent p

Variables of agent p

$food_p$: the amount of food p carries /* $food_p > 0$ implies $surplus\_food_p = 0$ */

$surplus\_food_p$: the amount of surplus food p carries

/* $surplus\_food_p > 0$ implies $food_p = 0$ */

$time_p$: local clock time

/* its value automatically increases at the same rate as the global time */

/*p makes a random walk on the network */

- when p is created
  - $food_p := \delta \cdot RF$
  - $surplus\_food_p := 0.0$
  - $time_p := 0$
- at the end of time interval of agent p (i.e., when $time_p = T$ holds)
  - $surplus\_food_p := food_p$
  - $food_p := 0.0$
- when all food is consumed (i.e., when ($food_p = 0.0 \wedge surplus\_food_p = 0.0$) holds)
  - kill itself

Figure 4.2: Algorithm of agent p for RDC

replica h, on node v is fed reaches RF, node v creates one new replica of h. If the agent feeds all the surplus food, it kills itself.

Now, we consider the amount of food F that each agent should supply. Since each node creates one agent every T time units, the amount of food supplied on the whole network at time t can be approximately estimated to be $F \cdot n(t)$. The goal of the replica density control problem is to adjust the total number $r(t)$ of replicas to $\delta \cdot n(t)$. Remind that the single species population model shows that the number of individuals converges and stabilizes at the carrying capacity $f_a(t)/f$. Thus, the algorithm tries to adjust $r(t)$ to $\delta \cdot n(t)$ by adjusting the carrying capacity to $\delta \cdot n(t)$. Since $f_a(t)$ corresponds to the total amount of supplied food on the whole network $n(t) \cdot F$ and f corresponds to the amount of food RF, the following equation should be satisfied:

$$\frac{f_a(t)}{f} = \frac{n(t) \cdot F}{RF} = \delta \cdot n(t).$$

From this equation, the amount of food F each agent should supply is determined to be $F = \delta \cdot RF$.

4.2.2 Simulation Results

In this subsection, we present simulation results to show that the proposed algorithm for RDC can adjust the replica density.

In the simulation, we assume that each agent repeatedly executes the following actions: each

agent stays at a node for one time unit, and then migrates to one of its neighboring nodes by a random walk. We also assume that the migration delay between any pair of neighboring nodes is two time units.

The following values are initialized randomly:

- the initial number and locations of agents
- the initial locations of replicas
- the initial values of the local clocks (i.e., $time_v, time_p (< T)$)
- the creation time of replicas on each node v (i.e., $create\_time(i) (< time_v)$)

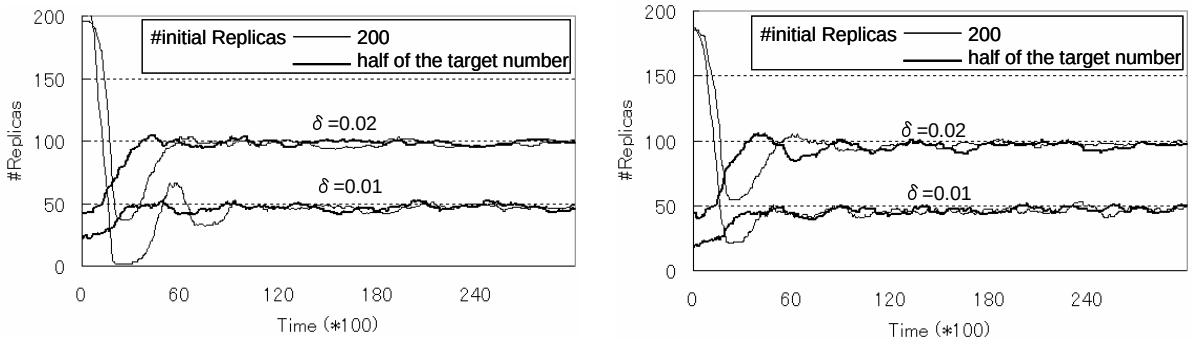
The initial amounts of food that agents have (i.e., $food_p$) and replicas have fed on in the current time interval (i.e., $eat\_food(i)$) are set to the value based on the local clocks and creation times: $food_p = (1 - time_p/T) \cdot \delta RF$ and $eat\_food(i) = (create\_time(i)/T) \cdot RF$. The initial amounts of surplus food that agents have (i.e., $surplus\_food_p$) and the initial amounts of surplus food that replicas have fed (i.e., $eat\_surplus\_food(i)$)are set to 0.

In the simulation, the configuration, after the simulators run for enough time to eliminating the effect of given initial parameters, to converge the replica density to the desired initial number, is consider the initial configuration. In particular, the simulator run with a given initial parameter for 10,000 time units to adjust the number of replicas to the desired initial number $r(0)$, and then, we get experimental data. For example, when we want to set the initial number $r(0)$ of replica to 200 on the network with 5,000 nodes, the value of δ is set to 0.04 for 10,000 time units, and then, the value of δ is set to the desired value. The reason of this is that the appropriate initial configuration, especially the initial number of agents that will depend on the current number of replicas and the amounts of food the replicas has fed, cannot be defined clearly.

In the simulation, a new replica created by a node is allocated to the node selected randomly with probability proportional to their degrees. In real systems, replica allocation is very important to get good performance. However, we focus on control of replica density rather than how to allocate replicas effectively on the network. The above allocation can be realized as follows: a replica traverses the network by a random walk during randomly long time units. Since replicas cannot migrate between nodes, in practice, an easy method to realize the allocation is to make a mobile agent carry the replica.

We present the simulation results for *random networks* and *scale-free networks*. The networks are created by the way presented in Section 3.3.2.

To show the adaptiveness of the proposed algorithm, we also show the difference ratio of the number of replicas: the ratio is defined by $|\delta \cdot n(t) - r(t)|/(\delta \cdot n(t))$ and represents the ratio of



a. random networks ($n(t) = 5000$) b. scale-free networks ($n(t) = 5000$)

Figure 4.3: Simulation results of the RDC algorithm on static networks

difference between the adjusted and the target numbers of replicas to the target number.

Simulation results for static networks

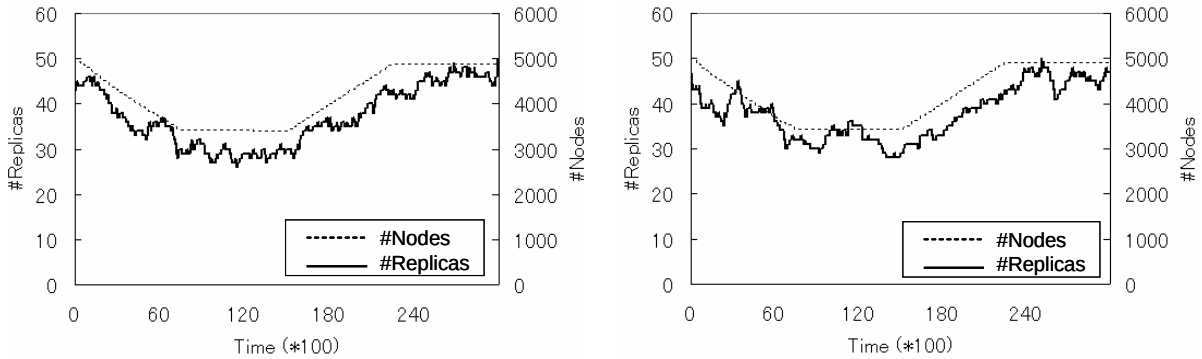
Figure 4.3 shows experimental results for "static" random networks and "static" scale-free networks where nodes and links of the networks remain unchanged. In the simulation, the number $n(t)$ of nodes is fixed at 5,000 during the simulation.

Figure 4.3 shows transition of the number $r(t)$ of replicas with time t. It shows the simulation results for four combinations of two values of δ (0.02 and 0.01), and two initial number $r(0)$ of replicas (200 and the half of the target number). The length T of the time interval is set to 1,000 time units. These simulation results show that the number of replicas quickly converges to the equilibrium point, and has small perturbation after the convergence. In addition, the average of the difference ratio is about 0.06 after convergence of the replica density.

Simulation results for dynamic networks

Figure 4.4 shows the experimental results for "dynamic" random networks and "dynamic" scale-free networks where nodes and links of networks vary with time. When a new node joins in the network, the network is constructed by the same way as that in Section 3.3.2.

Figure 4.4 shows simulation results for dynamic networks : some nodes join in the network and some nodes leave from the network constantly. In this simulation, the initial network size $n(0)$ is 5000, and the following dynamical changes occur every 200 time units. In the first quarter term (from time 0 to time 7,500) of the simulation, a single new node joins in the network with probability 0.1 and each node leaves from the network with probability 0.01 for random networks and with probability $avg\_deg \cdot 0.01/deg_v$ for scale-free networks, that is, the network size is decreasing in this term. In the second quarter term (from time 7,500 to time



a. random networks
$(n(0) = 5000, r(0) = 50, \delta = 0.01)$

b. scale-free networks
$(n(0) = 5000, r(0) = 50, \delta = 0.01)$

Figure 4.4: Simulation results of the RDC algorithm on dynamic networks

15,000) and the forth quarter term (from time 22,500 to time 30,000), a single new node joins in the network with probability 0.2 and each node leaves from the network with probability 0.0001 for random networks and with probability $avg\_deg \cdot 0.0001/deg_v$ for scale-free networks, that is, the network size stays about the same in these terms. In the third quarter term (from time 15,00 to 30,000), 40 nodes joins in the network with probability 1.0 and no node leaves from the network, that is, the network size is increasing in this term.

Figure 4.4 shows transition of the number $r(t)$ of replicas and the network size $n(t)$ with time t. The vertical scale of the network size is marked so that the transition of the network size corresponds to the target number. In the simulation results of Figure 4.4, the length T is set to 1000 time units, the value of δ is set to 0.01 and the initial number $r(0)$ of replicas is set to 50. The simulation results show that the number of replicas is adaptively adjusted in response to changes in the network size. In addition, the average difference ratio is 0.11.

Simulation results on lifetime of replicas

The goal of RDC is to adjust the number of replicas to a given ratio of the network size. However, from the point of application view, locations of each replica should not change frequently. In real applications, if there is almost no change of locations of each replica, the searching performance of applications can be improved. In our algorithm, each replica stays on the same node while the replica exists on the network. Thus, we can say the stability of locations is high by showing that lifetime of replicas is sufficiently long.

Lifetime lt_x of replica x is defined to be the time length from its creation to its elimination, i.e., $lt_x = td_x - tc_x$, where td_x is the time when x is deleted and tc_x is the time when x is created. Table 4.1 shows the average lifetime of replicas of ten trials. To focus on the lifetime of replicas

Table 4.1: Average lifetime of replicas

		T		
		500	1000	1500
n	2000	1447	7825	16123
	5000	1398	7740	17012
	10000	1393	8286	18084

a. random networks

		T		
		500	1000	1500
n	2000	2275	5545	10267
	5000	2143	6202	12830
	10000	2043	6817	12317

b. scale-free networks

after convergence of the number of replicas to the target number, we count lifetimes of only replicas that are deleted the last half of the runtime. In the simulation results of Table 4.1, the value of δ is set to 0.01. The simulation results show that lifetime quickly becomes longer when the length of the time interval T becomes longer. Therefore, by setting an appropriate value to T, it is strongly expected that lifetime of each replica becomes sufficiently long.

Replica density algorithm by using smaller number of agents

In the algorithm for RDC presented in Section 4.2, $n(t)$ agents are created and traverse the network every T time units. Although the size of the agent is so small since the agent has only information of food, the number $n(t)$ of agents may be large for the system. To reduce network traffic, we can modify the algorithm so that each node can create an agent every $c{\cdot}T$ time units for some constant c ($c > 1$). This method reduces network traffic of agents by $1/c$. Since the number of agents on the whole network is roughly reduced to $1/c \cdot n(t)$, each agent has to be created with the initial amount of food $c \cdot \delta \cdot RF$ to keep the total amount of food to be supplied. In this regard, however, the length of the time interval T needs to become longer depending on the value of c. The reason is that agents with larger amount of food must visit more nodes to supply food to more replicas.

We have verified by simulations that the network traffic of agents can be reduced to $1/c \cdot n(t)$ without sacrificing accuracy of the replica density control.

4.3 Algorithm for RDC of Multiple Resources

For the objective of controlling multiple resources that coexist in the system, the straightforward solution is to apply independently the algorithm presented in Section 4.2 to each resource: replicas of resource i are supplied food by agents for i. In this case, the number of agents

existing on the network are also proportional to the number of resources. Moreover, each node needs exact knowledge about all resources existing in the network, since each node must create agents for each resource.

In this section, we propose the algorithm that density of each resource is controlled. The goal of this algorithm is to adapt the total number of replicas of all resources to $\delta \cdot n(t)$ and the number of replicas of each resource equally. This algorithm is based on the algorithm in Section 4.2. In this algorithm, however, only $n(t)$ agents are created every T time units regardless of the number of resources. Each node needs to know only the rough number (instead of the exact number) of resources. We show by the simulation that the algorithm can adjust the density of each resource adaptively.

4.3.1 Algorithm

In this subsection, we present an algorithm that controls the densities of multiple replicas. The algorithm is based on the algorithm proposed in Section 4.2: agents are created by each node every T time units and make random walks to supply food to replicas, and replicas need to consume food to survive. The procedures for each replica is same as those in the first algorithm: the replica can survive to the next time interval if it can be fed food of amount RF during the time interval T, otherwise, the replica is deleted. The algorithm in this section has distinct difference from that in Section 4.2: it requires to balance the replica densities of all resources in addition to adjusting the total number of replicas.

We consider how each agent should supply food to replicas. To adjust the total number of replicas to $\delta \cdot n(t)$, each agent should supply food of amount $\delta \cdot RF$ as the first algorithm. However, since the initial replica density of each resource may be different from that of every other resource, each replica density cannot be adjusted appropriately by the same way as the first algorithm, where agents unconditionally supply food to replicas on the visited node. That is, the replicas in larger number can be fed more food than the replicas in smaller number, and keep the larger number.

To balance the replica densities of all resources, we introduce an additional condition for food supply: each agent tries to supply a same amount of food to replicas of each resource However, each agent cannot know the number of resources that exist on the network. Therefore, in the algorithm, we set an upper limit of the amount of food each agent can supply to a replica based on the amount of food the agent has supplied. Each agent p has information $sf_p(i)$ about the amount of food the agent p supplies to replicas of each resource i. Let SF_p be the maximum amount of food agent p has supplied to replicas of resources, that is, $SF_p = \max\{sf_p(i)\}$. At the agent p arriving at node v, the agent p can supply the replica of resource i the following

amount $F_p(i)$ of food.

$$F_p(i) = \begin{cases} SF_p - sf_p(i) & \textbf{if} SF_p \neq sf_p(i) \\ food_p & \textbf{if} SF_p = sf_p(i) \end{cases}$$

In this way of food supply, it is important for each agent to supply food to replicas of many resources. However, when the value δ is much smaller than the number of resources, each agent can supply food to only replicas of δ resources before it kills itself. In this case, the replicas that exist in large number can keep its large number since many agents can meet the replicas. Therefore, the amount of food each agent can supply at once should be limited to a little amount. The more resources exist in the network, the less amount of food each agent should supply to a replica to guarantee that the agent supplies food to replicas of many resources. On the other hand, the larger amount of food each agent has, the larger amount of food each agent should supply to a replica since each agent has to supply the amount $\delta \cdot RF$ of food during T time units. Thus, in the proposed algorithm, the amount of food each agent supplies to a replica on a visited node is less than $RF \cdot \delta/(Y \cdot kind(t))$, where $kind(t)$ is the number of resources at time t and Y is a positive value.

Based on the above discussion, in the second algorithm, the amount of food each agent p supplies to a replica of resource i on a visited node is represented by

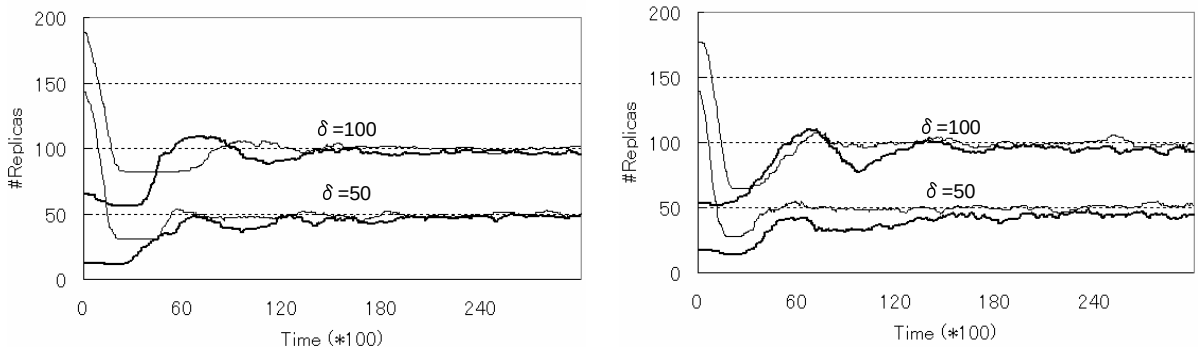
$$\min\{RF - eat\_food(i), food_p, F_p(i), RF \cdot \delta/(Y \cdot kind(t))\}.$$

In the above method, it is necessary for each node to know the number $kind(t)$ of resources at time t. To estimate the number $kind(t)$, each node v has the set rsc_list_v of resource identifiers believed to exist in the network. The node v can obtain the information about existing resources from the agents visiting v, that is, the identifiers of resources the visiting agent has fed food are added to the set rsc_list_v.

Another problem we should consider is delete of resources from the network. We consider dynamic networks that resources may be deleted by their original node. When an original node of a resource i decides deletion of i, the replicas of resource i also need to be deleted from the network. To feed no food to the replicas of the deleted resource, an original node keeps the set of identifiers of resources the node has deleted. Each agent p has the set del_list_p and collects the identifiers of the deleted resources on the visited nodes. If agent p finds the replica of the resource included in the set del_list_p, the replica is deleted from the network.

4.3.2 Simulation Results

Now, we show simulation results of the second algorithm. In the simulation, the behavior and the initial locations of agents are determined in the same way as that in Section 4.2.2. The



a. random networks ($n(t) = 5000$) b. scale-free networks ($n(t) = 5000$)

Figure 4.5: Simulation results of the RDC algorithm of multiple resources on static networks

positive value Y is set to 1.0.

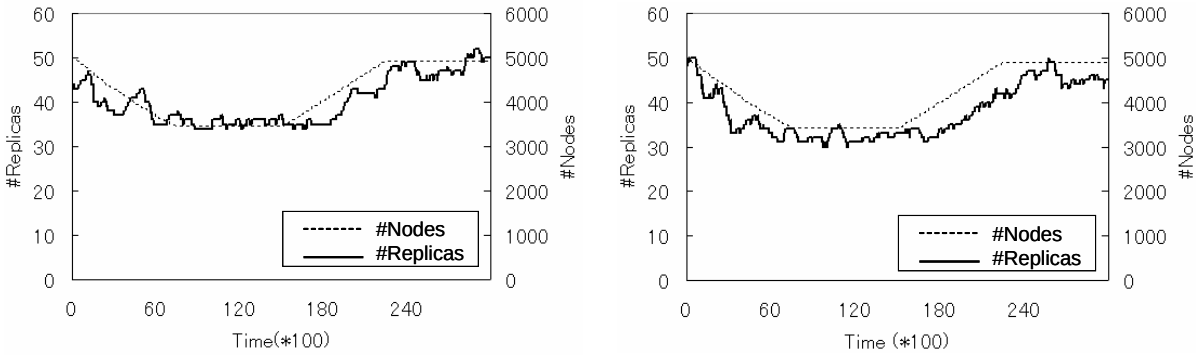
Simulation results for static networks

Figure 4.5 shows the experimental results for "static" networks. The network is created by the way presented in Section 3.3.2. In the simulation, the length T of the time interval is set to 1,000. The network size $n(t)$ is fixed at 5,000 during the simulation. The number of resources is set to 5,000, and the value δ is set to 50 and 100. That is, there are replicas of 5,000 different resources on the network, and the number of replicas of each resources should be kept to 50 and 100 respectively. The initial number of replicas of each resources is set to a random value from 1 to 200. In Figure 4.5, we show the transition of the replica density of two resources chosen randomly from the 5,000 resources on each case of δ is 50 and 100.

The simulation results in Figure 4.5 show that the number of replicas converges to and stabilizes at the target number respectively. The average difference ratio of 5,000 resources at the end of the simulation is less than 0.07.

Simulation results for dynamic networks

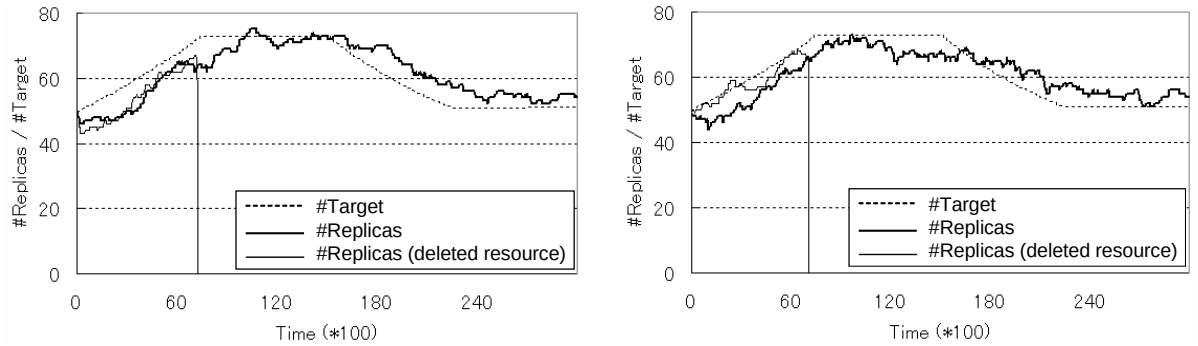
Figure 4.6 shows the simulation results for "dynamic" networks in which the network size $n(t)$ varies with time. The networks change in the same way as the simulation in Section 4.2.2. Note that the number of resources remains unchanged: when the original node v of resource i leaves from the network, node v sends the original resource i to one of its neighbors, and the neighbor becomes the original node of resource i . In the simulation, the length T of the time interval is set to 1,000, the number of resources is set to 5,000, and the value δ is set to 50. Since the network size varies with time, the target number of replicas of each resources also varies with time. The target number of replicas of each resources is 50 at time 0, about 35 at time



a. random networks
$(n(0) = 5000, r(0) = 50, \delta = 50)$

b. scale-free networks
$(n(0) = 5000, r(0) = 50, \delta = 50)$

Figure 4.6: Simulation results of the RDC algorithm of multiple resources on dynamic networks with changes of the network size



a. random networks
$(n(0) = 5000, r(0) = 50, \delta = 50)$

b. scale-free networks
$(n(0) = 5000, r(0) = 50, \delta = 50)$

Figure 4.7: Simulation results of the RDC algorithm of multiple resources on dynamic networks with changes of the number of resources

7,500 and about 48 at the end of the simulation. The initial number of replicas of each resource is set to 50.

In Figure 4.6, we show the transition of the number of replicas of a resource chosen randomly from 5,000 resources. These simulation results also show the replica density is adaptively adjusted in response to changes in the network size.

In the simulations presented on Figure 4.7, the number of resources changes. When the number of resources becomes larger, the target number of replicas of each resource becomes smaller. On the other hand, when the number of resources becomes smaller, the target number becomes larger. In the simulation results of Figure 4.7, the network size $n(t)$ is fixed at 5,000, the length T is set to 1,000 time units, the value of δ is set to 50 and the initial number of

replicas of each resource is set to 50. The initial number of resources is 5,000 (the target number is 50), and about 1,550 resources are deleted from the network at time 10,000 of the simulation (the target number becomes about 73), and about 1,550 resources are added to the network at time 20,000 of the simulation (target number becomes about 50). The deleted resources are chosen randomly.

The simulation results in Figure 4.7 show the adaptation of the replica density to changes in the number of resources. In addition, these results also show that replicas of deleted resources quickly deleted from the network because of no supplied food.

4.4 Concluding Remarks

In this chapter, we have proposed two distributed algorithms for RDC that requires to adapt the total number of replicas to a given constant fraction of the current network size and adapt the number of replicas of each resource equally. The algorithms are inspired by the single species population model. The simulation results show that the proposed algorithms can adequately adjust the number of replicas in dynamic networks. In addition, from the simulation results, the lifetime of each replica becomes sufficiently long by setting an appropriate value to algorithm parameter T.

The first algorithm is very simple and easily understandable. The algorithm can adapt the current density of a single resource to the current network size. For the objective of controlling multiple resources that coexist in the system, the straightforward solution is to apply independently the first algorithm to each resource. In this case, however, the number of agents existing on the network is proportional to the number of resources, and each node needs to know the identifiers of every resources exactly. The second algorithm controls densities of all resources. As a result, the number of agents created periodically is $n(t)$ regardless of the number of resources, where $n(t)$ is the network size at the time t. Moreover, in the second algorithm, each node estimates the number of resources to supply food of amount according to the number of resources, but does not need to have exact knowledge about all resources.

In this chapter, we focus on only the number of replicas. In real systems that provide resource replication, allocation of replicas is also very important. Our future work is to develop the replica allocation algorithm for improving system performance: agents determine allocations of new replicas from network conditions that agents can learn by traversing over the network.

Chapter 5

Mobile Agent Gossip Algorithm

In the mobile agent systems, *gossip* is the most fundamental scheme supporting cooperation among mobile agents. It requires to accomplish all-to-all information exchange over all agents so that each agent can obtain the all information each agent initially has.

In this chapter, we newly introduce the mobile agent gossip problem (MAGP). In this problem, an agent can obtain the information of another agent by meeting the agent itself or the agent that has already got the information. The gossip scheme is expected to accomplish the all-to-all information exchange with a smaller number of agents' moves than the rendezvous algorithms. This chapter is organized as follows. In Section 5.1, we define the system model in this chapter and MAGP. Some of our results in this chapter are based on the investigation of the relation between MAGP and the node leader election (NLEP) in message passing system. In Section 5.2, we investigate this relation; we show the reducibilities between MAGP and NLEP. To prove these reductions, we present an algorithm that simulates a message passing algorithm in a mobile agent system. Section 5.3 presents the upper and lower bounds on the total number of agents' moves for MAGP in several network classes. In these sections, we assume that whiteboards cannot store the information the other agents have to collect. In Section 5.4, we prove that the move complexity for MAGP cannot be improved even if agents are allowed to write the information on whiteboards. Section 5.5 introduce the related works for MAGP, and Section 5.6 concludes the chapter.

5.1 Preliminaries

5.1.1 System Models

In this chapter, we consider the *static network* described in Section 2: the network topology does not change with time.

In addition to the assumption in Section 2, we assume the following in this chapter. The agent in the network is not killed and created from initial configuration. That is, the number of agents denoted by k in the network G does not change. We assume that each agent has a distinct identifier[1]. Let id_i be the identifier of agent p_i (use id_i and p_i interchangeably hereinafter). Each agent does not initially know identifiers of other agents. We also assume that each agent has prior knowledge of neither the number n of nodes nor the number k of agents. Each agent is initially located on any node in G, and more than one agent is not located on the same node. A node on which an agent is initially located is called the *home node* of the agent.

5.1.2 Mobile Agent Gossip Problem

In this subsection, we define the *mobile agent gossip problem* (MAGP). In an initial configuration, each agent p_j has only its own information I_j. The goal of this problem is that every agent collects information of all agents. Hereinafter information means information each agent has to collect. MAGP is defined as follows.

Let $S_j(C_i)$ be a set of the information an agent p_j has in configuration C_i. In initial configuration C_0, each agent p_j has only its own information I_j;

$$S_j(C_0) = \{I_j\}. \tag{5.1}$$

We say that an agent p_j terminates in a configuration C_i iff p_j never executes any operation after C_i. MAGP is solved in configuration C_i iff all k agents terminate and the following condition is satisfied;

$$\forall j (0 \leq j < k)\ S_j(C_i) = \bigcup_{0 \leq l < k} \{I_l\} \tag{5.2}$$

Agents can write only the control data on a whiteboard, e.g., some number of identifiers and counter values. We disallow each agent p_j to leave any information I_j on a whiteboard for two reasons. The first is to avoid introducing huge space for whiteboard. Since we make no assumption on the size of information each agent initially has, each node needs a large memory space to store the information. The second is security reason. It is insecure to write precious information on a whiteboard since any agent can access it. In section 5.4, we prove that the move complexity for MAGP cannot be improved even if agents are allowed to write information on whiteboards.

Instead, we allow agents on the same node to exchange the set of information with each other. When a set P of agents is located on the same node in configuration C_i, then the

[1]In this dissertation, node IDs are not required if agents have unique IDs. Notice that it is not essential whether nodes have unique IDs or not because the node naming is possible by agents' traverse; each node can be labeled with the agent ID and a sequence number in the traverse.

following configuration C_{i+1} satisfies;

$$^{\forall}p_j \in P \; S_j(C_{i+1}) = \bigcup_{p_l \in P} S_l(C_i). \tag{5.3}$$

We define a move as a migration of an agent from one node to its neighbor node. The complexity of MAGP is measured by the total number of moves until all agents terminate in the worst case.

5.2 Relation between MAGP and NLEP

By the fact that MAGP and NLEP can be reduced to each other, we obtain the upper and lower bounds on the number of moves for MAGP. Notice that NLEP is the leader election among "nodes" in message passing systems, instead of agents. We first define the message passing system and NLEP, and present an simulation algorithm of message passing algorithm in mobile agent systems.

5.2.1 Message Passing Systems

The definition of the message passing system in this chapter follows [17].

A network $G = (V, E, \lambda)$ is modeled similarly to mobile agent systems. In a message passing system, each node is a computational entity and communicates to other nodes by exchanging messages. We assume that each node has a distinct identifier. A node v performs a sequence of the following operations;

- *receive*$(m, \lambda_v(l))$: node v receives a message m from its neighbor through a link labeled $\lambda_v(l)$, and performs local computation.
- *send*$(m, \lambda_v(l))$: node v sends a message m to its neighbor through a link labeled $\lambda_v(l)$.

A state of a node is represented by a set of variables of the node. In this chapter, we assume that a node changes its state only when receiving a message; each node executes local computation and sends some messages only when it receives a message from its neighbor. A state transition of node v is denoted by

$$(s_v, (m, \lambda_v(l))) \vdash (s'_v, M'),$$

where M' is a set of messages ($M' = \{(m'_1, \lambda_v(l_1)), (m'_2, \lambda_v(l_2)), \ldots\}$, $0 \leq |M'|$). The above transition indicates the node v with state s_v that has received a message m through the link labeled $\lambda_v(l)$ changes its state to s'_v and sends zero or more messages $m'_1, m'_2, \ldots$ through some

incident links. Exceptionally, *initiator nodes* would start an execution of local computation spontaneously. That is, an initial state transition of initiator node happens without reception of a message. We assume that each state transition is atomic.

A system is said to be asynchronous if communication delay and local computation time are unpredictable but finite. In contrast, a system is synchronous if execution of all nodes is partitioned into rounds; in each round, every node receives all messages sent in the previous round, executes local computation, and sends messages to its neighbors.

An algorithm in the message passing system assumes reliable communication. Reliable communication satisfies the following properties.

(Liveness) Every message sent by a node v to a neighbor node u is eventually received by u.

(Integrity) Every received message was previously sent by a node.

(No Duplicates) No message is received more than once at any node.

5.2.2 Node Leader Election Problem (NLEP)

NLEP is a problem that each node eventually decides a single common leader node. This chapter considers NLEP under the assumption that exactly k nodes (which are chosen arbitrary) are initiator nodes. More precisely, NLEP is specified as follows;

Definition 3 [Node Leader Election Problem (NLEP)] Let $v_0, \ldots, v_{k-1} (1 \leq k < n)$ be initiator nodes in network G. An algorithm is said to solve the leader election problem iff it satisfies the following conditions;

- Exactly one of the nodes is elected as the leader and all the nodes in G know the identifier of the leader node.
- Once a node decides to be the leader, the node never changes its decision.

5.2.3 Reduction of MAGP and NLEP

Now, we prove that NLEP can be reduced to MAGP. In the message passing system, the behavior of agents can be simulated easily in the same number of messages as the agents' moves. Thus the following theorem is obtained both in asynchronous and synchronous systems.

Theorem 1 Suppose that there exists an MAGP algorithm for k agents whose move complexity is m_g, and that the total number of moves required for an agent to travel the whole network is at most m_t. Then, NLEP with k initiator nodes is solved with at most $m_g + m_t$ messages in the message passing system.

Proof We prove this theorem by showing that NLEP is solved by using any MAGP algorithm in the moble agent systems. Each initiator node v_i creates an agent p_i that has v_i's identifier as its initial information I_i. By applying an algorithm for MAGP, the k agents created by k initiator nodes collect all identifiers of the initiator nodes. Each agent elects exactly one common leader agent from the k agents based on the initiator nodes' identifiers. The leader agent, say p_j, travels the whole network so that the node v_j becomes the leader and the other nodes know the identifier of v_j. □

Next, we prove that MAGP can be reduced to NLEP.

Theorem 2 Suppose that there exists an NLEP algorithm for k initiator nodes in an asynchronous message passing system whose move complexity is m_{pl}, and that the total number of moves required for an agent to travel the whole network is at most m_t. Then, MAGP with k agents is solved with at most $2(m_{pl} + m_t)$ moves in the asynchronous mobile agent system.

The proof of Theorem 2 is described below. To prove Theorem 2, we show that MAGP is solved by using an NLEP algorithm. Therefore, we need to simulate a message passing algorithm in the mobile agent system. Several algorithms that simulate a message passing algorithm in agent systems have been proposed [12, 17]. However, these algorithms cannot help the proof of Theorem 2 because they require $O(n)$ agents' moves per message. It assumes in [12] that the number of exchanged messages in any execution of a message passing algorithm is infinite, and in [17] that an agent may crash. Hence, in what follows, we present an efficient simulation algorithm of message passing algorithms that only requires $O(1)$ agents' moves per message. We assume that message passing algorithms terminate within a finite time in asynchronous systems, i.e., the number of exchanged messages in any execution of a message passing algorithm is finite. We also assume that one initiator node corresponds to the home node of an agent.

Simulation of a message passing algorithm

Figure 5.1 shows the algorithm *MSA* that simulates any message passing algorithm in the mobile agent system. To simplify the description of agent's behavior, we use nodes' identifiers instead of links' labels to represent sources or destinations of messages and agent movements. Notice this description is introduced just for simplicity. Actually, the proposed algorithm can be implemented on the system where only local labels of links are equipped.

To simulate a message passing algorithm in the mobile agent system, agents need to transfer messages between two nodes. The idea of simulation is simple; each node v maintains the

Data written on whiteboard at each node v
  A : a message passing algorithm
  s_v : a state of the node v in A
  $S\text{-}queue_v$: a queue of messages to be sent (initially $S\text{-}queue_v = \phi$)
    /* a pair (m, w) in $S\text{-}queue_v$ denotes that message m should be sent to neighbor w */
Variables of an agent p_i
  $visit_i$: a stack containing nodes p_i had passed (initially $visit_i = \phi$)
  msg_i : a memory space to hold contents of a message

• when an agent p_i is initially located on an initiator node v
  initiate the algorithm A
  Process (ϕ)
• on an agent p_i's arrival at a node v from a node u
  if ($msg_i \neq \phi$) **then** /* p_i arrives at v by a delivering move */
    push u on $visit_i$
    Process((msg_i, u))
  else *Send* /* p_i arrives at v by a backtracking move */

Procedure *Process* ((m, u))
  execute local computation $(s_v, (m, u)) \vdash (s'_v, M')$
  write $s_v = s'_v$ and add $\forall(m', u') \in M'$ to $S\text{-}queue_v$
  Send

Procedure *Send*
  if ($S\text{-}queue_v \neq \phi$) **then**
    delete the first entity (m, u) from $S\text{-}queue_v$
    $msg_i = m$
    move to u /* p_i executes a delivering move */
  else *Return*

Procedure *Return*
  if $visit_i = \phi$ **then** terminate
  else
    pop u from $visit_i$
    $msg_i = \phi$
    move to u /* p_i executes a backtracking move */

Figure 5.1: Algorithm *MSA*

messages generated by local computation and their destinations in a send queue $S\text{-}queue_v$ on the whiteboard. Each agent dequeues a message from a send queue on a visited node, moves between nodes with the message, executes local computation, and adds generated messages to a send queue on the visited node. In what follows, we explain the details of agent's behavior in the algorithm *MSA*. On an initiator node u, the corresponding agent p_i initiates a message passing algorithm and executes initial local computation of u. If no message is contained in

S-queue$_u$ then the agent p_i terminates the algorithm. Otherwise, p_i dequeues a message m from *S-queue*$_u$ and moves to m's destination with m (we call the migration with a message *delivering move*). After that, the agent p_i repeats the following actions on a visited node v; p_i pushes the previous node u on its stack *visit*$_i$ in memory space of p_i to return to u later, and executes local computation based on the delivered message m. If there is an entity in *S-queue*$_v$, the agent p_i performs the delivering move from v to the destination of m', where (m', w) is the first message in *S-queue*$_v$. If *S-queue*$_v$ is empty, p_i returns to the node u that is the top entity of *visit*$_i$ (we call the migration without a message *backtracking move*). In this case, the agent p_i that arrived at u does not execute local computation since p_i delivers no message. The agent p_i checks *S-queue*$_u$ and repeats the above actions. When the agent p_i returns to its home node u, p_i terminates the algorithm if *visit*$_i$= ϕ and *S-queue*$_u$ = ϕ hold.

Correctness of the simulation algorithm

Now, we prove the correctness of the algorithm *MSA*. Since agents simulate only transfers of messages in the algorithm *MSA*, the correctness of *MSA* is proved by showing that any execution of *MSA* satisfies the properties of reliable communication.

Theorem 3 Let A be a message passing algorithm that has a finite number of exchanged messages in any execution in an asynchronous system. Any possible execution of the algorithm *MSA* simulates some execution of A.

Proof It is trivial that any execution of *MSA* satisfies the properties of *integrity* and *no duplicates* since a message in *S-queue* is delivered by an agent exactly once.

Now, to prove that an execution of *MSA* satisfies the property of *liveness*, we show that a *S-queue* at every node eventually becomes empty. Suppose for contradiction that a message m in *S-queue*$_v$ is never delivered by agents. Let p_i be the agent that enqueues m. Since *S-queue*$_v \neq \phi$ holds, the agent p_i executes a delivering move from the node v and pushes v on its stack *visit*$_i$. If the agent p_i eventually returns the node v in its stack *visit*$_i$, p_i continues delivering moves from v until *S-queue*$_v$ becomes empty. As a result, the message m is eventually delivered by an agent. Thus, we consider that the agent p_i does not return to the node v in its stack *visit*$_i$. In this case, the agent p_i executes delivering moves infinitely many times on nodes other than v since p_i eventually returns to the node v by backtracking moves if the number of delivering moves is finite. An agent executes a delivering move at a node u only when *S-queue*$_u$ is not empty, and one message is dequeued from *S-queue*$_u$ by a delivering move. Thus, an infinite number of messages is added to queues *S-queue*since the agent p_i executes delivering moves infinitely many times. This implies that an execution of the original algorithm A includes an infinite number

of send events. It contradicts that the number of exchanged messages in any execution of A is finite. Therefore, the agent p_i eventually returns to the node v and the message m is eventually delivered. □

Furthermore, we prove the termination of the algorithm *MSA*; each agent terminates the algorithm only when the agent makes sure that *S-queue* at its home node is empty. From the proof of Theorem 3, it can be shown that each agent eventually returns to its home node and *S-queue* of the home node eventually becomes empty.

Lemma 1 All agents terminate the algorithm *MSA* within a finite time in any possible execution.

Now, we measure the cost of the simulation.

Theorem 4 Suppose the total number of exchanged messages in any execution of a message passing algorithm A is at most M_A. Then, *MSA* simulates the algorithm A with at most $2 \cdot M_A$ moves in the mobile agent system.

Proof The total number of delivering moves is at most M_A since an agent delivers a message when executing a delivering move. The total number of backtracking moves is equal to the number of push operations performed by agents since an agent executes a backtracking move by popping a node from its stack. Each agent pushes a node on its stack only when the agent executes a delivering move. Thus the total number of backtracking moves is also at most M_A. □

From Theorem 4, we can prove Theorem 2.

Proof From Theorem 4, exactly one agent writes "leader" state on a whiteboard at a node by simulating an NLEP algorithm within $2 \cdot m_{pl}$ moves. The agent is elected as the leader and the other agents stop their migration when they terminate the simulated NLEP algorithm. Gossip is accomplished by the leader agent's traverse of the network; the leader agent travels the whole network to collect the information each agent has, and travels again to deliver the information to all agents. Consequently, MAGP can be solved with at most $2(m_{pl} + m_t)$ agents' moves. □

5.3 Algorithms for MAGP

In this section, we present the upper and lower bounds on the total number of moves for MAGP in several network classes. In asynchronous networks, the upper and lower bounds for MAGP

are obtained from Theorem 1, 2 and the upper and lower bounds for NLEP. In synchronous networks, we present a mobile agent algorithm for MAGP.

The following lemma about the lower bound on agents' moves for MAGP clearly holds in arbitrary networks regardless of sense of direction.

Lemma 2 [Lower bound in arbitrary networks] Any algorithm for MAGP requires $\Omega(n)$ moves of agents in any network of size n.

5.3.1 Asynchronous Tree Networks

In [40], an NLEP algorithm in asynchronous non-rooted trees is proposed, and it is proved that its message complexity is $3n + k - 4$, where n and k are the number of nodes and initiator nodes respectively. Since one agent can travel any non-rooted tree with $O(n)$ moves using the depth first search traversal, we obtain the following corollary from Theorem 2.

Corollary 1 [Upper bound in tree networks] MAGP is solved with $O(n)$ moves of agents in any asynchronous non-rooted tree networks of size n.

Lemma 2 directly implies the lower bound for MAGP in rooted and non-rooted tree networks.

Corollary 2 [Lower bound in tree networks] Any algorithm for MAGP requires $\Omega(n)$ moves of agents in any asynchronous tree network of size n.

5.3.2 Asynchronous Complete Networks without Sense of Direction

An NLEP algorithm in asynchronous complete networks without sense of direction is proposed in [33]. It is proved in [33] that the message complexity of the algorithm is $\Omega(n \log n)$. This is the message complexity for an arbitrary number of initiator nodes in the worst case. It can easily shown that the message complexity $O(n \log k + n)$ for k initiator nodes is proved by a proof similar to the one in [33]. An agent can travel the whole network with $O(n)$ moves. Thus, the following corollary is proved from Theorem 2.

Corollary 3 (Upper bound in complete networks without sense of direction) MAGP is solved with $O(n \log k + n)$ moves of agents in any asynchronous complete network without sense of direction of size n, where k is the number of agents.

It is proved in [33] that the message complexity of any NLEP algorithm in complete networks without sense of direction is $\Omega(n \log n)$. Same as the upper bound, we can prove $\Omega(n \log k + n)$ lower bound on messages for NLEP in the case of k initiator nodes. From Theorem 1 and the above fact, we obtain the following Lemma.

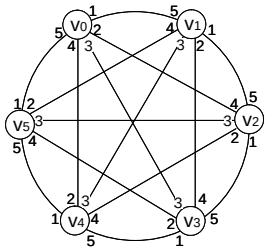


Figure 5.2: A complete network with sense of direction of six nodes

Lemma 3 (Lower bound in complete networks without sense of direction) Any algorithm for MAGP requires $\Omega(n \log k + n)$ moves of agents in any asynchronous complete network without sense of direction of size n, where k is the number of agents.

5.3.3 Asynchronous Complete Networks with Sense of Direction

In this subsection, we show the upper and lower bounds on the total number of moves for MAGP in asynchronous complete networks with sense of direction. The sense of direction is given at each node as follows; nodes are denoted by $v_0, v_1, \cdots, v_{n-1}$, numbered clockwise in a ring, and for every $i, j (0 \leq i, j \leq n-1, i \neq j)$, the link $l_{v_i v_j}$ is labeled by $(j - i) \bmod n$ at v_i. Figure 5.2 shows a complete network with the sense of direction of six nodes.

In [37], an NLEP algorithm in asynchronous complete networks with sense of direction is proposed, and its message complexity is proved to be $O(n)$. Since an agent can travel the whole network with $O(n)$ moves, we obtain the following corollary from Theorem 2.

Corollary 4 (Upper bound in complete networks with sense of direction) MAGP is solved with $O(n)$ moves of agents in any asynchronous complete network with sense of direction of size n.

Lemma 2 directly implies the lower bound.

Corollary 5 (Lower bound in complete networks with sense of direction) Any algorithm for MAGP requires $\Omega(n)$ moves of agents in any asynchronous complete network with sense of direction of size n.

5.3.4 Asynchronous Arbitrary Networks

In this subsection, we show the upper and lower bounds on the total number of moves for MAGP in asynchronous arbitrary networks.

NLEP in a network G can be solved by the Gallager's algorithm for constructing a minimum spanning tree (MST) in G, where no sense of direction is assumed [28]. In Gallager's algorithm, the MST is constructed by merging subtrees based on weights of links[2]. In an initial configuration, each subtree is constructed by a node, and one node that can include all nodes in its subtree becomes the leader. It is proved in [28] that the message complexity of the algorithm is $O(n \log n + e)$, where e is the number of links.

Now, we consider to construct MST with k initiator nodes. To reduce the message complexity for k initiator nodes, each initiator node constructs its first subtree that has no common node with other subtrees and the node that is not an initiator is included in exactly one subtree of an initiator node; an initiator node v sends messages including its identifier $id(v)$ to its all neighbors. The node that received the message including $id(v)$ belongs to v's subtree and sends the received message to all of its neighbors if the node has not received any message, otherwise, the node ignores the message. The construction of first subtrees is required $O(e)$ messages. By addition of the above actions to the Gallager's algorithm, the message complexity for constructing a MST is proved to be $O(n \log k + e)$ in a network with k initiator nodes.

An agent can travel the whole network with $O(n)$ moves by traversing the constructed MST. Thus, from Theorem 2, the following corollary is proved.

Corollary 6 [Upper bound in arbitrary networks] MAGP is solved with $O(n \log k + e)$ moves of agents in any asynchronous network of size n, where k and e are the numbers of agents and links respectively.

It is proved in [28] that the lower bound on messages for NLEP in arbitrary networks is $\Omega(n \log k + e)$ whether the network has a property of sense of direction or not. We can get the following lower bound on the total number of moves for MAGP from the trivial lower bound $\Omega(e)$, Theorem 1, and the traverse cost $O(e)$.

Lemma 4 [Lower bound in arbitrary networks] Any algorithm for MAGP requires $\Omega(n \log k + e)$ moves of agents in any asynchronous network of size n, where k and e are the number of agents and links respectively.

5.3.5 Asynchronous Ring Networks

In this subsection, we show the upper and lower bounds on the total number of moves for MAGP in asynchronous ring networks.

[2]The label $(\min\{id(u), id(v)\}, \max\{id(u), id(v)\})$ is assigned to each link l_{uv} as its weight, where $id(u)$ is the identifier of node u. Two weight labels (u_1, v_1) and (u_2, v_2) are compared with lexicographic order, that is, $(u_1, v_1) < (u_2, v_2) \leftrightarrow u_1 < u_2 \vee (u_1 = u_2 \wedge v_1 < v_2)$.

Corollary 6 directly implies the following corollary.

Corollary 7 [Upper bound in asynchronous ring networks] MAGP is solved with $O(n \log k + n)$ moves of agents in any asynchronous ring network of size n, where k is the number of agents.

As in the case of complete networks without sense of direction, we can get the following lower bound on moves for MAGP from Theorem 1 and the lower bound $\Omega(n \log k + n)$ on messages for NLEP proved in [32]. The proof of the lower bound on messages for NLEP in [32] is guaranteed even if the network has a property of sense of direction (in a ring with sense of direction, each node knows which link is clockwise or counterclockwise).

Lemma 5 [Lower bound in asynchronous ring networks] Any algorithm for MAGP requires $\Omega(n \log k + n)$ moves of agents in any asynchronous ring network of size n, where k is the number of agents.

5.3.6 Synchronous Ring Networks

In this subsection, we present an MAGP algorithm for synchronous rings. In synchronous rings, we cannot apply Theorem 2 since Theorem 2 holds only for asynchronous networks.

In the proposed algorithm, a leader agent is elected from k agents, and the leader agent travels the whole network to collect and deliver information each agent has. The leader agent is elected by similar actions with the NLEP algorithm proposed in [27]. Each agent waits on a node for a time depending on its identifier to reduce the number of moves; each agent p_i waits for $2^{id_i} - 1$ rounds every time p_i arrives at each node. That is, the agent p_i migrates to one of neighbors at most once every 2^{id_i} rounds.

The outline of the leader election among agents is as follows; each agent first writes its identifier on the whiteboard at its home node, and migrates in a direction along the ring. When the agent finds an identifier of other agents, the agent decides, according to the identifier, whether it should continue its migration or not; if the agent has the larger identifier than the written identifier, the agent stops its migration and waits on the node. The only one agent can return to its home node and then it becomes the leader.

The leader agent travels the whole network by migrating in a direction with $O(n)$ moves.

It is proved in [27] that the message complexity for NLEP in synchronous ring networks is $O(n)$. By the same way as [27], we can prove the move complexity of our algorithm.

Theorem 5 [Upper bound in synchronous ring networks] MAGP is solved with $O(n)$ moves of agents in any synchronous ring network of size n.

Lemma 2 directly implies the lower bound on agents' moves in ring networks with and without sense of direction.

Corollary 8 [Lower bound in synchronous ring networks] Any algorithm for MAGP requires $\Omega(n)$ moves of agents in any synchronous ring network of size n.

5.4 Restriction on Usage of Whiteboards

We have discussed MAGP under the assumption that each agent cannot leave information on a whiteboard. In this section, we prove that the move complexity for MAGP cannot be improved even if agents are allowed to write the information on whiteboards.

Theorem 6 Suppose that there exists an MAGP algorithm whose move complexity is m_{gw} under the assumption that each agent can leave any information on whiteboards, and that the total number of moves required for an agent to travel the whole network is at most m_t. Then, MAGP under the assumption that each agent cannot leave information on a whiteboard is solved with at most $m_{gw} + 2 \cdot m_t$ moves.

Proof We prove this theorem by showing that MAGP is solved with leaving no information by using the MAGP algorithm MAG_{wb} in which each agent may leave an information I_j on a whiteboard. Notice that each agent can write some data on a whiteboard except the information. In an execution of MAG_{wb}, whenever each agent p_i leaves an information I_j on a whiteboard, p_i writes the identifiers id_j on the whiteboard, instead of I_j. Similarly, whenever each agent p_i exchanges an information I_j with the other agents, p_i exchanges the identifiers id_j, instead of I_j. As a result, each agent can collect all identifiers of the k agents since the algorithm MAG_{wb} guarantees that each agent collects all information each agent initially has. Thus, each agent can elect exactly one common leader agent from the k agents based on their identifiers. Gossip without leaving information is accomplished by the leader agent's traverse of the network; the leader agent travels the whole network to collect all information each agent has, and travels again to deliver information to all agents. □

5.5 Related Works

Rendezvous is one of approaches to implement the gossip, and has been studied by many researchers [8, 29, 35, 36, 38]. Kranakis et.al. have summarized the recent studies about the rendezvous problem in [35]. Most of these results are classified into two communication models, *whiteboard model* and *token model*. In the whiteboard model, agents on a node can leave

some data on the whiteboard at the node[8], while agents can only put off and pick up some anonymous tokens on nodes in the token model [29, 36]. In this chapter, we assume that agents can communicate with each other using whiteboards; the whiteboard can store information for controlling agents' traversal, but cannot store the information the other agents have to collect. Barriere et.al. have shown in [8] that the computabilities of the rendezvous problem and the agent election problem (i.e., a single agent is elected among agents) are equivalent. It is obvious that MAGP can be solved by any rendezvous algorithm and that the agent election problem can be solved by any MAGP algorithm. Thus, the computabilities of MAGP, the rendezvous problem and the agent election problem are also equivalent. The agent election problem has also been studied in [7, 16, 19] (the whiteboard model is assumed in [16, 19]). In [16, 36], it is indicated that the rendezvous and the election problem can not be solved if agents are anonymous and know neither the network size n nor the number k of agents. Therefore, most of these studies assume that each agent knows the network size n. However, it is unrealistic to assume that each agent initially knows the global information n or k in distributed systems. Thus, in this chapter, we assume that each agent has unique identifier. While most of studies about rendezvous focus on time and memory complexities, we focus on move complexity since migration of agents is said to be costly in mobile agent systems. Several studies also focus on move complexity[16, 38]

The node gossip problem has been extensively investigated[11, 23] ; each node in a network obtains the information each node initially has by message exchanges. However, MAGP has not been considered before. The difference between gossips among nodes and agents is that agents must traverse in a network itself and meet to other agents to collect the information.

5.6 Concluding Remarks

In this chapter, we have considered the mobile agent gossip problem (MAGP). The gossip is a fundamental task in cooperation among mobile agents. The goal of MAGP is that all agents obtain the information each agent initially has with the smallest number of moves. We have shown that MAGP inherently has lower complexity for the total number of moves than the rendezvous problem.

We have investigated the relation between MAGP and the node leader election problem (NLEP); MAGP and NLEP can be reduced to each other. We have shown that the upper and lower bounds on the total number of moves for MAGP have been obtained from those for NLEP, and presented MAGP algorithms which are asymptotically optimal in terms of the total number of moves for several network classes. In all network topologies, the move complexities are

sublinear in k, where k is the number of agents. Especially, for synchronous rings, asynchronous trees, and asynchronous complete networks with sense of direction, their move complexities are independent of k.

Chapter 6

Conclusion

6.1 Summary of the Results

In this dissertation, we have considered the resource control for dynamic and static resources and the gossiping in the mobile agent systems. The resource control that requires to control the number of resources appropriately is a precious technique to guarantee superior performance in the system since the excessive resources increase the system load and the shortage of resources causes system performance decrement. The gossip that achieves all-to-all information exchange among mobile agents is one of the most fundamental communication primitives for cooperation in the mobile agent systems. We have proposed the adaptive resource control algorithms for dynamic networks and the efficient gossip algorithms for static networks in terms of the agents' migration cost.

In Chapter 3, we have tried to control the number of dynamic resources, that is, mobile agents, in the dynamic networks. We have considered the mobile agent population control (MAPC) that requires adapting the number of agents to a given constant fraction of the current number of nodes in the dynamic network. The proposed algorithms are inspired by the single species population model, which is well-known in the field of the population ecology. We have proposed two adaptive distributed algorithms for MAPC. These two algorithms are different in the information each node requires. That is, the first algorithm requires each node to know the numbers of nodes and links. On the other hand, in the second algorithm, each node needs no global information on networks, but locally estimates the link density. The simulation results have shown that the proposed algorithms can adequately adjust the number of agents in dynamic networks and that the second algorithm has slightly lower accuracy than the first algorithm.

In Chapter 4, we have tried to control the number of static resources such as files and data, and have tried to control the numbers of replicas of multiple original resources at the same time,

meanwhile we have controlled the agent population of only one kind in MAPC. Replica density control (RDC) is a problem to adjust the total number of replicas (e.g., copies of original files) to a constant fraction of the current number of nodes and to adjust the number of replicas of each resource equally. The proposed algorithms are based on the same approach as that for MAPC: the algorithms are inspired by the single species population model. The algorithm that controls densities of all resources does not need to have exact knowledge about all resources in the network, and creates only $n(t)$ agents periodically regardless of the number of resources, where $n(t)$ is the network size at the time t. The simulation results show that the proposed algorithms can adequately adjust the number of replicas in dynamic networks.

In Chapter 5, we have newly formulated the mobile agent gossip problem (MAGP): the goal of this problem is that all agents obtain the information each agent initially has with the smallest number of moves. We have shown that MAGP inherently has lower complexity for the total number of agents' moves than the rendezvous problem. We have investigated the relation between MAGP and the node leader election problem (NLEP); MAGP and NLEP can be reduced to each other. We have shown that the upper and lower bounds on the total number of moves for MAGP have been obtained from the upper and lower bounds on the total number of messages for NLEP, and presented MAGP algorithms which are asymptotically optimal in terms of the total number of moves for several network classes. In all network topologies, the move complexities are sublinear in k, where k is the number of agents. Especially, for synchronous rings, asynchronous trees, and asynchronous complete networks with sense of direction, their move complexities are independent of k.

6.2 Future Directions

In Chapter 3 and 4, we have discussed the resource control in which the total numbers of agents and replicas are adapted to a constant fraction of the current network size. Adjustment of the total number of resources proportional to the network size is natural since the more nodes exist in a network, the more processing power and memory space are provided for resources. However, the number of resources regardless of the network size may be desired in some applications. Our future work is to propose an algorithm that adjusts the number of resources to various objective numbers. One example of the objective numbers is a constant number. In this case, we are also able to propose an algorithm based on the single species population model. However, the global information, such as the number of nodes and links, is needed to determine the amount of supply food. Therefore, a different approach from the single species population model should be adopted to the agent system.

Another future work for the resource control is to realize the control of resources only with use of agent-side processes. One of advantages of mobile agent systems is to be able to construct systems only using remote executable programs (or agents). In the proposed algorithms for MAPC and RDC, however, we need actions of environments (e.g., food feeding by nodes) in addition to actions of agents. Therefore, we should consider control algorithms that require no action of network nodes.

In Chapter 5, we have discussed the move optimal all-to-all information exchange. In MAGP algorithms, we use whiteboard for agent to write some control data. Our interest is the minimum memory size to achieve the gossip over agents. Another our interest is a trade-off between the move and the time complexities for all-to-all information exchange; in MAGP, we have focused on the move complexity while the rendezvous problem focuses on the time complexity. Our future work is to analyze the time complexity for the gossip under the optimal moves, and to analyze the move complexity for the gossip under the optimal time.

Acknowledgements

The author has been fortunate to receive assistance from many people. She would especially like to express her gratitude to her supervisor Professor Toshimitsu Masuzawa for his guidance and encouragement. The author has also received precious advice from Professors of the Graduate School of Information Science and Technology, Osaka University. Among them, the author would like to extend her gratitude to Professor Kenichi Hagihara, Professor Shinji Kusumoto and Professor Hirotsugu Kakugawa, for their valuable comments on this dissertation.

The author would like to thank Professor Akinori Saitoh at Tottori University of Environmental Study, Professor Hideo Masuda at Kyoto Institute of Technology, and Dr. Tadashi Araragi at NTT Communication Science Laboratories for their useful comment on her work. The author also thanks to Mrs. Tomoko Arakawa, Ms. Megumi Maki, and Ms. Fusami Nagae for their kind support. She is also grateful to the staffs and students of Algorithm Engineering Laboratory, the Graduate School of Information Science and Technology, Osaka University. In particular, she thanks to Dr. Fukuhito Ooshita, Dr. Taisuke Izumi, and Dr. Yoshihiko Nakaminami for their time and kindness.

Finally, the author wishes to thank her parents Atsuo Suzuki and Yoko Suzuki, and all of her families for their support and kindness during her life at the university.

Bibliography

[1] R. Albert and A. L. Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, January 2002.

[2] K. A. Amin, A. R. Mikler, and P. V. Ivengger. Modeling dynamic agent population in agent-based distance vector routing. *Special issue on Advances in Intelligent Systems in the Journal of Neural Parallel and Scientific Computing*, 11(1&2):127–143, 2003.

[3] The anthill project. http://www.cs.unibo.it/projects/anthill/.

[4] O. Babaoglu, H. Meling, and A. Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. In *Proceedings of the 22th International Conference on Distributed Computing Systems*, pages 15–22, 2002.

[5] A. L. Barabasi and E. Bonabeau. Scale-free networks. *Scientific American*, 288:50–59, May 2003.

[6] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2002)*, pages 200–209, August 2002.

[7] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Can we elect if we cannot compare? In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2003)*, pages 324–332, June 2003.

[8] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems*, 40(2):143–162, April 2007.

[9] The bio-networking architecture. http://netresearch.ics.uci.edu/bionet/.

[10] L. Blin, P. Fraigniaud, N. Nisse, and S. Vial. Distributing chasing of network intruders. In *13th Colloquium on Structural Information and Communication Complexity (SIROCCO 2006)*, pages 70–84, July 2006.

[11] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1143–1156, november 1997.

[12] J. Chalopin, E. Codard, Y. Metivier, and R. Ossamy. Mobile agent algorithms versus message passing algorithms. In *10th International Conference On Principles Of Distributed Systems (OPODIS 2006)*, volume 4305 of *Lecture notes in computer science*, pages 187–201, December 2006.

[13] R. R. Choudhury, K. Paul, and S. Bandyopadhyay. Marp: A multi-agent routing protocol for mobile wireless ad hoc networks. *Autonomous Agents and Multi-Agent System*, 8(1):47–68, January 2004.

[14] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[15] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in tree networks. In *8th International Conference On Principles Of Distributed Systems (OPODIS 2004)*, pages 67–80, December 2004.

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

[17] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. Fault-tolerant simulation of message-passing algorithms by mobile agents. In *Proceedings of the 14rd Colloquium on Structural Information and Communication Complexity (SIROCCO 2007)*, pages 289–303, June 2007.

[18] A. Dessmark, P. Fraigniaud, and A. Pelc. Deterministic rendezvous in graphs. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003)*, pages 184–195, September 2003.

[19] D. Deugo. Mobile agents for electing a leader. In *Proceedings of the 4th International Symposium on Autonomous Decentralized Systems (ISADS 1999)*, pages 324–327, March 1999.

[20] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocols. *Distributed Computing*, 19(1):1–18, 2006.

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

[22] Y. Drougas and V. Kalogeraki. A fair resource allocation algorithm for peer-to-peer overlays. In *Proceedings of the 24th Conference on Computer Communications*, pages 2853– 2858, March 2005.

[23] M. Flammini and S. Perennes. On the optimality of general lower bounds for broadcasting and gossiping. *SIAM Journal on Discrete Mathematics*, 14(2):267–282, 2001.

[24] P. Flocchini, M. J. Huang, and F. L. Luccio. Contiguous search in the hypercube for capturing an intruder. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, page 62, April 2005.

[25] P. Flocchini, M. J. Huang, and F. L. Luccio. Decontamination of chordal rings and tori. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, page 8, April 2006.

[26] P. Flocchini, F. L. Luccio, and L. X. Song. Size optimal strategies for capturing an intruder in mesh networks. In *the 2005 International Conference on Communications in Computing (CIC 2005)*, pages 200–206, September 2005.

[27] G. N. Frederickson and N. Lynch. Electing a leader in a synchronous ring. *Journal of the ACM*, 31(1):98–115, January 1987.

[28] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning tree. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, january 1983.

[29] L. Gasieniec, E. Kranakis, D. Krizanc, and X. Zhang. Optimal memory rendezvous of anonymous mobile agents in a uni-directional ring. In *Proceedings of the 32nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2006)*, pages 282–292, January 2006.

[30] Gnutella.com. http://www.gnutella.com.

[31] R. Haberman. *Mathematical Model : Population Dynamics*. PRENTICE HALL, 1977.

[32] D. S. Hirschberg and J. B. Sinclair. Decentralized extrema-finding in circular configurations of processors. *Communications of the ACM*, 23(11):627–628, November 1980.

[33] E. Korach, S. Moran, and S. Zaks. Optimal lower bounds for some distributed algorithms for a complete network of processors. *Theoretical Computer Science*, 64(1):125–132, April 1989.

[34] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Placement algorithms for hierarchical cooperative caching. In *Proceedings of the 10th annual ACM-SIAM symposium on Discrete algorithms*, pages 586– 595, January 1999.

[35] E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. In *Proceedings of the 13rd Colloquium on Structural Information and Communication Complexity (SIROCCO 2006)*, pages 1–9, July 2006.

[36] E. Kranakis, N. Santoro, C. Sawchuk, and D. Krizanc. Mobile agent rendezvous in a ring. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)*, pages 592–599, May 2003.

[37] M. C. Loui, T. A. Matsushita, and D. B. West. Election in complete networks with sense of direction. *Information Processing Letters*, 22(4):185–187, April 1986.

[38] G. D. Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, April 2006.

[39] K. Miura, T. Tagawa, and H. Kakugawa. A quorum-based protocol for searching objects in peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(1):25–37, January 2006.

[40] N. Santoro. *Design and Analysis of Distributed Algorithms.* Wiley Interscience, 2007.

[41] R. Stephan, P. Ray, and N. Paramesh. Network management platform based on mobile agents. *International Journal of Network Management*, 14(1):59–73, 2004.

[42] J. Suzuki and T. Suda. Design and implementation of a scalable infrastructure for autonomous adaptive agents. In *Proceedings of the 15th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 594–603, November 2003.

[43] G. Tel. *Introduction to Distributed Algorithms (second edition).* Cambridge University Press, 2000.

[44] C. D. Timon, Y. L. Eldon, and C. An-Pin. Mobile agents in distributed network management. *Communications of the ACM*, 46(7):127–132, July 2003.