A Study on Protocol Design and Combinatorial Optimization in Resource Assignment Type Problems on the Basis of Net Theory and Genetic Algorithms

(資源割当型問題に対するネット理論と遺伝的アルゴリズムに基づく
プロトコル設計と組合せ最適化に関する研究)

Morikazu NAKAMURA

A dissertation
submitted to the Graduate School of Engineering of Osaka University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering

October 1995

ABSTRACT

Resource assignment type problems constitute the classical, essential core to computer sciences, and their solutions can be applied to a wide range of industrial situations. With recent advances in the computer technology, such as the appearance of high performance computers, the evolution of the computing environment from "centralized" to "distributed", the emergence of new methodologies like genetic algorithms (abbreviated as GA), etc., the quality as well as the quantity of research topics associated with these problems has been expanded.

This dissertation considers three kinds of resource assignment type problems: the mutual exclusion problem in distributed environments, the stable marriage problem, and the multiprocessor scheduling problem, and describes new results for these problems by means of the net theory (a combination of graph theory and Petri nets) and genetic algorithm techniques.

In Chapter 1, backgrounds and objects of this research work are outlined, stressing mainly on three primary subjects; the mutual exclusion problem in distributed environments, the stable marriage problem, and the multiprocessor scheduling problem.

Chapter 2 explains collectively fundamental concepts and notations, to be used in this dissertation; graphs and Petri nets, genetic algorithms, lattice structure, and NP-completeness.

In Chapter 3, a solution is proposed for protocol design of the mutual exclusion problem in distributed environments. The proposed protocol realizes mutual exclusion for the network resources by applying virtually a dynamic sequence of acyclic graphs, called an acyclic graph evolution, to the distributed environment. A topological condition and algorithmic schemes are proposed for determining the initial acyclic directioning, which is to ensure exactly one source node at any stage of the acyclic graph evolution.

Chapter 4 extends the proposed protocol in Chapter 3 to the case of multiple resources. A sufficient topological condition and algorithmic schemes are proposed for determining the initial acyclic directioning, which is to ensure less than k source nodes at any stage of the acyclic graph evolution, as an extension to the case of multiple resources. The validity and the properties of the protocol are analyzed by using graphs and Petri nets

Chapter 5 formulates a gender-fair matching in the stable marriage problem, and discusses a solution scheme by using a genetic algorithm.

Gale and Shapley originally proposed the stable marriage in which two groups to be matched are represented by a men group and a women group. Real applications range over matching software processes to CPUs, matching autonomous robots to battery charger devices, and matching members of two sets of autonomous robots to each other for pairing up in a cooperative work. The gender-fair stable marriage problem is one of the important open problems among stable marriage problems. The main result consists in a procedure constructed for applying a genetic algorithm (GA) to the gender-fair problem, in which the gender-fair marriage problem is transformed into a graph theoretic problem. This graph theoretic representation is more amenable to the application of GA, since it admits easier coding, and more efficient definition of genetic operators. Computer experiments confirm the effectiveness of the GA solution.

Chapter 6 proposes a genetized-knowledge genetic algorithm (abbreviated as gkGA), and discusses its application to multiprocessor scheduling problems. In this approach, the heuristic itself is represented by genes, and by means of GA selection superior heuristics survive for a given problem, while others die out during the selection process. Moreover, by the crossover of heuristic genes themselves, hybridized heuristics can also be generated. This is a novel strategy in the GA field of combinatorial problems; although the idea of reinforced GAs was already proposed, heuristics have not yet been explicitly genetized.

The effectiveness of our proposed strategy for multiprocessor scheduling problems is proved through computer evaluation.

Chapter 7 summarizes the main results of this dissertation and discusses future research topics.

TABLE OF CONTENTS

Chapter 1

Introduction

In this dissertation, we consider three kinds of resource assignment type problems, namely the mutual exclusion problem in distributed environments, the stable marriage problem, and the multiprocessor scheduling problem. These problems are classic ones, but constitute the essential core of the computer science [1-3]. New solutions are proposed based on net theory (graphs and Petri nets), and genetic algorithm techniques. In this chapter, we explain the background and summary of results for each problem, and outline the overall organization of this dissertation.

1.1 Mutual Exclusion Problem

First, we consider the mutual exclusion problem in a distributed environment and give a net-based solution of protocol design. This solution provides a communication structure and a protocol on the basis of analysis of graph theory and Petri nets. That is, the protocol realizes mutual exclusion for the network resources by virtual application of a dynamic sequence of acyclic graphs, called an acyclic graph evolution, to the distributed environment. The validity and properties of the protocol are also analyzed by using graphs and Petri nets.

The mutual exclusion problem is basic in computer operating systems [4] and has been extensively studied by many researchers. We treat first the mutual exclusion problem for a distributed network of autonomous nodes for the ordinary case of a single-kind/single-quantity network resource, and then extend to the case of multiple kinds of shared resources. The former has been treated by many researchers [5-12] but the latter by only a few.

There exist some distributed mutual exclusion protocols for the single shared resource case as follows:

(a) A logical ring of the nodes is constructed on a given network. The privilege of mutual exclusion is represented by a special message, called

a token, which is transferred from one node to another around the ring [5].

(b) A node requiring to enter the critical section sends a request message containing a time stamp to all other nodes. If more nodes than one request the entry, the node sending the oldest time stamp acquires the privilege [6-8].

(c) Let $C=\{Q_1,Q_2,...,Q_k\}$ be a collection of node subsets such that $Q_i \cap Q_j \neq \phi$, $i \neq j$, where C is called a coterie and Q_i a quorum. The node waiting to enter the critical section chooses one quorum from the coterie and collects the consensus of the nodes in the quorum. There are many interesting studies of the composition of a coterie [9,10] and relevant protocol design [10-11].

The protocol proposed in this dissertation uses the idea of a dynamic sequence of acyclic graphs, called acyclic graph evolution, devised by Chandy & Misra [13]. Using graph and Petri net theory, we extend their idea and propose a protocol which is suitable for distributed autonomous environments and applicable to the multiple shared resources case. First, let us explain the details of the Chandy-Misra approach and then describe the properties of our protocol.

Chandy and Misra proposed a mutual exclusion protocol for a single shared resource in a distributed network. Before the protocol starts, acyclicity with a single source and single sink is virtually introduced, on the bilateral communication graph $\boldsymbol{G}$ that is tacitly assumed completely connected with logical edges not necessarily physical, by communicating the existence of two special tokens (messages), one fork-token and one request-token placed at the terminal nodes of each edge. That is, the tokens are distributed over the node set of the communication graph $\boldsymbol{G}$ such that a directed graph $\vec{\boldsymbol{G}}$ obtained from $\boldsymbol{G}$ by replacing each edge by diedge (directed edge) from the fork to the request is acyclic.

Henceforth, the start-node of each diedge is understood to possess the higher priority for the shared resource usage than its end-node. Since the edge direction is acyclic globally, a total order relation is induced on the graph, and so only a single node becomes the source node. From this singleness, the privilege for using the single shared resource can be

given to the source node. Just after releasing the resource after using it, message exchange taking place between the source node and its neighbor nodes triggers reversal of the directions of all the diedges incident to the source node. We use a neutral term "to fire a source node" instead of "to use the resource and release it". By the edge reversal, the source node will turn into a sink node and a new source node will emerge (note that only a single source node should appear). This process of source firing is to be iterated.

The protocol ensures mutual exclusion with fairness and liveness. That is, there exists only one source node at any instant of the acyclic graph evolution in which every node become firable in turn (fairness) and deadlock cannot occur (liveness). We must note that in the Chandy-Misra model a given graph is assumed completely connected and so any of its acyclic directioning produces always a single source node. However, in the distributed environment, this is not so, as each node is not necessarily connected to all the others.

We extend the Chandy-Misra model to allow the graph to be only partially connected. In general an acyclic directioning of Chandy-Misra type on a partially connected graph, induces just a partial order relation. In the partial order relation, plural source nodes may appear at some stage of an acyclic graph sequence. We therefore introduce a condition which ensures the existence of only a single source on the acyclic graph evolution for the single shared resource. We then proceed to utilize the natural initial plurality of source nodes, and extend the protocol to cover the problem of mutual exclusion in the case of multiple shared resources.

In Chapter 3, we describe the complete mutual exclusion protocol, called Distributed MUTEX, present a topological condition and algorithmic schemes to determine the initial acyclic directioning which will ensure exactly one source node at any stage of the acyclic graph evolution.

In Chapter 4, we go on to present a sufficient topological condition and algorithmic schemes to determine the initial acyclic directioning which will ensure less than k source nodes at any stage of the acyclic

graph evolution, which means an extension to the case of multiple resources.

1.2 Gender-fair Stable Marriage Problem

We next consider the stable marriage problem. This is another resource assignment type problem, originally proposed in a seminal paper by Gale and Shapley [17]. The basic problem is explained as follows:

We consider two sets of equal size n, the men group and the women group,

$$\boldsymbol{m} = \{m_1, m_2, \ldots, m_n\}$$
$$\boldsymbol{w} = \{w_1, w_2, \ldots, w_n\}.$$

Each member in both sets possesses a strictly ordered preference list PL, containing all the members of the opposite sex. Let p be a person (man or woman) and q, r be members of the opposite sex to p. A person p prefers q to r if q precedes r on p's preference list. Two pairs (m_i, w_i) and (m_j, w_j) are said to be "unstable" if conditions

m_i prefers w_j to w_i, and

w_j prefers m_i to m_j

hold. When unstable, there is a high probability of break-up and of a new pair formation (m_i, w_j). A matching $\boldsymbol{M} = \{(m_1, w_1), (m_2, w_2), \ldots, (m_n, w_n)\}$ is considered "stable" if no two pairs in $\boldsymbol{M}$ are unstable.

Real applications range over matching software processes to CPUs, matching autonomous robots to battery charger devices, and matching members of two sets of autonomous robots to each other for pairing up in a cooperative work. In each case preference lists may be constructed according to environment parameters such as cost, distance, capacity, and efficiency. The stability is evaluated by monitoring various properties of the environment. For example, let us consider a matching between two kinds of autonomous robots in which they want to find the nearest partner to them. The preference list is then constructed according to the distance between possible partners. The stability means that after determining the set of matching partners, for any robot p, there is no robot q nearer to p than p's current partner to which p is nearer than q's current partner. Therefore, no robot tries to approach a robot which is not its partner.

Gale and Shapley proposed an algorithm to find a stable matching in their paper [17]. The stable matching found by their algorithm is extremal among the many possible stable matchings in that each member of the men group gets the best partner over all stable matchings, and then each woman gets the worst partner. We say Gale-Shapley algorithms can search for the man-optimal (or woman-pessimal) stable matching. Equally, by exchanging the roles, it produces the woman-optimal (or man-pessimal) stable matching. Finding a fair stable matching (fair to both genders) is an interesting problem, listed in [18,19] as one of the important open problems related to the stable marriage problem, and there are no known polynomial time algorithms for it.

If we are searching for a stable matching with a particular property, like gender-fairness, then a brute force algorithm that examines all stable matchings requires exponentially increasing time in the worst case, since the number of stable matchings can grow exponentially with the size of the groups to be matched [18]. In [20] an algorithm is proposed to find an "egalitarian" stable matching which minimizes the sum of the assigned partner's preference rank (or position) in each person's PL. However, it is still possible for one sex to fare much better than the other in the egalitarian stable matching.

We define the gender-fair stable marriage problem, an open problem [18,19], and describe a genetic algorithm solution to it. The problem of obtaining the gender-fair matching is again difficult by brute computation since we have to select certain matchings from among an exponentially increasing number of possible matchings. The use of genetic algorithms (abbreviated as GA hence after) has drawn attention recently as a new methodology for combinatorial optimization [14-16]. We propose a method for adapting GA to solve this particular problem. The main result consists in a procedure constructed for applying a genetic algorithm (GA) to the gender-fair problem, in which the gender-fair marriage problem is transformed into a graph theoretic problem. This graph theoretic representation is more amenable to the application of GA, since it admits easier coding, and more efficient definition of genetic operators.

In Chapter 5, we define and discuss a gender-fair stable matching that possesses the fairness property, that is, the sum of the male scores (rank of the partner in PL) is as close as possible to the sum of the female scores, and present a method for adapting GA to the gender-fair stable marriage problem. The performance evaluation by computer experiment shows that for randomly generated marriage instances, the proposed method produces the best fair solutions with high probability.

1.3 Multiprocessor Scheduling Problem

Lastly we consider the multiprocessor scheduling problem, again a resource assignment problem of classical reputation. The aim is to find the optimum schedule for assigning software tasks to individual processors. It is a classical problem and has been treated by many researchers [3]. We propose here a new methodology to solve the problem using genetic algorithms (GA).

Various researchers [14-16,21] have pointed out that genetic algorithms are good at global searches but not at local ones, and have then proposed a reinforcement of genetic algorithms with a heuristic technique tuned for local search effectiveness (so-called hybrid GAs in [14-16]). In a reinforced GA, a prescribed domain knowledge is employed to produce an efficient string (or a chromosome; a candidate of the solution, is represented by a string in genetic algorithms) that leads to better solutions than those obtained by simple GA. In this dissertation we develop a novel GA in which domain knowledge is genetized (represented as genes) and is included in a string (a chromosome). A given set of heuristic search techniques constitutes a gene pool of knowledge, and crossover of knowledge genes A and B generates a hybrid gene AB. The new method is called "genetized-knowledge GA," abbreviated as gkGA.

The multiprocessor scheduling problem is known to be NP-hard, even when various restrictions are added to simplify the problem [22-24]. Therefore many researchers have tried to develop approximation algorithms in preference to absolute optimization [25-27]. Over many years of research, most effective solutions found have been based on the list scheduling scheme [22,27]. In this scheme the quality of a schedule is

known to depend strongly on a prescribed priority list of tasks; several methods for organizing these priority lists have been proposed and are summarized in [22]. The critical path (CP) method organizes a priority list such that tasks are arranged in a sequence according to their time depth [22]. Kasahara [27] proposed a modified CP method, called CP/MISF(Critical Path / Most Immediate Successors First), in which a task having the most number of immediate successors has higher priority over multiple tasks possessing the same time depth. Nakasumi [28] tried ordering tasks of the same time depth by appealing to genetic algorithms. These modified CP algorithms were reported to produce good quality schedules for a wide range of task graphs [22,27,28]. However, it was revealed that for certain kinds of task graphs, even these modifications were led to local optimum schedules.

In Chapter 6, we propose the gkGA. The search heuristic is expressed as a gene in the string, and a hybrid gene, a product of the crossover process, offers a new heuristic that may prove to be more efficient for a given task graph. This is a novel strategy in GA research; although reinforced GAs are treated in the literature [14-16], but the heuristic is not genetized. By computer experiment, we prove that the proposed gkGA is capable of generating uniformly better schedules over a variety of task graph instances compared with the CP/MISF and Nakasumi's GA although at the expenditure of more computation time.

1.4 Organization

The organization of this dissertation is as follows: In Chapter 2, we describe the primary concepts which form the base of the following chapters. In Chapter 3, we describe a net-based mutual exclusion protocol for the single resource case. In Chapter 4, we analyze the concurrency of the network protocol proposed in Chapter 3 and extend the protocol to the multiple shared resource case. In Chapter 5, we define a gender-fair stable matching and propose a method to adapt GA. In Chapter 6, we propose a novel strategy for scheduling problems, named genetized-knowledge GA. We finish in Chapter 7 with some concluding remarks.

Chapter 2

Fundamental Concepts and Notations

In this chapter, we explain collectively fundamental concepts and notations, to be used in this dissertation; graphs and Petri nets, genetic algorithms, lattice structure, and NP-completeness, for improving the readability of this dissertation. See references on each topic for more information.

2.1 Net Theory: Graphs and Petri nets [29-31]

We use the term "net theory" in a sense of a combination of graph theory and Petri net theory.

A (undirected) *graph* is denoted by $G(V,E)$, where V and E are the sets of *nodes* and *edges*; if edge e is incident to a node u and v, then it is represented by the unordered pair of nodes $<u,v>$. A directed graph (*digraph*) $\vec{G}(V,\vec{E})$ is a structure that consists of the sets of nodes V and directed edges (*diedges*) $\vec{E}$. We denote a diedge e directed from u to v by the ordered pair $e=(u,v)$. The graph $G(V,E)$ generated by replacing each diedge (u,v) with undirected edge $<u,v>$ in a digraph $\vec{G}(V,\vec{E})$ is said the *underlying graph* of $\vec{G}(V,\vec{E})$.

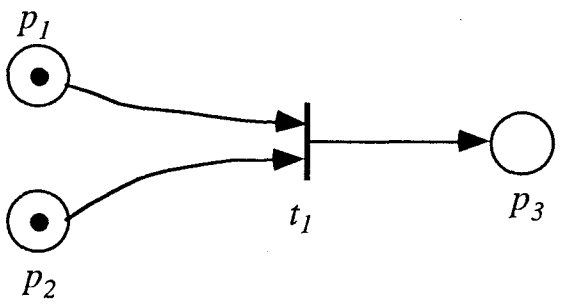
A *circuit* of length $k(\geq 2)$ in a graph G, $c=(V_c,E_c)$, $V_c \subseteq V$, $E_c \subseteq E$, $|V_c|=|E_c|=k$, is a subgraph of G such that there is a path on c between any two nodes in V_c, and, for each node v_i in V_c, the number of its incident edges in c(but not in G) exactly equals 2. Let c be a circuit in the underlying graph of a digraph $\vec{G}$, and let $\vec{c}$ be a subgraph of $\vec{G}$ corresponding to c. If each diedge on $\vec{c}$ is directed in the same orientation, the $\vec{c}$ is called a cycle of $\vec{G}$.

A set of circuits $\Omega=\{c_1,c_2,\cdots c_k\}$, is said a *circuit-cover* of G if any node v of G is contained in at least one circuit from Ω but no proper subset of Ω has this property. A graph G is said *connected* when there exists at least one path between any two nodes in G. A connected graph G is said *circuit-connected* if every node in G belongs to some circuit.

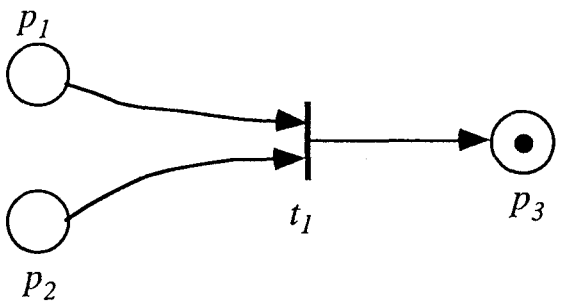
When a directed graph does not contain any cycle, it is called *acyclic* or acyclicly directed. Let $*u = \{w|(w,u) \in \vec{E}\}$, $u* = \{v|(u,v) \in \vec{E}\}$, for each $u \in V$. If $|*u| = 0$ (or $|u*| = 0$), then u is called a *source* (or a *sink*) of $\vec{G}$, where $|*u|$ denotes the number of elements in set $*u$.

A Petri net is a graphical and mathematical modeling tool. Many concurrent / asynchronous / distributed / nondeterministic systems have been effectively modeled and analyzed by this tool. Structures and dynamic behaviors of systems can be expressed graphically in the form of the token distribution. A Petri net is also amenable to mathematical analysis, since they are algebraically representable which allow numerical as well as symbolic manipulations.

A Petri net is a directed bipartite graph, with *initial marking*, M_0. The net consists of two kinds of nodes, called *places* and *transitions*, in which *arcs* either from a place to a transition or from a transition to a place. Fig. 2.1(a) shows an example of a Petri net. Places are drawn as circles, transitions as bars or boxes.



(a) Before firing



(b) After firing

Fig. 2.1 Example of Petri nets

The behavior of a system is simulated by tracing changes of markings in the Petri net model. A marking of a Petri net changes itself in accord with the following firing rule:

(1) A transition t is said to be *enabled* if each input place p of t is marked.

(2) An enabled transition may or may not *fire*.

(3) A firing of an enabled transition t removes one token from each input place p of t, and adds one token to each output place q of t.

In Fig. 2.1(a), transition t_1 is enabled, and Fig. 2.1(b) shows the state or marking after firing t_1.

2.2 Genetic Algorithms [14-16]

Genetic algorithms (abbreviated as GA) were first introduced by Holland's paper "Adaptation in natural and Artificial Systems" in 1975, and can be regarded as a scheme for probabilistic searching, learning, and optimization. Recently, many researchers reported efficiency and robustness of GAs when applied to the combinatorial optimization problems [14-16].

A solution candidate of a problem is represented by a *string* (or a *chromosome*). A string consists of a sequence of *gene*s. We have to find methods of encoding, that is, how to represent a solution candidate by a string, in which each gene is carefully structured to reflect problem semantics. Performance of a string to the environment is evaluated by a *fitness function*. The return value of the fitness function is called a fitness value, which expresses a degree of the string or solution candidate's adaptability to a given environment. Lager the value means more fitted to the environment. In GA, maintenance of he gene variety is vitally important because genetic operations are repeatedly applied onto a pool of strings, in order to generate population of gene varieties from which the fittest are selected to survive. The number of the strings in a population is said *population size*.

A genetic algorithm is a Generate-and-Test type algorithm and generally composed of three genetic operations, *selection*, *crossover*, and *mutation*: Selection determines which strings to survive in the string pool depending on their fitness values. The roulette selection is a

standard in which strings with lager fitness value are selected for survival with higher probability. Crossover combines the genes of two parent strings to generate two offsprings by swapping corresponding parts of the parent genes. There are several ways of performing crossovers, one-point crossover, two-point crossover, OX-method, and so on. Mutation arbitrarily changes one or more genes of a selected string randomly with a probability equal to the mutation rate.

A GA is usually explained as follows:

1° Generate the initial population of strings
2° Evaluate each string by the fitness function
3° Select which strings to survive
4° Perform Crossover operations onto the string pool
5° Mutation
6° goto Step 2 unless the condition for the termination hold.

Usually, the initial population at Step 1 is randomly generated. Each iteration is called *generation* in a GA execution. The fitness value of strings in the population will grow lager at later generations. Finally, a string that has the largest fitness value in the population of the last generation is the best candidate among those searched in the GA execution.

2.3 Miscellaneous [18,23]

[Distributed Lattice] A *distributed lattice* is a partial order in which
(1) each pair of elements a, b has a greatest lower bound, or *meet*, denoted by $a \wedge b$, so that $a \wedge b \preceq a$, $a \wedge b \preceq b$, and there is no element c such that $c \preceq a$, $c \preceq b$ and $a \wedge b \prec c$.
(2) each pair of elements a, b has a least upper bound, or *join*, denoted by $a \vee b$, so that $a \preceq a \vee b$, $b \preceq a \vee b$, and there is no element c such that $a \preceq$, $b \preceq c$ and $c \prec a \vee b$.
(3) the distributed law holds, that is, $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ and $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$.

[Proof Technique of NP-completeness] To prove that a problem P is NP-complete, we merely show that
(1) P$\in$NP,

(2) some NP-complete problem P' can be reduced to P in polynomial time.

Chapter 3

A Topological Design of Mutual Exclusion Protocol for Single Shared Resource in Distributed Environments of Autonomous Nodes

In this chapter we treat a mutual exclusion problem, which is formulated for a single shared resource in distributed environments of autonomous nodes. The most important property of the network treated in this chapter is its membership variability, that is, frequent occurrence of entries of new nodes and exits of old nodes. Thus, when the network is of large-scale, keeping up the information of all other nodes is not possible for each node. Therefore, distributed protocols in such environments should be established by only local message communication but not global one. We in this chapter design a mutual exclusion protocol for distributed environments of autonomous nodes, which is established by message communication between neighbors. The proposed protocol, called Distributed MUTEX, is based on Chandy-Misra protocol for Dining Philosopher(diners) problem. We consider requirements of the communication topology that makes mutual exclusion possible, and propose entry- and exit- protocols for each node to act them individualistically and autonomously.

3.1 Introduction

The mutual exclusion problem is basic in computer operating systems[4] and has been extensively studied by many researchers. The solutions for this problem are classified, by the system architecture on which the problem is to be formulated [32,33], as follows:
(1) Centralized systems in which processes can communicate each other using shared-memory: a) Dijkstra's semaphore [34] and its variant [38],
b) Implementation by hardware instructions, test-set [35], lock [36], and replace-add [37].

(2) Distributed networks in which processes can communicate only through message passing: a) A logical ring of the nodes is constructed on a given network. The privilege of mutual exclusion is represented by a special message, called the token, which is transferred from one node to another around the ring [5]. b) A node requiring to enter the critical section sends a request message containing a time stamp to all other nodes. If more nodes than one request to enter, the node sending the oldest time stamp acquires the privilege[6-8]. c) Let $C=\{Q_1,Q_2,...,Q_k\}$ be a collection of node subsets such that $Q_i \cap Q_j \neq \phi$, $i \neq j$, where C is called a coterie and Q_i a quorum. The node waiting to enter the critical section chooses one of the quorums and collects the consensus of the nodes in the quorum. There are many interesting researches of composing a coterie[9,10] and protocol design[10-12]. d)Using message communication, a virtual acyclic graph is constructed on a given network. By the partial-order relations induced by this kind acyclic graphs, the privileged node is selfchosen sequentially. This method was devised by Chandy-Misra[13].

This chapter treats the mutual exclusion problem for a distributed network of autonomous nodes. The network considered in this chapter belongs to the category (2) of distributed networks. However, the characteristic of the network is different from those considered in [4], and explained below:

«Distributed Networks of Autonomous Nodes»

Each node is assumed to possess only a local view of the network topology and network communication, and behaves individualistically in accord with its own preference. There is no central manager in the network and all nodes are equal in behavior and capability. The most notable characteristic is its self-centered but rational attitude of the node toward any regulation and protocol, and its variability of the network size, that is, frequent occurrence of entries of new nodes and exits of old nodes. Therefore, when the network is large-scale, each node is only possible to possess a local view of topology, identity, communication and state change. In summary, a distributed network of autonomous nodes is characterized by individualism of the node, the variability of

the network structure, and needs of explicit message exchange for information gain.

We in this chapter consider the mutual exclusion (MUTEX) problem only for, the ordinary case, a single-kind/single-quantity network resource, even though similar problems for multiple resources are possible and will be treated in the next chapter. Chandy-Misra protocol[13] is used as a basis of mutual exclusion control, but must be extended, in which an edge between two nodes x and y is interpreted as x and y being in competition for the exclusive use of the shared network resources. If there is only one resource in the network, the network must be sufficiently well connected. In the distributed environment, however, the edge set cannot be so because each node does not know identity of all other nodes being in competition for the same resource. This limitation must be overcome when the initial acyclic graph is set up, which is the heart of the mutual exclusion mechanism we are going to design in Section 3.4. This set-up actually determines the entire course of the network behavior. Our model treats two edge types, *built-in* and *free*, where a built-in edge (x,y) represents a real hardware/software mutual exclusion requirement of x and y, due to some internal reason, but a free edge can be added to the network for cooperation as necessity arises.

Our protocol to be developed may be interpreted as a distributed and autonomous implementation of the well-known token-ring[5]. The token-ring method is well suited for distributed environments, because it does not need the addresses of all other nodes except just two neighbors, provided network fluctuation caused by node entry/exit is correctly handled.

We first formally define the model in Section 3.2 and 3.3, and secondly analyze the feasibility condition of the single resource mutual exclusion for a given network in Section 3.4, and thirdly propose entry and exit protocols of autonomous nodes in Section 3.5, and then design a dynamic mutual exclusion algorithm in Section 3.6. Finally we discuss the algorithm verification in Section 3.7.

3.2 Distributed Environments of Autonomous Nodes

Suppose a network $N=(V,E)$ of the node set V and the edge set E is given with the following interpretation: Each node represents an autonomous element, that is, an independent but cooperating process. Each edge denotes a logical link in a sense that there is a communication path between the two endpoints of the edge and they know each other's identification ID, say telephone number. Knowing the mutual ID is a necessary condition for the two nodes to be able to communicate. That is, even when there is a physical link between two nodes, they cannot communicate if they do not know each other's ID. Thus, the edge in our network does not mean a physical link but a logical relationship of knowing each other's ID. Communication between nodes is effected by message exchange and messages are carried by an independent communication operator.

In the mutual exclusion protocol, to be described in Section 3.3, the node is assumed fully agreed with engaging in cooperation to achieve the common goal, and hence the edge now represents the *alternation constraint* in the sense that any two nodes connected by an edge cannot use the shared resource at a same time, but alternately.

We have two kinds of edges, *built-in* edges, and *free* edges. Two nodes are connected by a built-in edge because of certain internal reasons(hardware or software constraints) that prohibit access to the shared network resource at the same time, while connected by a free edge due to necessity of cooperation in achieving the mutual exclusion. At first, there are only built-in edges in the given network. As to be described in Section 3.4, the feasibility of the mutual exclusion may require more edges than built-in edges. The edges to be added for mutual exclusion control should be called free edges, because there is some freedom to choose.

A simple example of the network is shown in Fig. 3.1, where the printers are shared network resources and computers A,B,C, and D are nodes. The A and B cannot use the printer simultaneously because of the shared-bus between them even if two printers are available. We connect

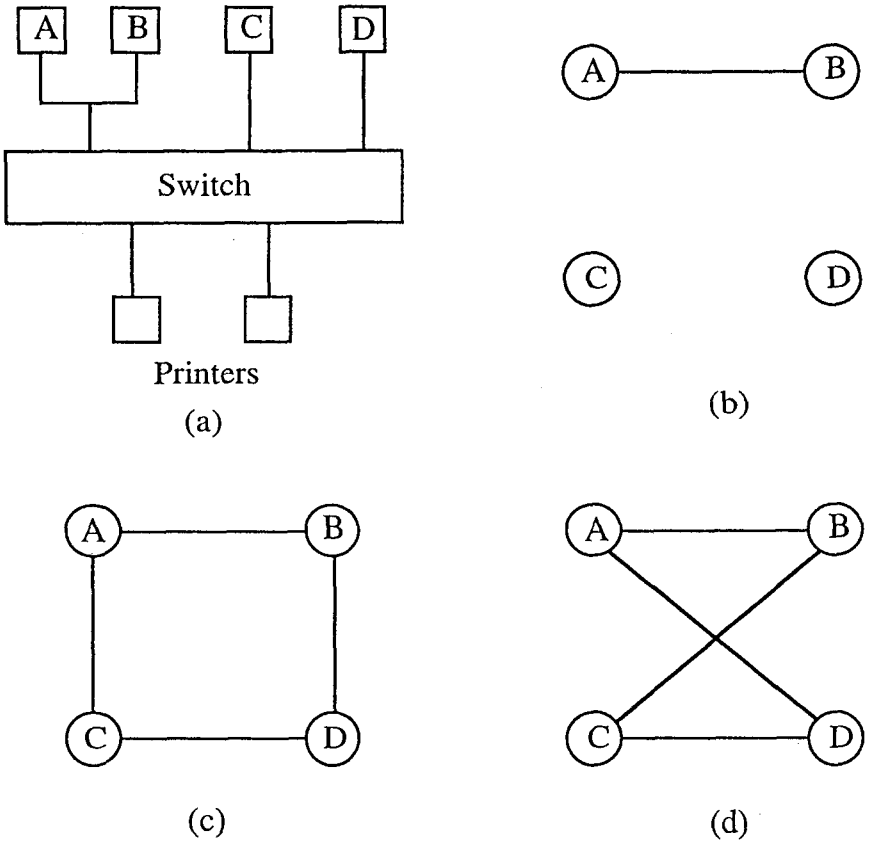


Fig. 3.1 An illustrative example of free edges, built-in edges and networks.

nodes *A* and *B* by a built-in edge(solid line) but leave nodes C and D isolated, as shown in Fig. 3.1(b). As to be shown in Section 3.4, when the number of the shared network resources is one, the mutual exclusion control is possible if and only if there exists a Hamiltonian circuit in the network. So we need to add three free-edges(dotted lines) as in Fig. 3.1(c) or Fig. 3.1(d).

3.3 Mutual Exclusion Protocol: Distributed MUTEX

In this chapter, we propose a new solution for the mutual exclusion problem fitted to a distributed network of autonomous nodes based on Chandy-Misra protocol for the diners problem.

Chandy-Misra[13] expanded the original problem of Dijkstra[34] as follows: A philosopher at each node in the network repeats *thinking*, *hungry*, and *eating*. A fork is placed on each edge. Each fork is shared by two philosophers at both sides of the edge. When a philosopher eats, he needs the forks on the all incident edges. Chandy-Misra protocol determines autonomously the order of eating, based on the partial-order relation produced by initially setting up an acyclic graph on the given network, and by maintaining the acyclicity of evolving subsequent graphs.

In this chapter, we consider shared network resources in the network, which are not treated in [13]. At most k philosophers can eat at a same time when there are available k shared network resources. In the sequel, a node's *eating* corresponds to its *using* the shared network resources. In our model, each node acts independently in accord with the following protocol which is called "Distributed MUTEX". A thinking period is assumed finite and an eating period must be finite. Then, the fairness of our model is guaranteed, that is, any hungry node can eat in a finite time. An example of the protocol execution is shown in Fig. 3.2.

«Distributed MUTEX»

Initialization: Each edge in the network is given a direction so that the graph representing the structure of the network becomes acyclic. Each fork placed on the edge has one of two states, *dirty* and *clean*. A *dirty* fork is placed on the endpoint and a request token on the startpoint of each directed edge, respectively. First, all philosophers are in *thinking*. This situation is called the *initial set-up* that influences future behaviors of the network.

A philosopher who becomes hungry: If the philosopher has request tokens at hand, he sends them to the corresponding philosopher and waits for the responses.

A hungry philosopher who has received all clean forks: The *hungry* philosopher becomes the source of the acyclic graph and can start *eating* immediately. Then, the forks used by him become *dirty*, which implies that all incident edges turn the direction reversed. So, the

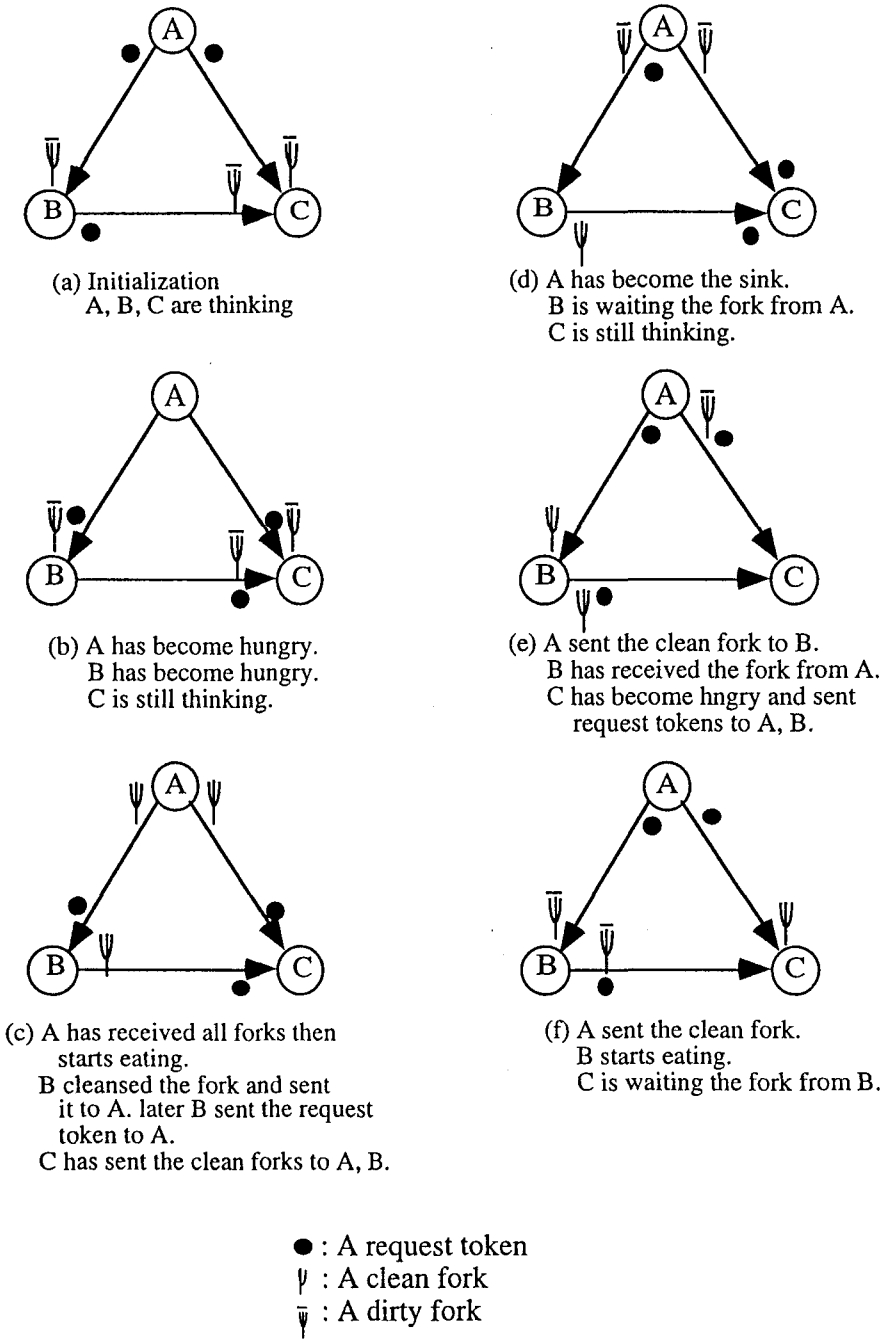


Fig. 3.2 A simple example of Distributed MUTEX

corresponding node becomes the sink. Any message sent to the philosopher is not processed during his *eating*.

A philosopher who is not eating: When he receives a request token, he cleans the corresponding fork and sends it back as the response to the request token.

Iterations: Each philosopher is assumed to repeat the cycle of *thinking*, *hungry*, and *eating* in a bounded time.

Chandy-Misra protocol is so simple, easy for understanding that the following facts are readily obtained using graph theoretical analysis.

(1) After the initial set-up, the philosopher at a source node can collect all clean forks that he needs since he holds all request tokens at hand. A source node has the highest priority.

(2) The philosopher at a source node can start *eating* in a finite time. Then, he turns around the direction of all incident edges and then becomes a sink node, that is, all forks used by him change the states into *dirty*. A sink node has the lowest priority.

(3) The direction of the edge between two nodes a and b is from a to b if and only if (i)a holds the fork shared by a and b, and the fork is *clean*, (ii)b holds the fork, and the fork is *dirty*, and (iii)the fork is in transit from b to a.

(4) The network maintains its acyclicity anytime even if a source node turns into a sink node. Then, another node becomes a new source node.

(5) When the thinking period is finite, the protocol avoids deadlocks and guarantees fairness.

3.4 Initial Acyclic Set-Up

From now on, we use a neutral word *fire* instead of *eat* or *use* resources. An initial set-up of a network seems to determine the entire course of future node firing, as acyclic graph evolves in a sequence. We have to answer the following two questions:

(1) Does the mutual exclusion protocol work?

(2) If so, are the shared resources efficiently used?

When the number of the shared network resources is just one, the mutual exclusion protocol does not work if more nodes than one become the source in the same graph or if there exists a node never becomes a source. On the contrary, for multiple quantity resources, they are not efficiently used if the source nodes are fewer in number than the resources.

In general the multiple resource case must be treated, but here we only consider the initial set-up that makes mutual exclusion possible for the single-resource case. Other cases will be discussed in Chapter 4. Any acyclic graph derived from the initial one must have a single source node and any node must become a source of a some later graph. We now show the main theorem.

Theorem 3.1: *Setting up an initial acyclic graph in which any acyclic graph evolved from the initial one is possible possesses always a single source node if and only if the given undirected network* $G=(V,E)$ *has a Hamiltonian circuit.*

Proof: necessity: Let us denote the single source node in the initial acyclic graph $\vec{G}_0$ by s_0, and let the source node after s_i's firing be s_{i+1} for each $i \geq 0$. Then, the trace of source nodes, $s_0, s_1, \ldots$ continues indefinitely because Chandy-Misra protocol never halts and keeps the graph acyclic forever. We suppose that s_i, $i \geq 1$, is the first node that appears twice at j-th, $j > i$, in the sequence of the source node trace, that is, $s_i = s_j$. Let us denote the sequence of the acyclic graphs by $\vec{G}_0, \vec{G}_1, \ldots$, that is, s_0 is the source node of $\vec{G}_0$, s_i the source node of $\vec{G}_i$. Since $\vec{G}_1$ is generated by turning around all incident edges of s_0 in $\vec{G}_0$, the directed edge (s_0, s_1) exists in $\vec{G}_0$. Inductively, the directed edges (s_k, s_{k+1}), $0 \leq k \leq j$, can be shown to be in $\vec{G}_0$. $\vec{G}_0$ is depicted in Fig. 3.3. Now, we examine topology of the trace. In $\vec{G}_{j-1}$, as shown in Fig. 3.4(a), s_{j-1} is the source node but the directed edge (s_{i-1}, s_i) exists. In $\vec{G}_j$, as shown in Fig. 3.4(b), s_{j-1} becomes the sink by its firing and the directed edge (s_{i-1}, s_i) still exists. Therefore, in $\vec{G}_j$, $s_i = s_j$ cannot be the source node, contradiction. This is because we supposed $i \geq 1$. Consequently, we must deduce $i = 0$, that is, s_0 must be the first node that appears twice in the sequence of the source trace $(s_0, s_1, s_2, \ldots)$.

What we have to prove next is that all other nodes except s_0 appear in the trace between the first and second $s_0 = s_i$. Suppose some nodes have not appeared yet in the trace. Denote the set of these unappeared nodes by $U \neq \phi$. Then, in $\vec{G}_i$, the edges between U and $\overline{U}$ are all directed from U to $\overline{U}$ as shown in Fig. 3.5 where $\overline{U}$ denotes the complementary set of U. However, then, s_0 must be a source by

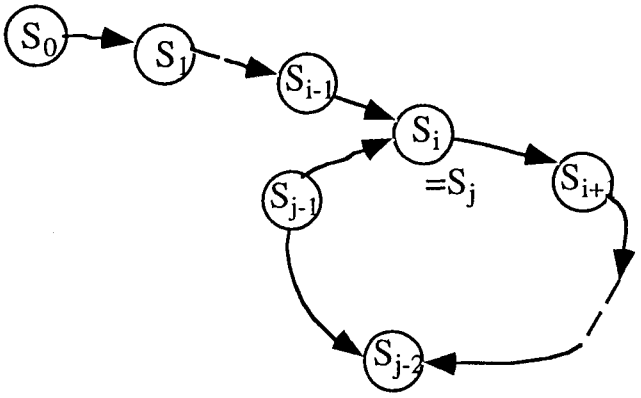
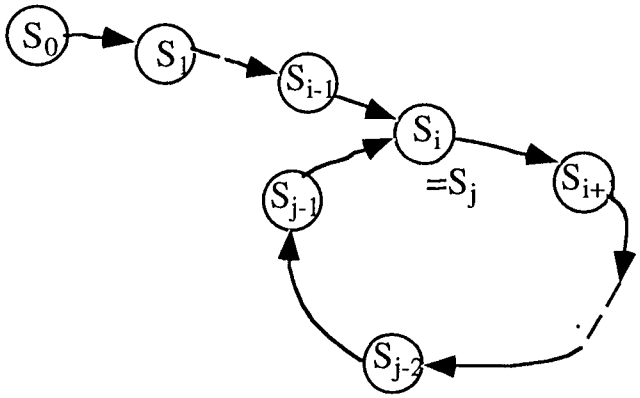
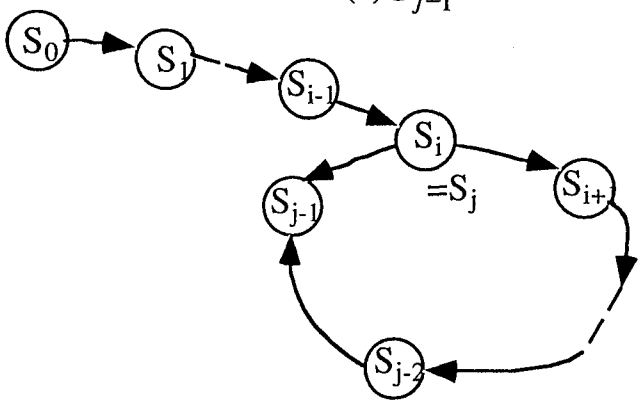


Fig. 3.3 The initial graph $\vec{G}_0$ and trace (S_0, S_1, ..., S_i, ..., S_j, ..)



(a) $\vec{G}_{j-1}$



(b) $\vec{G}_j$

Fig. 3.4 Topology of $\vec{G}_{j-1}$ and $\vec{G}_j$. S_i is not a source in G_j contrary to the assumption.

assumption just made but not the only one because there is no directed cycle in $\vec{G}_i$ and there should exist another source node in U. This is contradiction to the singleness assumption of the source. Therefore, all nodes except s_0 appear once in the trace between the first and second $s_0 = s_i$. Consequently, the trace of the source node constitutes a Hamiltonian circuit in the original G.

Sufficiency: We specify a Hamiltonian circuit $H = (x_0, x_1, ..., x_{n-1}, x_n)$ on G, assign the number i to x_i, and draw edge direction (x_i, x_j) from x_i to x_j when $i > j$. We have formed the initial acyclic graph $\vec{G}_0$(if not acyclic we can contradict). We prove by induction that any acyclic graphs evolved from $\vec{G}_0$ by the protocol have the one and only one source node without starvation of any node.

(1) In $\vec{G}_0$, x_0 is the unique source node.

(2) Assume now that the statement holds for no greater than i, we also show that it must hold for $i+1$. The situation of $\vec{G}_{i+1}$ is shown in Fig. 3.6. Showing the following facts is easy.

(1) The directed path $x_0, x_1, ..., x_i$ exists and let $V_1 = \{x_0, x_1, ..., x_i\}$.

(2) The directed path $x_{i+2}, x_{i+3}, ..., x_{n-1}$ exists and let $V_2 = \{x_{i+2}, x_{i+3}, ..., x_{n-1}\}$.

(3) The edges between V_1 and V_2 are directed from V_2 to V_1.

(4) x_{i+1} is a source node.

Since $(x_{i+2}, x_{i+3}, ..., x_{n-1}, x_0, x_1, ..., x_i)$ is a directed path, x_{i+1} must be the only one source node of $\vec{G}_{i+1}$. Q.E.D.

In the evolution of acyclic graphs, the source node moves along the Hamiltonian cycle. In the case there exist more than one Hamiltonian circuit in the original network G, any one of them can be chosen.

The method of initial acyclic set-up used in the previous *proof* is of theoretical interests. The current acyclic network should be considered as an outgrowth of a history of node entry/exit processes applied onto a small scale network made acyclic at the start. Entry and Exit Protocols will be described in Section 3.5.

Corollary 3.1: *When the built-in edge set of the original G is not enough for providing a Hamiltonian circuit, the free edges should be introduced in order to assure existence of at least one Hamiltonian circuit.*

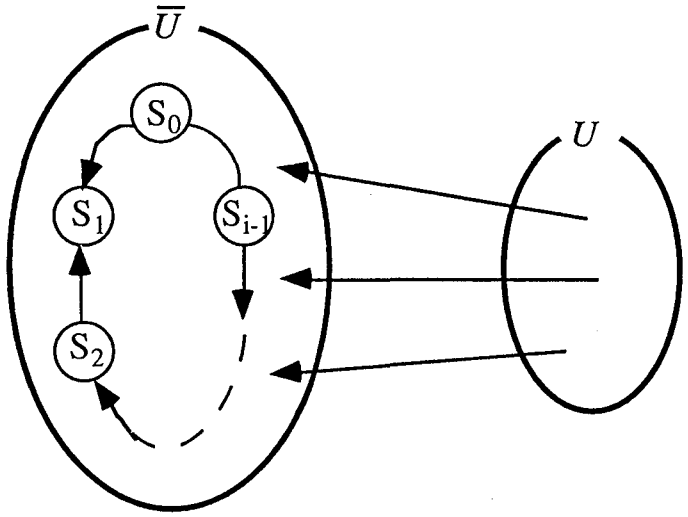


Fig. 3.5 Topology of $\vec{G}_i$ where S_0=S_i and $U = \phi$.

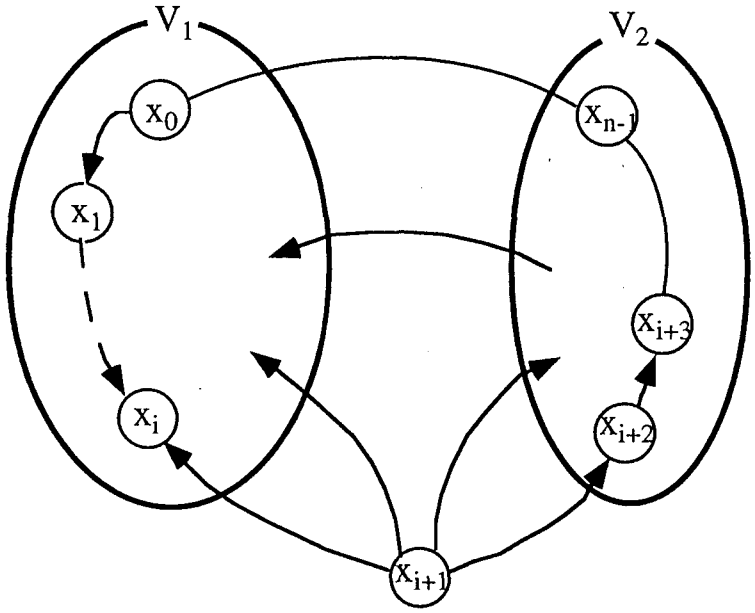


Fig. 3.6 Topology of $\vec{G}_{i+1}$.

«Initial Acyclic Set Up for A Single Resource MUTEX »

1° Select a Hamiltonian circuit of G.

2° Choose arbitrarily a node on the circuit.

3° Assign a node number to each node such that there is a monotone sequence of node numbers either increasing or decreasing along the circuit starting from the chosen node.

4° Assign the edge direction so that the startpoint of any arrow corresponds to the node with a smaller number.

3.5 Entry and Exit Protocols

A distributed network of autonomous nodes varies its structure by frequent occurrence of entries of new nodes and exits of old nodes. Entry/exit protocol is essential in such an environment. In this section, we design an entry/exit protocol that keeps MUTEX functioning intact and endows the network *non-consecutive* node sequence fault tolerance in a sense that any non-consecutive node sequence breakdown does not give a fatal damage to MUTEX. The fault tolerance, however, is not a main topic of this chapter and hence not well-discussed.

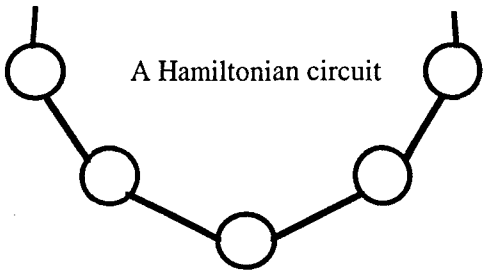
We describe non-consecutive node sequence fault tolerant networks in more detail in Subsection 3.5.1, and propose Entry/Exit Protocols in Subsection 3.5.2 .

3.5.1 Non-Consecutive Node Sequence Fault Tolerant Networks

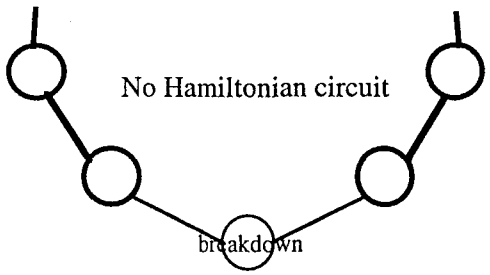
We, in our model, consider node breakdowns but not edge breakdowns because edges in the network are considered as logical entity.

If a node breaks down, it must be detected and eliminated from the network, which may affect the mutual execution protocol. Theorem 3.1, as we have seen, says that a Hamiltonian circuit is needed on the network for the single-resource mutual exclusion, in other words, the mutual exclusion protocol does not work if there is no Hamiltonian circuit on the network. Let us examine effects of node breakdowns to the protocols.

(1) First, let us consider the simple ring network shown in Fig. 3.7(a). It is the simplest network which has a Hamiltonian circuit. Suppose that a node breaks down as shown in Fig. 3.7(b). Then, there is no longer a Hamiltonian circuit. As a result, the mutual exclusion protocol cannot function correctly and hence damage will be incurred. Moreover, a new Hamiltonian circuit must be established with much costs for adding new free edges.

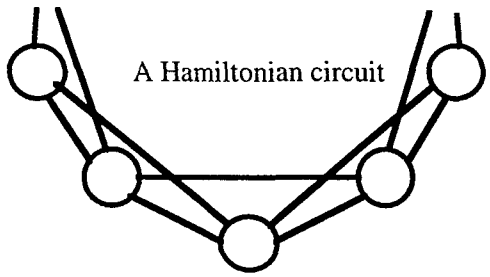


(a) Before breakdown

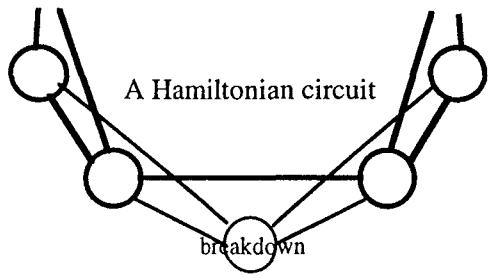


(b) After breakdown

Fig. 3.7 Non fault tolerant network.



(a) Before breakdown



(b) After breakdown

Fig. 3.8 Non-consecutive node sequencce fault tolerant network

(2) We consider a network which has four neighbor nodes, two nodes immediately preceding, the others two immediately succeeding on the current Hamiltonian circuit as shown in Fig. 3.8(a). We suppose that a node breaks down like Fig. 3.8(b). There is still a Hamiltonian circuit on the network even if the breakdown node and its incident edges are removed. Moreover, It is easy to understand that there is one source node in any acyclic graphs derived from the remaining network. Therefore, we can say that one node breakdown does not affect the protocol in this type network.

Next, we consider plural nodes breakdown. In this case, we can easily obtain the following facts.

(1) A set of node breakdowns in which any two are not consecutive on the current Hamiltonian circuit does not affect the protocol in the network.

(2) Any consecutive breakdowns disconnect the Hamiltonian circuit and cause the protocol malfunction.

In this chapter, networks in which the above two properties hold are called *non-consecutive* node sequence fault tolerant networks.

Generally, k-consecutive node sequence breakdown at the current Hamiltonian circuit does not give a fatal damage to MUTEX if and only if each node in the network has at least $2(k+1)$ neighbor nodes, the first $k+1$ are the immediate preceding nodes, the others the immediate succeeding nodes on the current Hamiltonian circuit.

3.5.2 Entry and Exit Protocols

What we do here is, anticipating future problems, to construct a network of at least *non-consecutive* node sequence fault tolerance. To make description simple, all edges are assumed free. When some node has built-in edges, necessary modification in the description is self-evident and hence not mentioned.

Let us denote four adjacent node of p along the current Hamiltonian circuit by

bp(before p), bbp(before bp), ap(after p), aap(after ap).

These nodes are located on the current Hamiltonian circuit HC in the order of

$$bbp,\ bp,\ p,\ ap,\ aap.$$

Each node possesses ID, say telephone number, of such adjacent nodes in its own memory as well as those of nodes associated to built-in edge. Since we are assuming non-existence of built-in edges, all edges are free and allowed to connect points of necessity. The set of *ID*s of the adjacent nodes is called "the adjacent nodes information(ANI)". Let us denote the *ID* of a node x by $\#x$. Then, the *ANI* of p is represented by a 4-tuple,

$$(\#bbp,\#bp,\#ap,\#aap).$$

The entry and exit protocols are presented below with their processes illustrated in Fig. 3.9, Fig. 3.10, respectively.

«Entry Protocol»

For node x which wants to enter the network:

1° Choose any friendly node P as a host HOST and send an *Entry* message to it. The node x will join the MUTEX cycle in front of HOST.

2° Choose another node as a HOST. When a *Welcome* message is received, wait. When a *Busy* tone is heard, try again some time later when the *Sorry* message is received.

3° Receive from the HOST data $\#bbP,\#bP,\#aP$. Send a *Hello* message containing ID of $HOST$ to each node corresponding to $\#bbP,\#bP,\#aP$.

4° On receiving *Welcome* messages from nodes to which x has sent a *Hello*, modify the *ANI* as follows.

$$(\#bbP,\#,\#HOST,\#aP).$$

And send a *Ready* message to *HOST*.

5° Upon receiving a *Complete* message from *HOST*, place the request token on every new edge just established, and send a Thank message to all node involved.

For node P which has received an Entry message from x:

If P is waiting for its exit, P sends x a *Sorry* message, if P is using the resource, a *Busy*, otherwise a *Welcome* and enqueue the *Entry* message. When P becomes enable, P processes the *Entry*s in its queue as follows, where *enable* means that P has all clean forks on his incident edges.

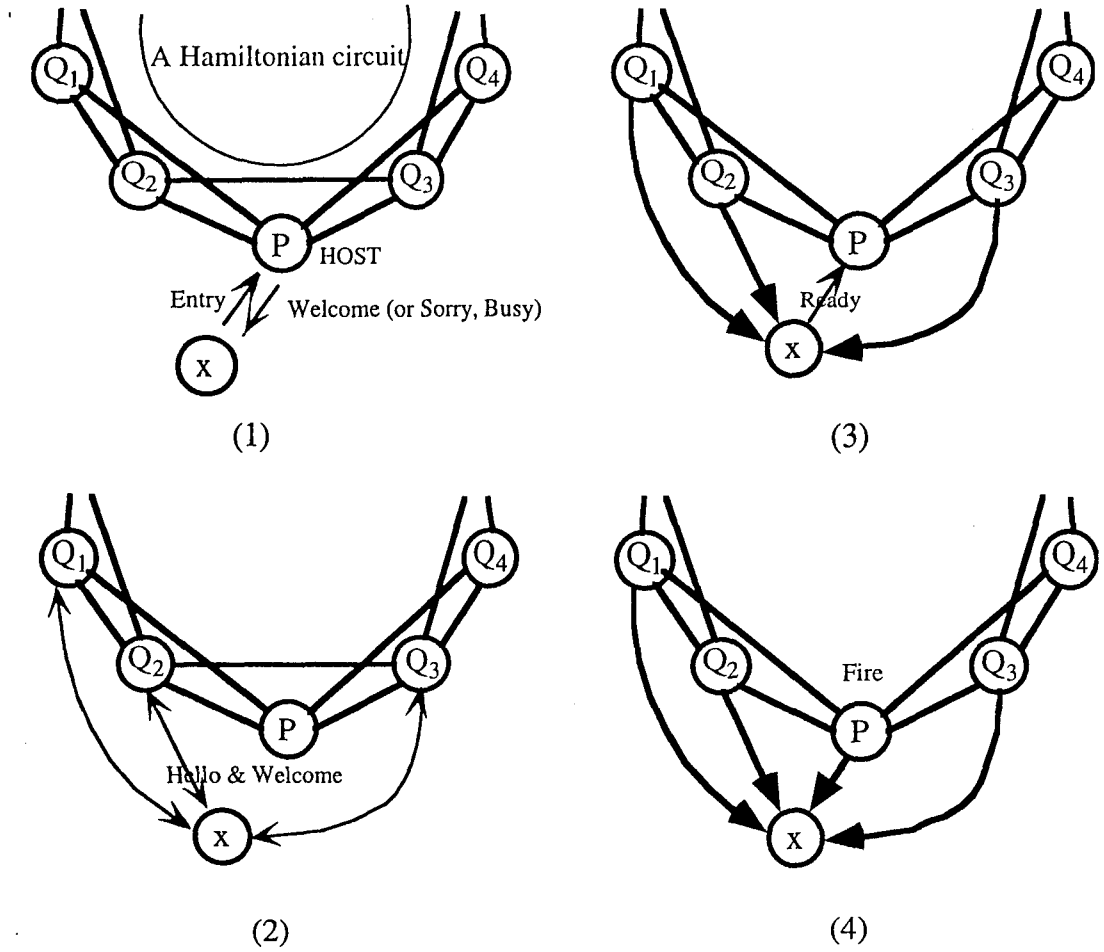


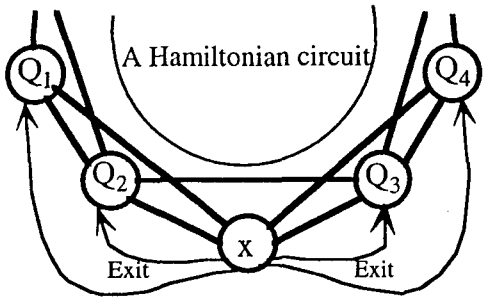
Fig. 3.9 Message exchange in the entry process of node x.
(1) -> (2) -> (3) -> (4)

The *Entry(x)* is accepted on the first-come first-served basis but not handled before P becomes firable. On becoming firable, the actual firing is put off until the HOST action of the following is completed.

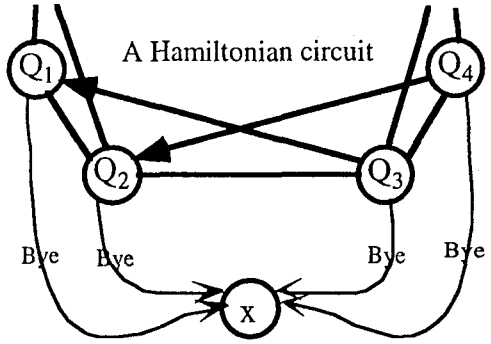
1° Send $\#bbP,\#bP,\#aP$ to x.

2° On receiving a *Ready* from x, and *Welcomes* from the neighbors, reestablish the Hamiltonian neighborhood by performing a remove the edge (P, bbP) and reassignments $\#bbP{:=}\#bP$, $\#bP{:=}\#x$. The entry of x is now completed with the request token at x and the clean fork at the Hamiltonian neighbor of x. Send a *Complete* message to all.

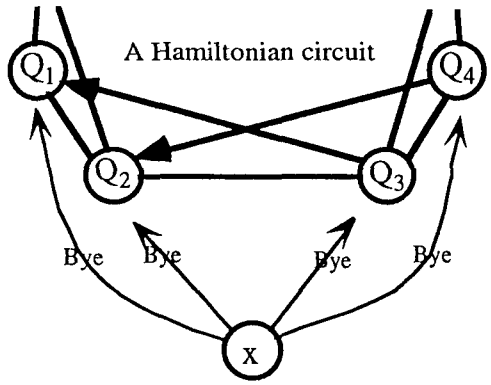
3° The Entry process completes when a *Thank* message is received.



(1)



(2)



(3)

Fig. 3.10 Message exchange in the Exit process of node x.

4° When all *Entry*s in waiting are processed, P is ready to fire for itself.

For node Q which has received a Hello message containing *#HOST*:

1° Send a *Welcome* message to x and the HOST. Then, an edge is established between x and Q where one *clean fork* is assigned at the side Q and one *Request Token* at the side x.

If HOST=aaQ, delete the edge between Q and HOST.

2° Modify the *ANI* according to the following rule.

case when #aaQ=#*HOST*:

Reassign, #aaQ:=#x;

Remove the edge(Q,*HOST*).

case when #aQ=#*HOST*:

Remove the edge(Q,aaQ);

Reassign, #aaQ:=#aQ,#aQ:=#x.

case when #bQ=#*HOST*:

Remove the edge(Q,bbQ);

Reassign, #bbQ:=#x.

The Entry process completes when a *Thank* is received.

«Exit Protocol»

For node x which wants to exit: When x wants to exit, it sends an *Exit* to itself, that is, puts the *Exit* into its queue. After the *Exit*, any received message are responded with *Sorry*. Then, x behaves as he is hungry in order to become enable, that is, requests all clean forks of the incident edges. When x becomes enabled, the messages in the queue are processed sequentially. When dequeue=*Exit*, x executes the following protocol.

1° Send to each neighbor node an *Exit* message containing information t defined below:

case when the neighbor node = bbx

assign, t:=#ax.

case when the neighbor node = bx

assign, t:=#aax.

case when the neighbor node = ax

assign, t:=#bbx.

case when the neighbor node = aax

assign, t:=#bx.

2° On receiving a *Bye* message from the neighbor, send back a *Thank* message for acknowledgment.

3° When the *Thanks* is sent to all the neighbor, the exit is completed.

For node Q which has received an Exit message from x:

1° Receive an *Exit* and store the information t.

2° Establish an edge (Q,t).

3° Remove the edge (Q,x) by transmitting a *Bye* message to x.

4° The exit is completed when a *Thank* is received.

3.6 Dynamic MUTEX Algorithm

We present a design of the dynamic MUTEX algorithm that links the entry protocol that enables each autonomous node to enter the network, the Distributed MUTEX enabling each node to join the MUTEX process, and the exit protocol enabling each node to exit the network.

«Dynamic MUTEX algorithm for each node P »

1° When P wants to enter the network, choose a HOST and send an *Entry* request message to the HOST and follow the Entry Protocol.

2° Once entered the network, follow the basic Chandy-Misra MUTEX protocol.

3° When P wants to exit the network, send an *Exit* request message to itself and then follow the Exit Protocol.

The detail state transition diagram of the Dynamic MUTEX Algorithm is depicted in Fig. 3.11, 3.12, 3.13, 3.14. The node travels the diagram by starting from Entry, circling around the *MUTEX cycle*, and reaching to *Exit* state.

The notation has the following meaning:

-MSG: message MSG is sent.

+MSG: message MSG is received.

MSG*: all message MSGs are sent to or received from all incident edges.

+Hello(y): message *Hello* is received from y.

-Welcome: message *Welcome* is sent out.

-EntryProtocol: Entry Protocol is initiated.

+EntryProtocol(dequeue): Entry request message is taken out of the queue and is processed in accord with the Entry Protocol.

+Fork*: all forks are received from neighbors.

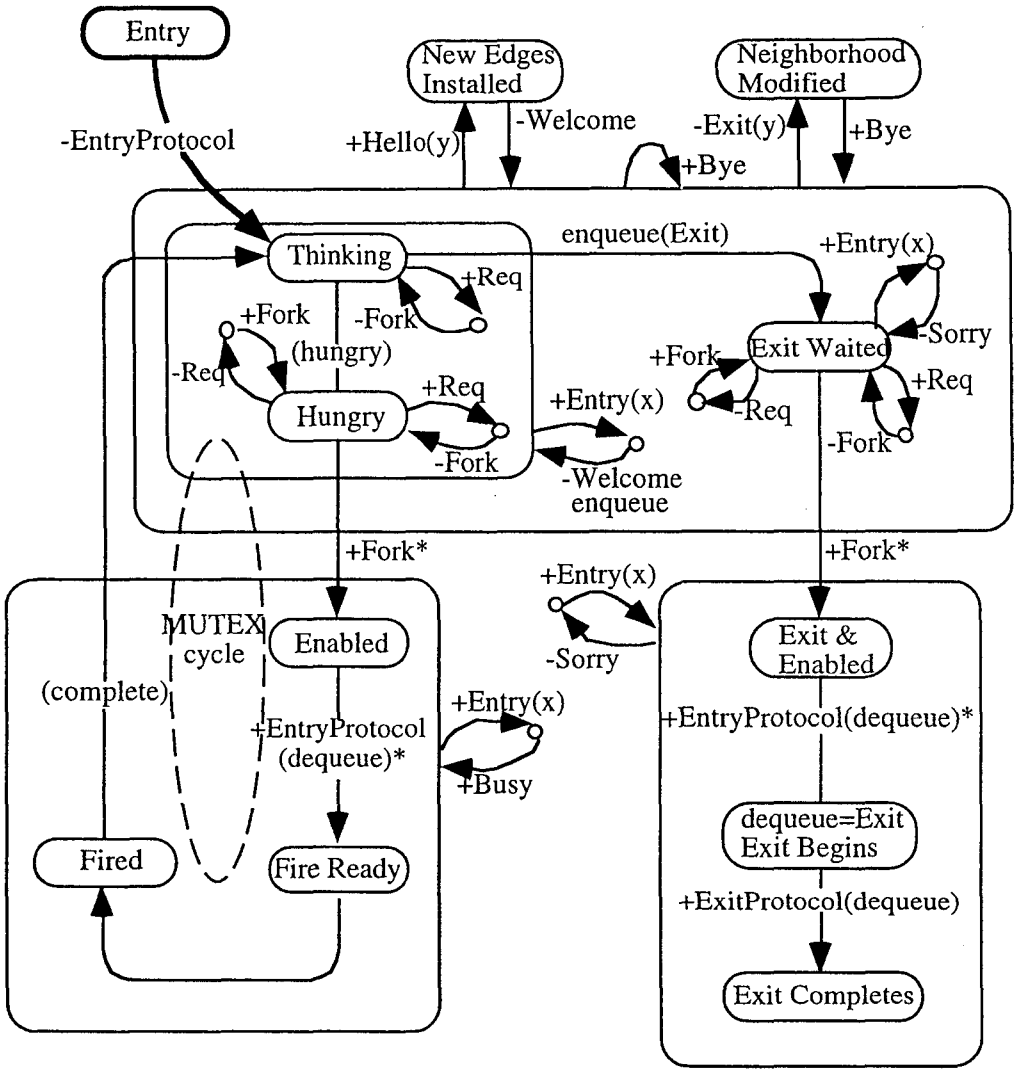


Fig. 3.11 State transition diagram of dynamic MUTEX algorithm for node *P*.

+EntryProtocol(dequeue)*: Entry requests in the FIFO queue are before Exit all processed.

enqueue: the received message is put into the FIFO queue.

dequeue: the message is fetched from top the FIFO queue.

Other messages are similarly interpreted, and hence their descriptions omitted. The main diagram Fig. 3.11 is composed of the entry path, the MUTEX cycle, the exit path, and the house-keeping message loops as to be explained below.

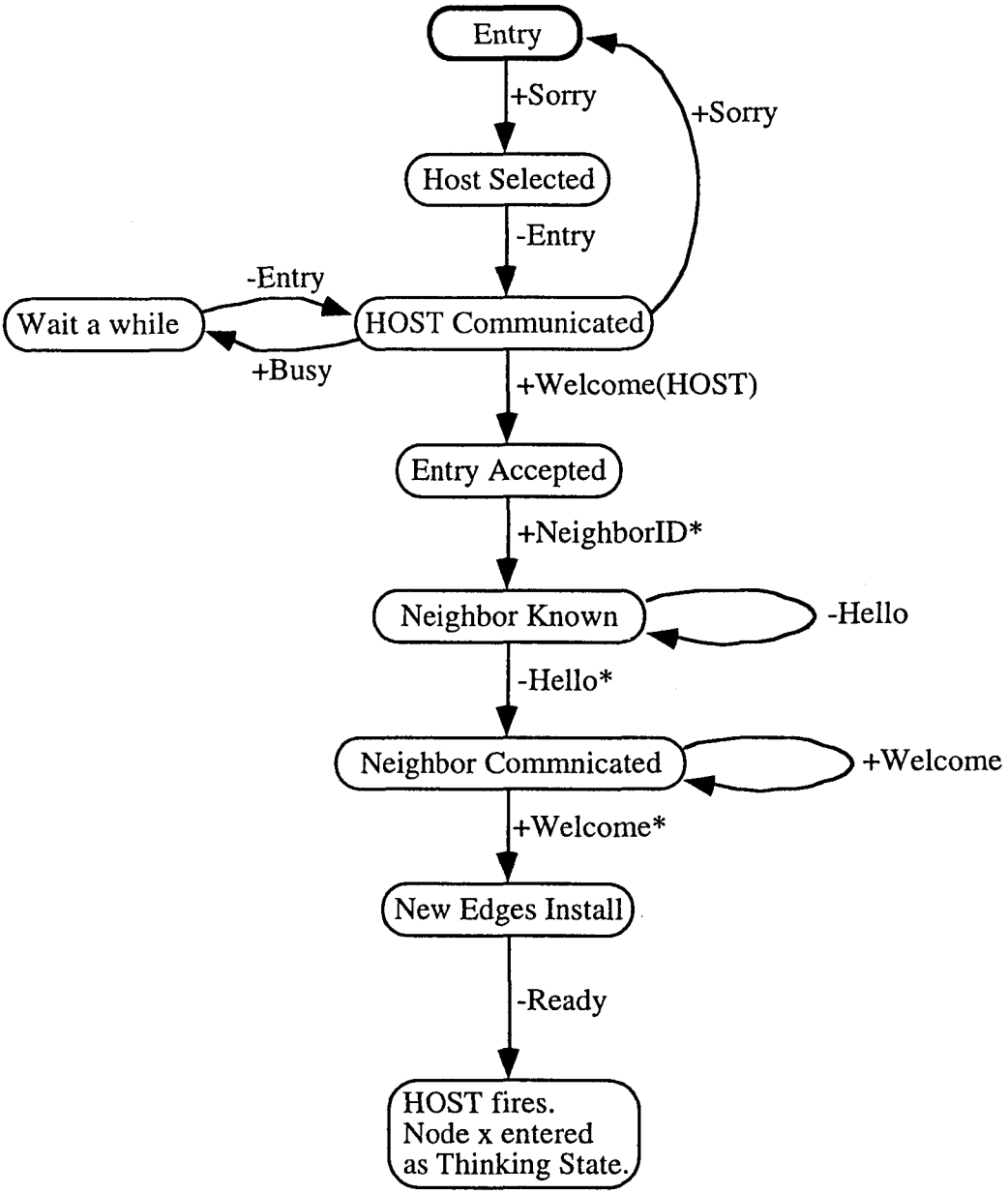


Fig. 3.12 State transition diagram of +EntryProtocol for node *x*.

(1) **Entry path:** An autonomous node can enter the network by performing -EntryProtocol at the *Entry state*. See Fig. 3.12 for the state transition diagram. The node is entered as *Thinking state*.

(2) **MUTEX cycle:** The node that has entered the network now circulates around the MUTEX cycle containing the edge labeled by +EntryProtocol(dequeue)*

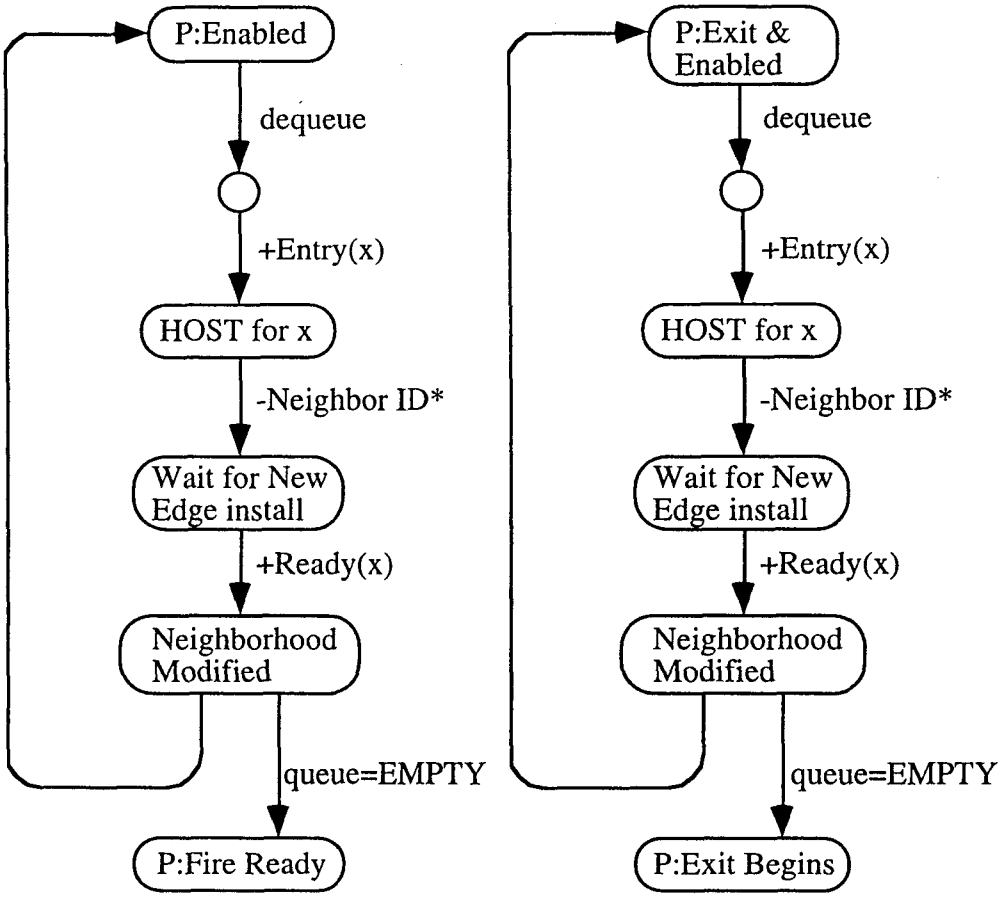


Fig. 3.13 State transition diagram of -EntryProtocol(dequeue)* for node *P*.

that represents repeat of the EntryProtocol applied to *Entry* messages fetched from the queue until exhausted. The state transition diagram is depicted in Fig. 3.13.

(3) **Exit path:** The node can exit the network by issuing Exit request message to itself, which is put into the queue, and by traversing the path to *Exit state*. The path contains edges labeled by

+EntryProtocol(dequeue)*
+ExitProtocol(dequeue),

where the latter represents application of ExitProtocol to the *Exit* request message fetched from the queue. See Fig. 3.14 for the state transition diagram.

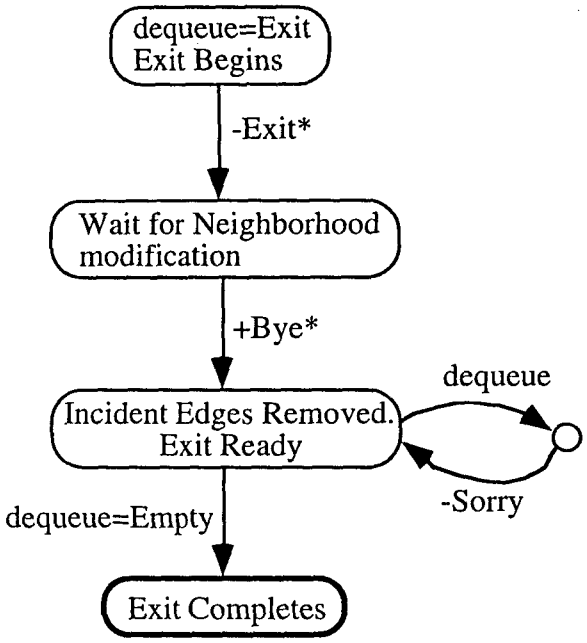


Fig. 3.14 State transition diagram of +EntryProtocol for node *x*.

(4) **House-keeping message loops:** The loop is a cycle of at most two edges attached to a state or a composite of several states, that performs a house-keeping task of the following:

1) (+Req, -Fork) loop at *Thinking* or *Hungry* sends back a cleansed fork if the fork is *dirty*(recall the fork is dirty) upon receipt of the request token from a hungry neighbor.
2) (-Req,+Fork) loop at *Hungry* performs just the opposite mentioned above, seen from the hungry *P* itself.
3) (+dequeue,-Sorry) loop at the composite state including *Exit* Completes in Fig. 3.11 informs, those wanting to enter, that another HOST must be chosen because *P* itself is leaving the network.
4) (+Entry(x),-Busy) loop at the composite of *Enabled*,*Fire Ready*,*Fired* sends back busy-tone when called by those *x* wanting the entry.
5) There are four loops at the composite of *Thinking*, *Hungry*, *Exit Waited*. This composite state is a main waiting state where messages needing cooperation come in.

(+Hello(y),-Welcome*) loop cooperates *y*'s entry, at non-HOST node, by installing new edges for the modified Hamiltonian neighborhood.

(+Exit(y),-Bye*) loop cooperates *y*'s exit by installing new edges for restoring Hamiltonian neighborhood to be restored.

(+Entry(x), -welcome.enqueue) loop acknowledges request of y to act as HOST for y's entry. This entry request is put into the waiting queue for later processing.

3.7 Verification of Dynamic MUTEX Algorithm

A distributed algorithm based on message passing is interaction of concurrent processes and best analyzed by the diagram of its state transitions caused by issue/receipt of messages or internal actions/evolutions.

The verification items of the algorithm are the following.

(1) **Progress:** The algorithm never reaches a dead state from where no transition is possible in contrary to the specification.

(2) **Fairness:** Any one can enter/exit the network within a finite time. Moreover, within the network, any one has the equal access to the resource.

(3) **Correct Implementation:** Necessary and sufficient number of states must be distinguished and defined. This induces, in a natural way, design of message types and their task assignments. The component processes must work correctly and their interactions must achieve the specification without ill side-effects.

3.7.1 Progress Property

The progress property can be verified by checking the state transition diagram on the following items.

(1) Every issued message must be received by the intended receiver and vice versa. That is, (- MSG) must be matched by (+MSG) and vice versa. The verification process is exhaustive but tedious check on every message label on the edge. This has been done.

(2) The message scheduled to be issued must actually be issued. That is, -MSG must take place. For example (-Req) at *Hungry* is capable to

take place because a hungry node eventually possesses a request token according to Chandy-Misra protocol. (-Welcome) at *New Edge installed* is capable of taking place, because node P is in the idle state and cooperative, and (+Hello(y)) contains all necessary node IDs for installing new edges. The exhaustive check can be carried out on every message label.

(3) The message scheduled to be received must be actually received. For example (+Fork) at *Hungry* occurs eventually due to Chandy-Misra protocol. (+Fork*) at *Enabled* means all forks are received from the neighbor nodes, and occurs eventually because Chandy-Misra protocol is progressive. (+Exit(y)) at *neighborhood modified* occurs since node P is idle (hungry or thinking) and obliged to receive and process any *Exit* message which contains all necessary node IDs to modify neighborhood of the Hamiltonian circuit.

A special treatment is necessary for composite edges, shown by bold lines in the diagram of Fig. 3.11, whose labels are:

-EntryProtocol from *Entry* to *Thinking*, whose state transition diagram is shown in Fig. 3.12;

+EntryProtocol(dequeue) from *Enabled*(or *Exit & Enabled*) to *Fire Ready*(or *Exit begins*), whose state transition diagram is shown in Fig. 3.13;

+ExitProtocol(dequeue) from *Exit Begins* to *Exit Ends*, whose diagram is in Fig. 3.14.

By checking the diagrams of Fig. 3.12, 3.13, 3.14 we can prove that those are progressive and occur in a finite time.

(4) The message must contain enough information to cause the state change, and the state change must settle in a finite time. Change from *Thinking* to *Hungry* is internal action, and assumed to take place in a finite time.

Now we are in position to conclude,

Theorem 3.2 *Dynamic MUTEX algorithm is progressive and a node takes a finite time to traverse from Entry to Exit if the number of the MUTEX cycling (i.e. resource usage) is finite.*

3.7.2 Fairness Property

The resource usage takes place at node by node along the Hamiltonian circuit. Hence one cycling is called the *MUTEX round*, and the cycling time is called the *round time*. Fairness has three components, entry, resource-usage, exit. The entry position is just before the HOST on the Hamiltonian circuit with the lowest priority, that is, as the sink, and all entries waiting are processed when the HOST is enabled and before the HOST fires. Hence the entry succeeds within a single MUTEX-round time. The resource usage is known fair due to Chandy-Misra protocol. Similarly, the exit succeeds within a single MUTEX-round time.

Hence we can conclude,

Theorem 3.3 *Dynamic MUTEX algorithm is fair to every autonomous node, with respect to entry, resource-usage, and exit.*

3.7.3 Correct Implementation of Concurrent Processes

Each node engages in one of four processes at any time instant; Idle process, Entry process, MUTEX process, and Exit process, cyclically in this order. The nodes are asynchronously operating one of these processes disregards of others. Now a question arises, do they correctly run as intended? We answer this question positively.

First we note that MUTEX processes are serial in a sense that only a single node has the highest priority in the message handling and such a node moves along the Hamiltonian circuit.

Node P executes the HOST role only when it is enabled and before it uses the resource for oneself or it executes its own exit process, where the latter two are exclusive events. Exit process is executed only when P is enabled. So in the whole network it is clear only a single process is executing at any instant of time.

Theorem 3.4 *The processes of Dynamic MUTEX Algorithm are serialized and hence are correctly running as intended.*

Serialization of the three processed is admittedly conservative and deliberately adapted in order to avoid ill-effects of parallelization. There is some room for parallelizing. Entry and Exit with respect to MUTEX which introduces added complexity so not mentioned in this dissertation.

3.8 Concluding Remarks

A distributed network of autonomous nodes is characterized by autonomy of the node, the variability of the network structure, and need of explicit message exchange for information gain. In this chapter, we have designed a dynamic mutual exclusion algorithm for distributed autonomous environments based on Chandy-Misra protocol for diners problem[13], which realizes a distributed implementation of the token ring method[5]. We have obtained the communication feasibility condition that makes mutual exclusion possible and proposed entry and exit protocols for individual nodes.

Chapter 4

Concurrency Analysis of Acyclic Graph Evolution and Extension of Distributed MUTEX to Multiple Shared Resource Cases

We discuss properties of acyclic graph evolution driven by node-firing. We introduce a new notion "canonical circuit-cover", and show the condition that the maximum concurrency of an acyclic graph evolution is no more than a given desired value, k. Using the theorem, the initial acyclic graph can be established in which the maximum concurrency of the acyclic graph evolution initiated is no more than k. A marked graph, a subclass of Petri nets, is often utilized as a proof tool in analysis.

From theoretical interests, we also consider a problem to establish the initial acyclic graph for a given graph. To find a canonical circuit-cover whose cardinality is k is difficult theoretically. In order to show this difficulty, we define the decision version of the problem and show its NP-completeness, and then we propose a method using a genetic algorithm to solve it.

By using results of the concurrency analysis, the Distributed MUTEX is extended to the multiple shared resource case. Entry & exit protocols for the case are omitted since they can be extended straightforwardly from those of the single resource case and are not important topics.

4.1 Introduction

We discuss in this chapter properties of acyclic graph evolution driven by node-firing proposed in the previous chapter. The idea of the acyclic graph evolution was originated in the paper [13] by Chandy and Misra. (We in this chapter use a word "graph" for "directed graph" or "undirected graph" if no confusion arises.)

Chandy and Misra proposed a mutex exclusion protocol for the single shared resource in a distributed network. Before the algorithm starts, acyclicity with single source/sink is virtually introduced, on a given (bilateral) communication graph G which is tacitly assumed completely connected, by communicating two special tokens (messages), one fork-token and one request-token placed at the terminal nodes of each edge. That is, the tokens are distributed over the edge set of the communication graph G such that a directed graph $\vec{G}$ obtained from G by replacing each edge by diedge from fork to request is acyclic. Hence after, for each diedge, its start node is understood to possess the higher priority for the shared resource usage than its end node. Since the edge direction is acyclic globally, a total order relation is induced on the graph, and hence only a single node becomes the source node. From this singleness, the privilege for using the single shared resource can be given to the source node. Just after releasing the resource after using it, message exchange taking place between the source node and its neighbor nodes triggers reversal of the directions of all the diedges incident to the source node. In this chapter, we use a neutral term "to fire a source node" instead of "to use the resource and release it". By the edge reversal, the source node will turn into a sink node and a new source node will emerge (note that only a single source node should be appeared). This process of the source firing is to be iterated. The algorithm ensures mutual exclusion with fairness and liveness. That is, there exists an only single source node at any instance of the evolution trajectory emanating from the given initial acyclic graph in which every node can be firable(fairness) and deadlock cannot occur(liveness). We must note that in the Chandy-Misra model a given graph is assumed completely connected and hence any of its acyclic directioning produces a single source (i.e., firable) node only.

We extended the Chandy-Misra model to allow the graph only partially connected. An acyclic directioning of Chandy-Misra type on a partially connected graph, in general, induces just a partial order relation. In the partial order relation, multiple source nodes may appear at some stages of an evolution of acyclic graphs. In the previous chapter, we presented a topological condition and algorithmic schemes

of the initial acyclic directioning that ensure exactly one source (i.e., firable) node at any instance of the acyclic graph evolution. This extension leads to k-mutual exclusion problems in which k resources are available, each for exclusive use throughout the network nodes. Some additional graph theoretical properties were presented as well, which are interesting in its own right. A marked graph, a subclass of Petri nets, is often utilized as a proof tool in analysis. There is a wealth of net theory, contributed from Petri net community[31,39-49], readily available for our research.

T. Murata et al [47-49] introduced the token distance matrix to be used in concurrency investigation of marked graph and presented a method for finding "optimum" initial markings to produce a desired concurrency. In this line of thought, we present here the notion, first proposed in author's paper[50], of canonical circuit-cover, discuss firing concurrency (the number of firable nodes), and investigate topological conditions for controlling the concurrency below a some fixed constant k, at any instance of the evolution trajectory.

Before proceeding to discussion, we present an illustrative example of an acyclic graph evolution for assisting readability of our presentation.

Example 4.1: We use in this chapter a neutral term "to fire a source node" instead of "to use the resource and release it". In Fig. 4.1, 4.2, and 4.3, the nodes represented by the shaded circle are sources and firable.

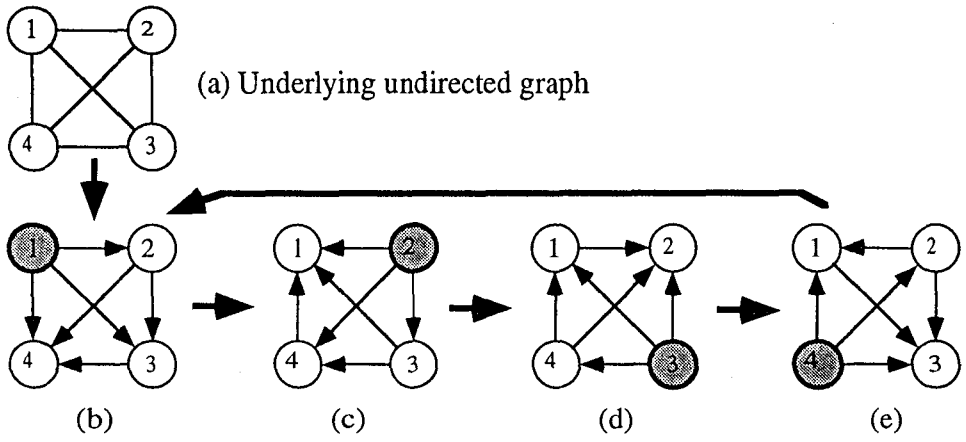


Fig. 4.1 Example1: Evolution trajectory (bcde)*
Underlying graph is completely connected.

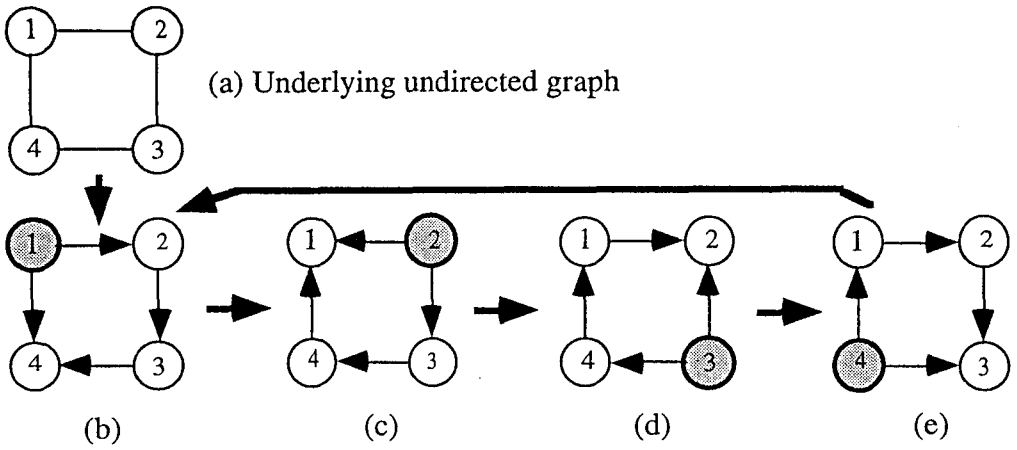


Fig. 4.2 Example2: Evolution trajectory. (bcde)*
Underlying graph is NOT completely connected.
There is only single source at any stage of the evolution.

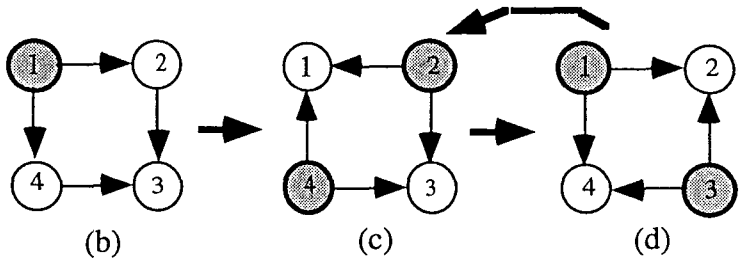


Fig. 4.3 Example3: Evolution trajectory b(cd)*.
Underlying graph is NOT completely connected.
Multiple sources exists at some stage and
all of them are fired simultaneously.

Fig. 4.1 shows an example of the Chandy-Misra model and its evolution trajectory of acyclic graphs. Note the given undirected graph in Fig. 4.1.(a) is completely connected. Fig. 4.1(b) presents an initial acyclic directioning of the given graph from which a periodic sequence of acyclic graphs evolves by source node firing. Note that any acyclicdirection of the completely connected graph always produces a single source. By the source-firing, the node 1 reverses the direction of all its incident edges. Then the node 2 becomes a new source as shown in Fig. 4.1(c). The evolution trajectory is represented by (bcde)* where * denotes the repeated concatenation.

Fig. 4.2 and 4.3 show other examples. The underlying graph for these examples is partially connected as shown in Fig. 4.2(a). Note that there is just a single source at any instance of the evolution trajectory (bcde)* originated from this particular acyclic direction of Fig. 4.2(b). But in Fig. 4.3(b), the underlying graph is the same but the initial acyclic direction is different on the edge <3,4> . This seemingly trivial difference in the initial direction causes a great difference in the evolution trajectory of acyclic graphs. That is, there can appear multiple sources nodes at some instances even when the initial graph has a single source. (Fig. 4.3 shows an example that all multiple sources fire simultaneously. This firing rule will be defined as the unison firing in Section 4.2.)

As far as the authors aware, the notion of acyclic graph evolution is novel and the associated literature is almost non-existent even though the acyclicity itself is well-known.

During course of investigation, we found the Petri net (marked graph) model provides a good mathematical tool for stating and proving dynamic behaviors but the acyclic graph model is more appealing to wider audience. For this reason we decide that definitions and theorems should be stated in the acyclic graph model. We first formally define the evolution model and describe the conversion rule of acyclic graphs into equivalent Petri net in Section 4.2. In Section 4.3 we present graph theoretical properties, and finally discuss constructive schemes of initial acyclic directioning that ensures a stated concurrency. In Section 4.4 we propose a method to design an initial acyclic graph for desired concurrency. In Section 4.5 we extend the proposed protocol, Distributed MUTEX, in Chapter 3 to multiple shared resource cases.

4.2 Basic Concepts and Notations of Acyclic Graph Evolution

In this section, we show basic concepts and notations of acyclic graph evolution. We assume that the reader is familiar with graph and Petri net theory (see Refs.[29,30] for graph theory, and [31] for Petri nets).

4.2.1 Acyclic Graph Evolution

A (undirected) graph is denoted by $G(V,E)$, where V and E are the sets of nodes and edges; if edge e is incident to node u and v, then it is represented by the unordered pair of nodes $<u,v>$. Edge direction α is a function that defines a direction on each edge in E. A directed graph $\vec{G}(V,\vec{E})$ is a structure which consists of the sets of nodes V and diedges $\vec{E}$. We denote a diedge e directed from u to v by the ordered pair $e=(u,v)$. Subsequently, we denote $\vec{G}(V,\vec{E})$ by $\vec{G}^{\alpha}$, or simply $\vec{G}$ if no confusion arises where α is the edge direction for $\vec{E}$.

When a directed graph does not contain any cycle, it is called acyclic or acyclicly directed. Let $^*u=\{w|(w,u)\in\vec{E}\}$, $u^*=\{v|(u,v)\in\vec{E}\}$, for each $u\in V$. If $|^*u|=0$ (or $|u^*|=0$), then u is called a source (or a sink) of $\vec{G}$. If $\vec{G}^{\alpha}$ is acyclic, α is called acyclic directioning. Henceafter we assume the edge direction α be acyclic unless otherwise stated.

Let $\vec{G}_0$ be an initial acyclic graph and let $SO(\vec{G}_0)$ be the set of the sources in $\vec{G}_0$. Since $\vec{G}_0$ is acyclic, $SO(\vec{G}_0)\neq\varnothing$. Firing of the sources in $S_0(\subseteq SO(\vec{G}_0))$ is an event in which the direction of every diedge incident to S_0 is reversed. It is easy to show that the acyclic graph after simultaneous firing of S_0 is acyclic again. Let $f(\vec{G}_i,S_i)=\vec{G}_{i+1}$, for any i, denote the graph after firing of a source set $S_i(\subseteq SO(\vec{G}_i))$ in $\vec{G}_i$. All nodes in s_i are turned into sinks in $f(\vec{G}_i,S_i)=\vec{G}_{i+1}$. We call a sequence of $\vec{G}_i$, $i=0,1,2,3,\cdots$, such that $\vec{G}_{i+1}=f(\vec{G}_i,S_i)$, $S_i\subseteq SO(\vec{G}_i)$ an evolution trajectory of firing activity and denote it by $tr(\vec{G}_0:\sigma)\triangleq\vec{G}_0\vec{G}_1\vec{G}_2\cdots$, where $\sigma=S_0S_1S_2\cdots$. The evolution trajectory takes a different course when the source selection $S_0S_1S_2\cdots$ is made different. The set of acyclic directed graphs appearing in $tr(\vec{G}_0:\sigma)$ is denoted by $\#tr(\vec{G}_0:\sigma)$. $EV(\vec{G}_0)=\cup_{\sigma}\#tr(\vec{G}_0:\sigma)$ is called the evolution set emanated from $\vec{G}_0$.

If the firing node set s_i of each step of $\sigma=S_0S_1S_2\cdots$ uniformly equals to $SO(\vec{G}_i)$ then we say σ is in the unison firing mode(UF), otherwise in the free firing mode (FF). We should note that in the unison firing mode UF, only a single trajectory emanates from $\vec{G}_0$ but infinitely many trajectories in the free firing mode FF. When the unison firing trajectory is infinite in length, it is necessarily periodic as to be proved shortly.

Lemma 4.1 *Let us define that an evolution trajectory completes in finite length iff* $SO(\vec{G}_i)=\phi$ *for some* i. *Then any evolution trajectory emanated from an acyclic graph is infinite in length. In UF mode, the trajectory eventually falls into a cycle.*

Proof: The followings are true:

(1) An evolution trajectory $tr(\vec{G}_0:\sigma)$ initiated from acyclic graph $\vec{G}_0$ is infinite since $SO(\vec{G}_i)=\phi$ for any i.

(2) The number of acyclic graphs based on G is finite.

(3) Let $\vec{G}_i$ be an acyclic graph in the evolution set $\#tr(\vec{G}_0:\sigma)$. In UF mode, the graph $\vec{G}_{i+1}$ generated after firing of all the sources $SO(\vec{G}_i)$ in $\vec{G}_i$ is unique.

Therefore, an evolution trajectory $tr(\vec{G}_0:\sigma)$ initiated from $\vec{G}_0$ should be periodic in UF mode. Q.E.D.

4.2.2 Petri net (Marked Graph) Representation

The firing activity of an acyclic graph can be equivalently representable by that of a marked graph, a subclass of Petri nets, where each place has at most one input transition and at most one output transition. For simplicity, we use the directed graph expression of marked graphs where a node becomes firable iff all of its input edges possess at least one token. The marked graph corresponding to $\vec{G}_0$ is denoted by $MG(\hat{G},M_0)$ or MG_0 for short, where $\hat{G}$ is the directed graph and M_0 is a marking, a placement of tokens, which expresses the initial edge direction. A node of $\hat{G}$ is firable on M_0 when there is a token on every incident edge of the node. Let us show the conversion rule of $\vec{G}_0$ into MG_0: for each $e=(v,u)$ in $\vec{G}_0$, it is replaced by the parallel of paired diedges (v,u) and (u,v) in $\hat{G}$ of $MG(\hat{G},M_0)$. The direction of the diedge in $\vec{G}_0$ is translated into placement of the token. That is, the diedge from v to u in $\vec{G}_0$ is translated in $\hat{G}$ by a single token on edge (u,v) and no token on edge (v,u). In both representations, it is meant that the node v has a higher priority of firing than u, that is, v can fire before u. Therefore the source nodes in $\vec{G}_0$ are converted into the firable nodes in $MG(\hat{G},M_0)$. Fig. 4.4 summarizes the conversion rule.

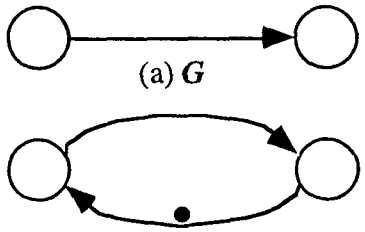


Fig. 4.4 Conversion Rule of G into MG

Note that a firing in M_0 effects a change in M_0 such that each token on all the input edges of the firing node is removed and one token is added on each of its output edges, resulting in a new marking M_1. Therefore a firing in $MG(\hat{G}, M_0)$ is an event corresponding to one step move of each enabling token across the firing node.

4.3 Concurrency Analysis of Acyclic Graph Evolution

Concurrency is the number of firable nodes in acyclic graph instance $\vec{G}_i$ of an evolution trajectory, $|SO(\vec{G}_i)|$. In this section, we discuss concurrency behaviors of firing activity in a trajectory of acyclic graph evolution driven by the source firing. Concurrency analysis is of primary importance for execution control of parallel / distributed systems. K-mutual exclusion protocols should not allow more than k nodes to be firable simultaneously at any instance. Concurrency behavior is strongly dependent on the network structure. Given (undirected) graph G and a positive number k, is it possible to find an acyclic directioning that ensure k-mutual exclusion? When k=1, we showed in Chapter 3 that if and only if there is a Hamiltonian circuit in a graph, there exists an acyclic directioning of G that ensures the concurrency of the evolution be exactly unity. To extend this result for case $k \geq 2$, we introduced in preliminary work [50] a notion of canonical circuit-cover and show that an initial acyclic graph can be set up for a given graph G to ensure the concurrency below k if there exists canonical circuit-cover of cardinality k for G. We will show that

the decision problem of finding any canonical circuit-cover is NP-complete.

The main analysis in this section is for the upper bound of concurrency, but we also treat the lower bound. We are interested in the lower bound from graph theoretical point of views. In concrete, we discuss in this section the following firing concurrency bounds of a trajectory of acyclic graph evolution in FF mode:

$\max_{\vec{G}_i \in EV(\vec{G}_0)} |SO(\vec{G}_i)|$ = The Upper bound of Concurrency in $EV(\vec{G}_0)$, $U_{FC}(\vec{G}_0)$,

$\min_{\vec{G}_i \in EV(\vec{G}_0)} |SO(\vec{G}_i)|$ = The Lower bound of Concurrency in $EV(\vec{G}_0)$, $L_{FC}(\vec{G}_0)$.

Definition 4.1 A circuit of length $k(\geq 2)$ in a graph G, $c=(V_c, E_c)$, $|V_c|=|E_c|=k$, is a subgraph of G such that there is a path on c between any two nodes in V_c, and, for each node v_i in V_c, the number of its incident edges in c(but not in G) exactly equals 2. c oriented by β, written as (c,β), is a cyclic permutation of the nodes and the edges of c in whose order a path goes through the nodes and the edges in c. There are two orientations, clockwise and anti-clockwise.

In order for Chandy-Misra type 1-mutual exclusion to be feasible, the graph G must possess a Hamiltonian circuit. Moreover, we introduce a concept "circuit-cover", for $k(\geq 2)$-mutual exclusion, which requires that each node is contained in a circuit. These suggest an assumption of G that every node in G belongs to some circuit. In this chapter we treat only a connected graph. Therefore we consider a connected graph G such that every node in G belongs to some circuit. We call such a graph "circuit-connected graph". The assumption will simplify a discussion, theorem statements, and their proofs. Great deal this situation is particularly the case when $k \geq 2$. Henceafter, we concentrate our discussion on circuit-connected graphs unless explicitly stated otherwise.

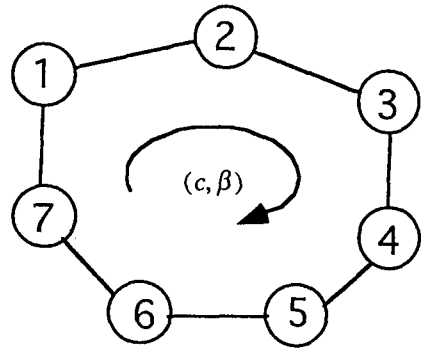
4.3.1 Upper bound of Firing Concurrency in the FF mode

We first consider the upper bound of firing concurrency in the FF mode. We introduced a notion of canonical circuit-cover[50]. Based on this notion the initial acyclic direction can be constructed such that the

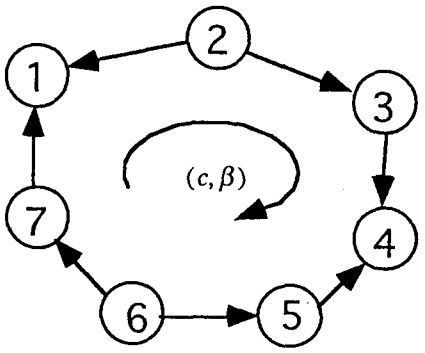
maximum concurrency is bounded by the cardinality of the circuit-cover. Before describing this notion, we introduce some additional notations as necessity arises, but liberally use basic graph theoretical terms.

We distinguish term "direction" from "orientation". The term "direction" is used for "edge" as "diedge", on the other hand, "orientation" is for "circuit" as "oriented circuit". But we have the term, "dicircuit" composed of a sequence of diedges in the same orientation. Since the orientation of an oriented circuit has no relation with a direction on the edge in the circuit, we have to distinguish "oriented circuit" from "dicircuit".

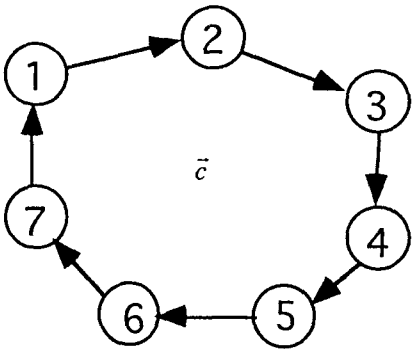
Example 4.2: Fig. 4.5 shows examples of an oriented circuit, a dicircuit.



(a) Oriented Circuit (c,β) 1-2-3-4-5-6-7 on Undirected Graph



(b) Oriented Circuit (c, β) 1-2-3-4-5-6-7 on Directed graph



(c) Dicircuit 1-2-3-4-5-6-7

Fig. 4.5 Example 2: Oriented Circuit and Dicircuit

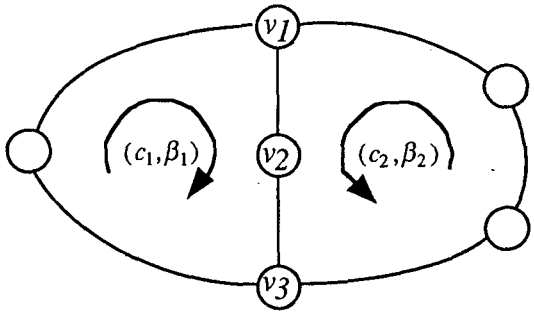


Fig. 4.6 Example 4.3:"compatible"
(c_1,β_1) and (c_2,β_2) are compatible.

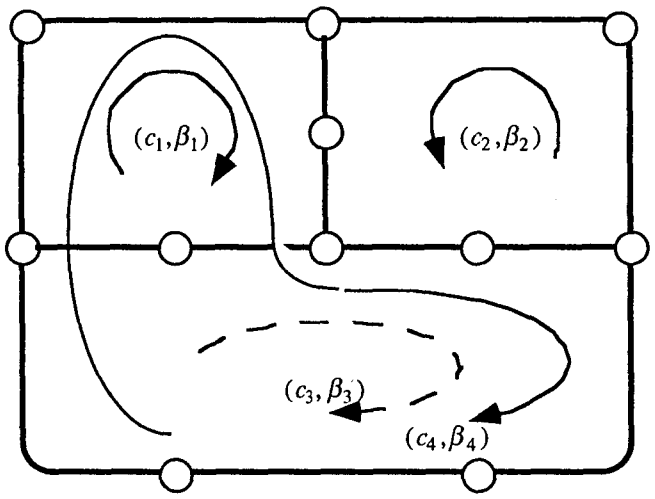


Fig. 4.7 Example 4.4: A set of compatible oriented circuits.
$\{(c_1,\beta_1), (c_2,\beta_2), (c_3,\beta_3)\}$ is not compatible but
$\{(c_1,\beta_1), (c_2,\beta_2), (c_4,\beta_4)\}$ is.

Definition 4.2 Two oriented circuits, (c_1,β_1) and (c_2,β_2) are said compatible if the ordering of the nodes appearing in both c_1 and c_2 in (c_1,β_1) is the same as that in (c_2,β_2).

Example 4.3: In Fig. 4.6, (c_1,β_1) and (c_2,β_2) are compatible.

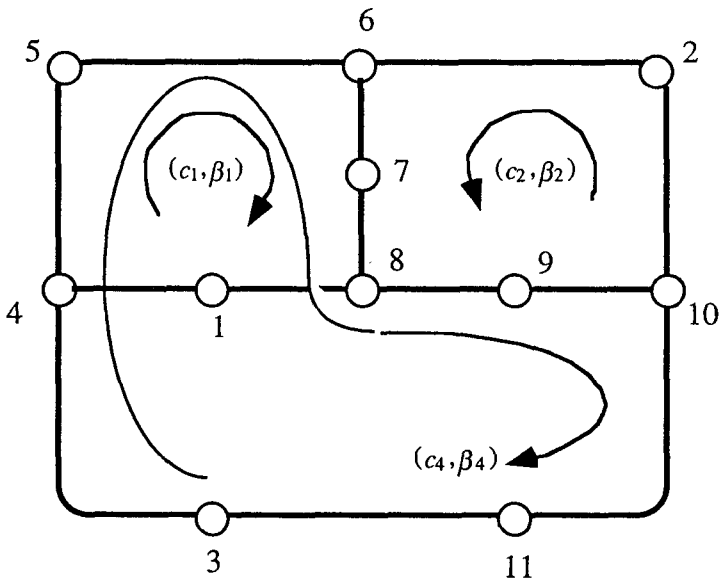
Definition 4.3 Let $\Omega^{\beta}=\{(c_1,\beta_1),(c_2,\beta_2),\cdots(c_k,\beta_k)\}$ be a set of oriented circuits of undirected graph G, we say that the orientation, $\beta=\{\beta_1,\beta_2,\cdots,\beta_k\}$, is compatible if any pair (c_i,β_i) and (c_j,β_j) from Ω^{β} is compatible.

Example 4.4: In Fig. 4.7, $\{(c_1,\beta_1),(c_2,\beta_2),(c_3,\beta_3)\}$ is not compatible but $\{(c_1,\beta_1),(c_2,\beta_2),(c_4,\beta_4)\}$ is compatible.

Definition 4.4 Suppose graph G be circuit-connected. A set of circuits of G, $\Omega=\{c_1,c_2,\cdots c_k\}$, is said a circuit-cover of G if any node v of G is contained in at least one circuit from Ω but no proper subset of Ω has this property.

Definition 4.5 A node numbering η that assigns a unique positive integer to each node of G is said compatible with an oriented circuit-cover $\Omega^{\beta}=\{(c_1,\beta_1),(c_2,\beta_2),\cdots(c_k,\beta_k)\}$ if sequence $(\eta(v_1),\eta(v_2),\cdots,\eta(v_r))$ assigned to any c in Ω, $c=\{v_1,v_2,\cdots,v_r\}$ is cyclically monotone, i.e., for some cyclic permutation q of $c=\{v_1,v_2,\cdots,v_r\}$, $\eta(q(v_1))<\eta(q(v_2))<\cdots<\eta(q(v_r))$ holds.

Example 4.5: In Fig. 4.8, the numbering is compatible with (c_1,β_1), (c_2,β_2), and (c_4,β_4).



(c_1,β_1): 1, 4, 5, 6, 7, 8

(c_2,β_2): 2, 6, 7, 8, 9, 10

(c_4,β_4): 3, 4, 5, 6, 7, 8, 9, 10, 11

Fig.4.8 Example 4.5: Canonical Node-Numbering.

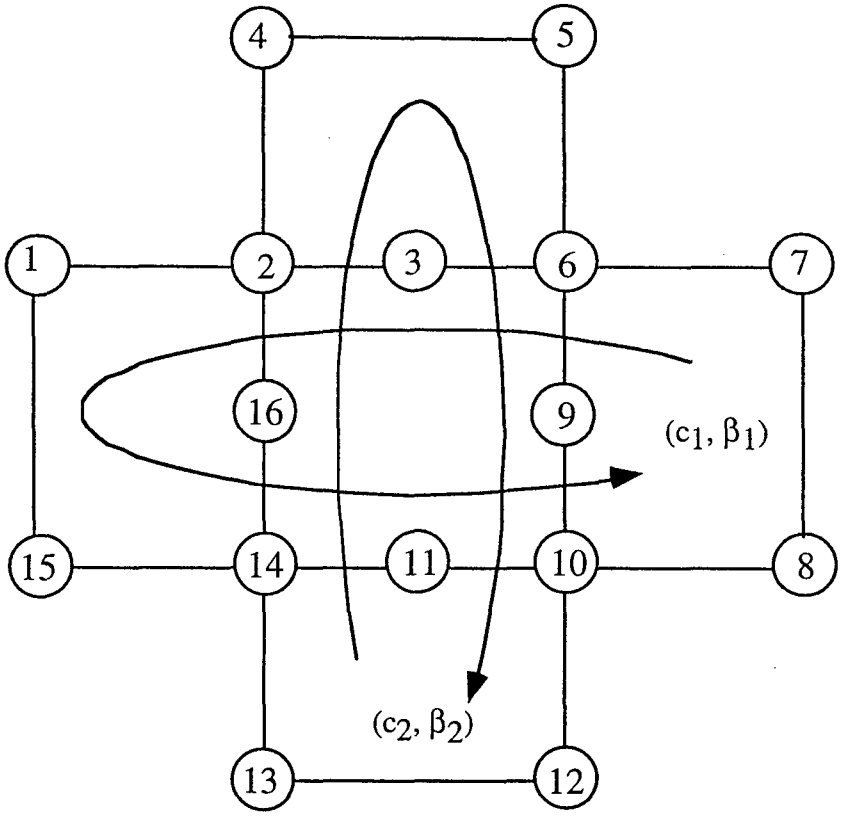


Fig. 4.9 Example 6: Circuit-cover $\{(c_1, \beta_1), (c_2, \beta_2)\}$ is not canonical

Definition 4.6 Suppose G be circuit-connected. Let $\Omega = \{c_1, c_2, \cdots c_k\}$ be a circuit-cover and its orientation $\Omega^{\beta} = \{(c_1, \beta_1), (c_2, \beta_2), \cdots (c_k, \beta_k)\}$ be compatible. An edge may or may not be contained in c_i in Ω. If an edge is not be contained in any c_i, remove it from G. Let $G(c_1 c_2 \cdots c_k)$ be the graph generated by this removal. Each edge in $G(c_1 c_2 \cdots c_k)$ is made uniquely directed by the compatible orientation of Ω^{β}. Now (c_i, β_i) becomes a dicircuit $\vec{c}_i$. The result is a directed graph denoted by $\vec{G}(\vec{c}_1 \vec{c}_2 \cdots \vec{c}_k)$.

Definition 4.7 Let $\Omega = \{c_1, c_2, \cdots c_k\}$ be a circuit-cover and its orientation Ω^{β} be compatible. Ω^{β} is said "canonical" if there exists an acyclic graph $\vec{A}$ which is made by removal of a certain edge $\hat{e}_i = (\hat{u}_i, \hat{v}_i)$ from each dicircuit $\vec{c}$ of $\vec{G}(\vec{c}_1 \vec{c}_2 \cdots \vec{c}_k)$, $\vec{A} = \vec{G}(\vec{c}_1 \vec{c}_2 \cdots \vec{c}_k) - \{\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_k\}$. We denote a path $\vec{c}_i - \hat{e}_i$ by $(v_{i_0} = \hat{v}_i, v_{i_1}, v_{i_2}, \cdots, v_{i_r} = \hat{u}_i)$, where v_{i_0} is

the head node of the orientation. We call the edge set, $\{\hat{e}_1,\hat{e}_2,...,\hat{e}_k\}$ canonical feed back edge set.

Example 4.6 For a graph shown in Fig. 4.9, a circuit-cover is not canonical since there does not exist an acyclic graph $\vec{A}$ as defined in Definition 4.7.

Now we show two lemmas before a main theorem.

Lemma 4.2 *Let* G *be a circuit-connected graph. If* G *possesses a canonical circuit-cover,* $\Omega^{\beta}=\{(c_1,\beta_1),(c_2,\beta_2),\cdots(c_k,\beta_k)\}$, *then there exists a node-numbering scheme* $\hat{\eta}$ *of* G, *compatible with* Ω^{β}.

Proof: We construct a particular node-numbering $\hat{\eta}$, that is known as the topological node number of acyclic graphs, step by step as follows:

1° Consider $\vec{A}$ as defined in Definition 4.7. Initially all nodes are unassigned. Let $\vec{A}$ be $\vec{A}_0$ and $i=0$.

2° Find the set of sources of $\vec{A}_i$, SO_i. Assign remaining young positive integers in any manner to each node in SO_i. $\vec{A}_{i+1}:=\vec{A}_i-SO_i$ and stop if $\vec{A}_{i+1}$ is null.

3° return to 2° with $i:=i+1$.

We prove the node-number $\hat{\eta}$ so constructed is indeed desired one. Let $\hat{e}_i=(\hat{u}_i,\hat{v}_i)$, $\vec{c}_i-\hat{e}_i=(v_{i_0}=\hat{v}_i,v_{i_1},v_{i_2},\cdots,v_{i_r}=\hat{u}_i)$ be node expression where arrangement $(v_{i_0}=\hat{v}_i,v_{i_1},v_{i_2},\cdots,v_{i_r}=\hat{u}_i)$ in accord with c_i's orientation β_i constitutes a dipath. All we have to show is monotonicity,

$$\hat{\eta}(v_{i_0})<\hat{\eta}(v_{i_1})<\cdots<\hat{\eta}(v_{i_r}).$$

The inequality comes from well-known property of the topological node numbering: It is clear from the above construction that $v_{i_0}\in SO_{i_0}$, $v_{i_1}\in SO_{i_1}$, ..., $v_{i_{r-1}}\in SO_{i_{r-1}}$, $v_{i_r}\in SO_{i_r}$ and index inequality $i_0<i_1<\cdots<i_r$ hold. Since the younger node number is assigned to the younger indexed source set, the index inequality implies the desired monotonicity. Q.E.D.

Lemma 4.3 *Let* $\Omega^{\beta}=\{(c_1,\beta_1),(c_2,\beta_2),\cdots(c_k,\beta_k)\}$ *be a canonical circuit-cover of circuit-connected* G *and let* $\hat{\eta}$ *be a compatible node-numbering as constructed in Lemma 4.2. Introduce to each edge* $e=<u,v>$ *of* G *direction from* u *to* v *if* $\hat{\eta}(u)<\hat{\eta}(v)$ *holds. Then the resulting directed graph* $\vec{G}$ *is acyclic.*

Proof: Once a unique node-numbering is introduced and the edge direction is defined as in the lemma, a dicircuit (or a cycle) can not appear in $\vec{G}$. Suppose $\vec{c}=(v_1,v_2,\cdots,v_k)$ be a cycle. Then it demands $\hat{\eta}(v_1)<\hat{\eta}(v_2)<\cdots<\hat{\eta}(v_k)$ and $\hat{\eta}(v_k)<\hat{\eta}(v_1)$, which is self contradictory. Q.E.D.

Definition 4.8 The node-numbering $\hat{\eta}$ so constructed in Lemma 4.2 is said canonical, and the acyclic graph $\vec{G}$ so generated by $\hat{\eta}$ is also said "canonical".

Theorem 4.1 *Suppose circuit-connected G possess a "canonical" circuit-cover of the compatible orientation, $\Omega^{\beta}=\{(c_1,\beta_1),(c_2,\beta_2),\cdots(c_k,\beta_k)\}$. Then we can design an acyclic graph $\vec{G}_0$ such that the maximum number of firable nodes at any instance of the acyclic graph evolution starting from $\vec{G}_0$ is at most k, under any free firing mode. That is $U_{FC}(\vec{G})=k$. Moreover, unless there exists a feed back edge set, $\{\hat{e}_1,\hat{e}_2,\ldots,\hat{e}_k\}$, in $\vec{G}(\vec{c}_1\vec{c}_2\cdots\vec{c}_k)$ such that the end node $\hat{v}_i$ of each edge $\hat{e}_i$ is unique, the concurrency cannot be k by any firing schedule.*

Proof: By Lemma 4.2, we have a compatible node-numbering $\hat{\eta}$. Let $\vec{G}_0$ be the canonical acyclic direction of G induced by $\hat{\eta}$ as constructed in Lemma 4.3. Let $\vec{G}_j$ be the j-th acyclic graph of a given free firing evolution.

Let $\hat{e}_i=(\hat{u}_i,\hat{v}_i)$, $\vec{c}_i-\hat{e}_i=(v_{i_0}=\hat{v}_i,v_{i_1},v_{i_2},\cdots,v_{i_r}=\hat{u}_i)$ be node expression where arrangement $(v_{i_0}=\hat{v}_i,v_{i_1},v_{i_2},\cdots,v_{i_r}=\hat{u}_i)$ in accord with c_i's orientation β_i constitutes a dipath. As shown in Lemma 4.2, $\hat{\eta}(v_{i_0})<\hat{\eta}(v_{i_1})<\cdots<\hat{\eta}(v_{i_r})$ holds. Thus there exists a single source node $\hat{v}_i$ in each oriented circuit in Ω^{β} when $\vec{G}_0$ is generated. Moreover, only single edge, that is, $\hat{e}_i$ in an oriented circuit c_i has the direction opposite to the orientation β_i in $\vec{G}_0$ and $\hat{e}_i$ is connected to the source node $\hat{v}_i$ of oriented circuit c_i. By firing of the source of c_i, the two incident edges in c_i reverse their directions. Then only single edge in the circuit c_i has the direction opposite to β_i in $\vec{G}_0$. This means there is only one source node in each circuit c_i at any time. Since node v cannot be firable in $\vec{G}_j$ unless v is a source simultaneously with respect to individual oriented circuit in Ω^{β} that covers v, the number of the

sources in $\vec{G}_j$ is necessarily not larger than the cardinality of Ω^{β}, that is, k.

Suppose there exists no canonical feed back edge set, $\{\hat{e}_1,\hat{e}_2,...,\hat{e}_k\}$, in $\vec{G}(\vec{c}_1\vec{c}_2\cdots\vec{c}_k)$ such that the end node $\hat{v}_i$ of each edge $\hat{e}_i$ is unique. Then there is no initial acyclic graph with k source nodes, which means that no acyclic graph reachable from an initial acyclic graph has k source nodes. Q.E.D.

Theorem 4.1 says that if we find a canonical circuit-cover of a given graph with the feed back edge set, we can set up an initial acyclic graph such that the concurrency of the evolution trajectory is bounded by the cardinality of the circuit-cover k.

Example 4.7: Fig. 4.10 shows a process of canonical acyclic direction.

The next theorem gives a method to find a circuit-cover, say the depth first search.

Theorem 4.2 *Let $G(V,E)$ be a circuit-connected graph. The maximum cardinality of canonical circuit-covers is $|E|-|V|+1$, that is, $U_{FC}(\vec{G}_0)\leq |E|-|V|+1$, where $\vec{G}_0$ is generated by the canonical numbering as stated in Lemma 4.3. By the Depth First Search (DFS) we can construct a initial acyclic graph $\vec{G}_0$ with $U_{FC}(\vec{G}_0)\leq |E|-|V|+1$ in $O(|V|+|E|)$ time.*

Proof: We show that at most $|E|-|V|+1$ circuits can cover all the edges. Let $|E|-|V|+1=k$.Then these k circuits can cover all the nodes. Suppose at least $h(>k)$ circuits $C=\{c_1,c_2,\cdots,c_h\}$ are needed for covering all the edges in G. Then, there exists an edge e_i, $i=1,2,\cdots,h$, not contained in any circuit $c_j, j\neq i$. Now remove such edges from G. Let $C'=\{c_{h+1},c_{h+2},\cdots,c_{h+r}\}$ be the set of the remaining circuits of the remaining graph G'. If $C'\neq\varnothing$, remove edges so that the graph after the removal become a tree of G. All the removal edges must compose the set of chords for the tree. However, the cardinality of a chord set is k.We have now contradiction.

By DFS we can find a fundamental circuit set $FC=\{c_1,c_2,...,c_k\}$ in which each visited node is numbered a positive integer in ascending order. Note that the numbering is compatible. A subset of FC, Ω, is a circuit-cover of G, that is, every node is contained in some circuit of

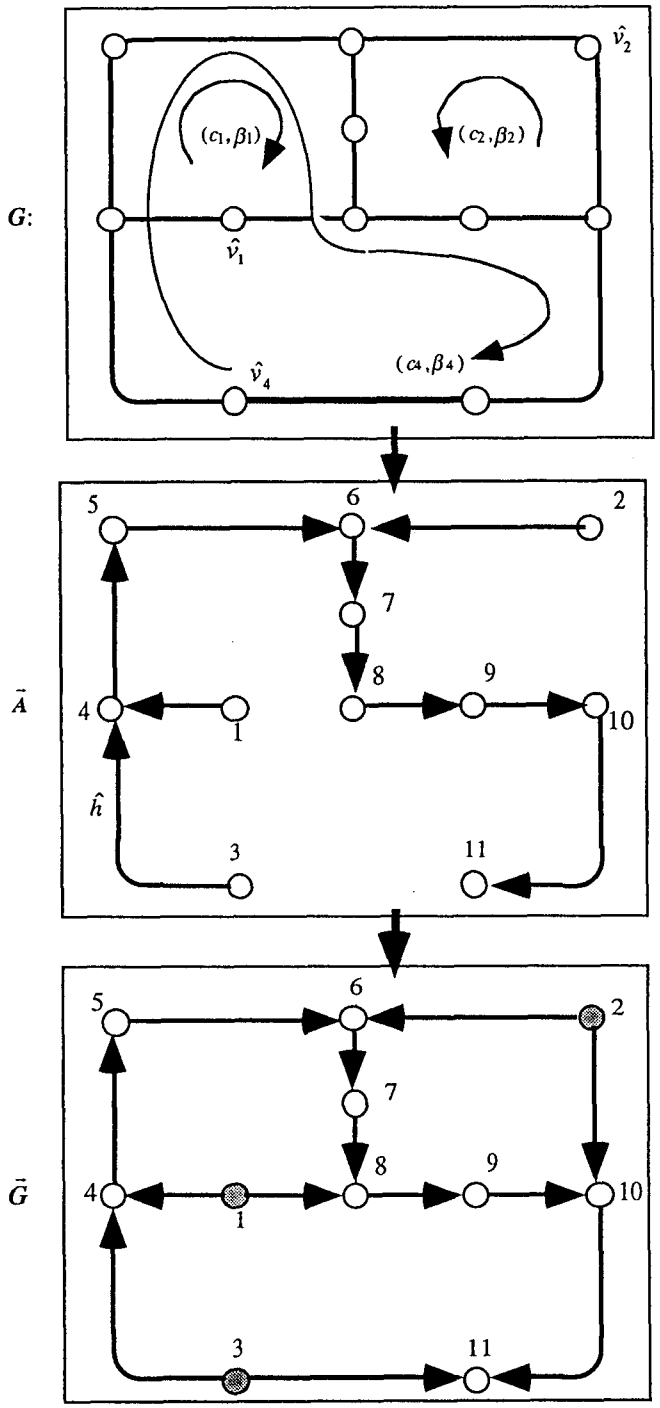


Fig. 4.10 Example 4.7: Process of Canonical Acyclic Direction
● : Source Nodes.

Ω. Let $\Omega = \{c_{i_1}, c_{i_2}, \ldots, c_{i_h}\}$ and let $\Omega^{\beta} = \{(c_1, \beta_1), (c_2, \beta_2), \cdots (c_k, \beta_k)\}$ be a canonical circuit-cover in which each orientation β_{i_j} is the same as the ascending order of the node numbers in c_{i_j}. By removal of the chords, found in the DFS, we can obtain a tree, that is an acyclic graph. Therefore Ω is a canonical circuit-cover and its size below $|E|-|V|+1$. Note that we do not need to construct Ω from FC so as to set up the initial acyclic graph $\vec{G}_0$ since the numbering is canonical. Therefore we can obtain $\vec{G}_0$ from G in $O(|V|+|E|)$ time, which is the time complexity for DFS[29], since the edge directioning based on the numbering requires only $O(|E|)$. Q.E.D.

For a given graph, the problem to find a canonical circuit-cover with the minimum size is of prime importance but its computational complexity is suspected NP-hard. We define the following decision problem CC(k) which associates with just circuit-cover but not canonical one and show its NP-completeness.

Definition 4.9 A decision problem of Circuit-Cover, CC(k), is defined as follows:

Instance: A circuit-connected graph G, a positive integer k.

Question: For a given graph, is there a circuit set in which every node belongs to at least one circuit from the set whose cardinality is not greater than k ?

By the above definition we can easily know CC(1) is Hamiltonian Circuit problem, HC.

Theorem 4.3 *The decision problem CC(k) is NP-complete for a circuit-connected graph.*

Proof: It is easy to see that CC(k) $\in$ NP, since a nondeterministic algorithm needs only to guess a subset V_i in which $\cup V_i = V$ and an ordering in each subset and check that all the required edges connecting adjacent nodes belong to the edge set E in polynomial time. Let us transform HC to CC(k). Let $G(V,E)$ be an arbitrary instance of HC. We will define the instance of CC(k), $G'(V',E')$. We prepare k isomorphic graphs of $G(V,E)$, $G_1, G_2, \cdots, G_k$. Choose two nodes each from the graphs $G_i, 2 \le i \le k-1$, denote the nodes by v_i^+ and v_i^-, and one node each from the graphs G_i, $i = 1, k$, v_1^- and v_k^+. Now we define

the instance of CC(k), $G'(V',E')$, connecting all the graphs G_i, $1 \le i \le k$ by identifying v_i^- with v_{i+1}^+, for $1 \le i \le k-1$. An instance of CC(k) is generated in polynomial time since k is a constant. If there exists a Hamiltonian circuit c in G, then each G_i has a Hamiltonian circuit c_i. Therefore $G'(V',E')$ has a circuit-cover $\Omega = \{c_1, c_2, \ldots, c_k\}$ where $|\Omega|=k$. Conversely, suppose $G'(V',E')$ has a circuit-cover $\Omega = \{c_1, c_2, \ldots, c_{k'}\}$ where $|\Omega|=k' \le k$. Since $k' \ge 1$ and $G'(V',E')$ has k-1 cut nodes and no circuit across G_i and G_{i+1} is possible, $|\Omega|=k$ and each G_i has a Hamiltonian circuit c_i. Therefore G has a Hamiltonian circuit. Q.E.D.

By Theorem 4.3 it is difficult to find a circuit-cover with the desired cardinality. It should mean that the problem for canonical circuit-cover is also difficult.

4.3.2 Lower Bound of Firing Concurrency in FF mode

Knowing an upper bound of firing concurrency, our interest now shift to its lower bound. We show a theorem which states $L_{FC}(\vec{G}_0)=1$. The theorem is proven by using properties of marked graphs. We need some preparation to state it.

The shortest token-distance matrix of a marked graph MG is an $n \times n$ symmetric matrix $D=[d_{ij}]$ defined by:

$$d_{ij} = \begin{cases} \min M(\vec{P}_{ij}): \textit{if directed path } \vec{P}_{ij} \textit{ exists from node } i \textit{ to node } j. \\ \infty \ : \textit{otherwise}, \\ 0 \ : \textit{if } i=j \end{cases}$$

where $M(\vec{P}_{ij})$ is the total number of tokens on dipath $\vec{P}_{ij}$ in marking M. Then we have the following lemma by [31].

Lemma 4.4[31] *A node j is firable under a marking M if and only if every element of the j-th column of $D(M)$, except its diagonal, is positive.*

When node i fires, the shortest token-distance matrix of $MG(\hat{G},M)$ changes itself as follows: Let node i be fired in marking M, whose shortest token-distance matrix is $D(M)$. Then the new distance matrix $D(M')$ after i's firing is obtained by:

(1) subtract 1 from each element of the i-th column of $D(M)$.

(2) add 1 to each element of the i-th row of $\boldsymbol{D}(\boldsymbol{M})$.

Theorem 4.4 *Let G be circuit-connected and let $\vec{G}_0$ be any of its initial acyclic direction. Then, $L_{FC}(\vec{G}_0)=1$.*

Proof: Let $\boldsymbol{MG}_0$ be the marked graph expression of $\vec{G}_0$. Since $\boldsymbol{MG}_0$ is strongly connected, the shortest token distance matrix of $\boldsymbol{MG}_0$ does not include any infinite element. Let $|SO(\boldsymbol{MG})|$ be the set of firable nodes of $\boldsymbol{MG}$.

i) Suppose that $|SO(\boldsymbol{MG})|=k(\geq 2)$. Choose arbitrarily one node s among k firable sources. Keep firing other firable nodes except s, then the firing eventually comes to stop. To prove this, let $\boldsymbol{M}'$ be any marking reachable from the initial marking $\boldsymbol{M}_0$. Then $s \in SO(\boldsymbol{MG}')$ holds, since the directions of all the incident edges of s are never changed. If $SO(\boldsymbol{MG}')$ contains node v other than s, continue firing v. Since firing v reduces all elements of v-th column of $\boldsymbol{D}$ matrix by one, eventually d_{si} for all i will be reduce to zero. Then, s becomes the only source.

ii) Suppose that $SO(MG_k)=\phi$ for some $\boldsymbol{MG}'$, that is, there is no firable node in $\boldsymbol{MG}'$. This is contradiction to the fact that $\boldsymbol{MG}_0$ is live. Therefore, $L_{FC}(\vec{G}_0)=1$. Q.E.D.

4.4 Method to Design Initial Acyclic Graph

In this section, we propose a method for design of initial acyclic graph for desired concurrency. The method proposed in this section is composed of two steps generally, to find a compatible circuit-cover, $\Omega^{\beta}=\{(c_1,\beta_1), \cdots(c_k,\beta_k)\}$, and to find a feedback edge set, $\boldsymbol{F}$, $|\boldsymbol{F}|\leq k$. Since the problem to find a (compatible) circuit-cover is NP-hard, there must not exist deterministic polynomial time algorithms for it. Therefore we propose, in Section 4.4.1, a method applying genetic algorithms to find a compatible circuit-cover.

For a general directed graph, the problem to find a feedback edge set with the desired cardinality is NP-hard. However, for directed graphs based on compatible circuit-covers generated in the step 1, this problem is not NP-hard. We show, in Section 4.4.2, we have a polynomial deterministic algorithm for the problem.

4.4.1 Method to Find Compatible Circuit-Cover

In this subsection, we propose a method to find a compatible circuit-cover applying genetic algorithms. We explain our method by dividing into the following five steps of genetic algorithms.

(1) Coding of Circuit-covers

Using the GA method, we have to first decide how to map circuit-covers in searching space to chromosomes in GA space, that is, coding. We describe coding here.

Circuit-covers are represented by two tables, the node table and the edge table in a GA space. A row vector of the node table corresponds to one circuit. A circuit-cover is composed of all the row vectors. Each entry of a row, corresponding to a node in the graph, shows the order of the node in the corresponding circuit. We can construct a circuit-cover by tracing the orders. The edge table is prepared to check efficiently the compatibility between two circuits. A row vector of the edge table also corresponds to one circuit. Each entry of a row, corresponding to an edge in the graph, has one value of 0, 1, -1. It depends on whether the edge is directed from the smaller of the node identifier to the other, or opposite direction, or not included in the circuit, where the comparison of node identifiers is based on the alphabetic order.

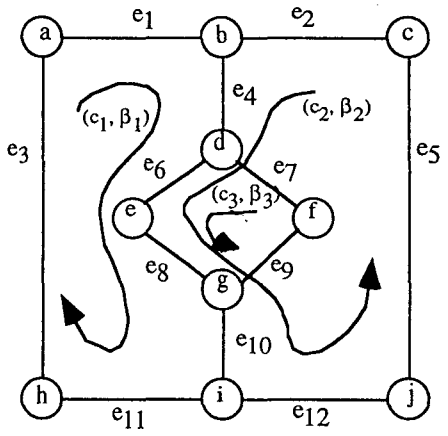
Example 4.8: Examples of coding for a given graph are shown in Fig. 4.11.

(2) Initial Population

In the previous section, we showed that by the Depth First Search (DFS) we can construct an initial acyclic graph $\vec{G}_0$ with the cardinality no more than $|E|-|V|+1$ in $O(|V|+|E|)$ time. Therefore, by one execution of the DFS, we can obtain one compatible circuit-cover with the cardinality no more than $|E|-|V|+1$. We generate, as the initial population, compatible circuit-covers of the number of the population size by the DFS in which it chooses randomly one, if multiple choices, at each node.

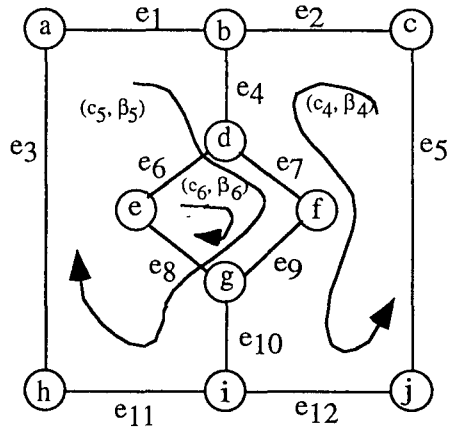
«Generating Initial Population»

1° By the DFS, number to each node of a given graph.



	a	b	c	d	e	f	g	h	i	j
c1	1	2		3	4		5	7	6	
c2		1	7	2	3		4		5	6
c3				1	2	4	3			

(a) Compatible Circuit-cover C_1 & Its node table.



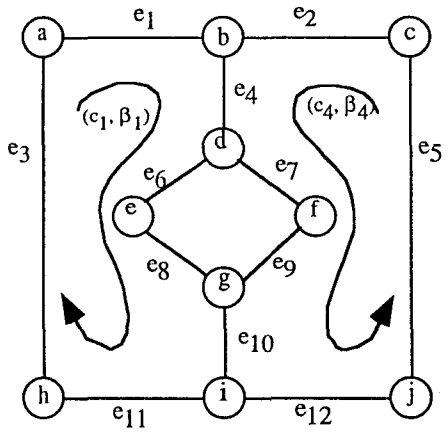
	a	b	c	d	e	f	g	h	i	j
c4		2	1	3		4	5		6	7
c5	1	2		3		4	5	7	6	
c6				1	4	2	3			

(b) Compatible Circuit-cover C_2 & Its node table.

Fig. 4.11 Examples of Compatible Circuit-covers and Node Tables

2° Make the node table and the edge table with respect to the numbering at 1°.

3° Return 1° if less than the number of the population size.



	a	b	c	d	e	f	g	h	i	j
c1	1	2		3	4		5	7	6	
c4		2	1	3		4	5		6	7

Fig. 4.12 Compatible Circuit-cover C_3 & Its node table.
Crossover result between C_1 and C_2 in Fig. 4.11.

(3) Crossover

The crossover operation is as follows:

«Crossover»

1° From the GA space, choose randomly two chromosomes. Let us denote them by P_1, P_2.

2° From P_1, choose randomly one row vector, that is, one circuit. Let us denote it by c_1.

3° Collect from P_2 all the circuits that are compatible with c_1.

4° Collect from P_1 all the circuits that are compatible with the circuits collected at 3°.

5° Let c_1 and the collected circuits at 3° and 4° be in the set s.

6° Construct a compatible circuit-cover c_1 from s.

7° Change the roles between P_1 and P_2 and similarly construct c_2.

Example 4.9: Fig. 4.12 is a result of the crossover operation between two chromosomes in Fig. 4.11(a), (b).

(4) Mutation

The mutation operation is as follows:

«Mutation»

1° From the GA space, choose randomly one chromosome. Let us denote them by P.

2° Reverse the direction of every edge.

(5) Fitness Function

We use the following function as the fitness function:

$$f(C) = |E| - |V| + 1 - \textit{the number of the circuits}.$$

By the proposed method we can obtain compatible circuit-covers $\Omega^{\beta} = \{(c_1, \beta_1), (c_2, \beta_2), \cdots (c_k, \beta_k)\}$ for a given graph $G(V, E)$. Then we construct the directed graph $\vec{G}(\vec{c}_1 \vec{c}_2 \cdots \vec{c}_k)$ from Ω^{β} and $G(V, E)$ (See Definition 4.6).

4.4.2 Method to Find Feedback Edge Sets

For a general directed graph, the problem to find a minimum feedback edge set is known to be NP-hard[23]. However, for planar graphs the problem is solvable in polynomial time. In this section we show, for a directed graph based on compatible circuit-covers, the problem is in the class P though not planar graphs, that is, there are polynomial time algorithms.

Theorem 4.5 *For a directed graph based on a compatible circuit-cover with its cardinality k circuits, the problem to find a feedback edge set with the cardinality k is in P.*

Proof: There are at least k circuits on the directed graph. A feedback edge set should include an edge in each circuit of the circuit-cover. All we have to do is to check a set that includes just one edge from each circuit. The number of such sets is less than $|E|^k$. Since k is a constant, we can solve the problem in polynomial time. Q.E.D.

For a directed graph based on a compatible circuit-cover, we can find, in polynomial time, a feedback edge set with the cardinality k if exists. Unless there is a feedback edge set with the cardinality k in the directed graph, we can know the fact in polynomial time also. The later case requires to try to find a feedback edge set again for another directed graph. Therefore, the first step of our method should generate more than one directed graphs based on compatible circuit-covers. The GA method of the first step satisfies the requirement.

4.5 Mutual Exclusion for Multiple Shared Resource Cases

In Chapter 3, we have proposed a mutual exclusion protocol for a single resource case, Distributed MUTEX. In this section, we extend the protocol to multiple shared resource cases using the results in Section 4.3 and 4.4.

[Mutual Exclusion Problems] Let us consider a network environment $N=(G,R,U)$, where $G=(V,E)$ represents the network structure, R the set of the shared resources, and U the resource requirement list. Then, design an initial acyclic graph such that the Distributed MUTEX works well under $N=(G,R,U)$.

In this section, we consider the cases of the shared resources, (1) the single kind and the single quantity (given in Chapter 3), (2) the single kind and the multiple quantity, and (3) the multiple kinds and the single quantity.

(1) Single Kind and Single Quantity

This case is represented by $R=\{r\}$, $|r|=1$, $U=\{U_r\}$. Without losing generality, we assume that every node uses the resource r, that is, $U_r=V$. The solution is given as a corollary of Theorem 3.1.

Corollary 4.1 *Let us consider a network* $N=(G,R,U)$, $R=\{r\}$, $|r|=1$, $U=\{U_r\}$, $U_r=V$. *We can have an initial acyclic graph such that the Distributed MUTEX works correctly under* $N=(G,R,U)$, *if and only if there exists a Hamiltonian circuit on* $G=(V,E)$.

(2) Single Kind and Multiple Quantity

This case is represented by $R=\{r\}$, $|r|=k(\geq 2)$, $U=\{U_r\}$. Without losing generality, we assume that every node uses the resource r, that is, $U_r=V$. The solution is given as a corollary of Theorem 4.1.

Corollary 4.2 *Let us consider a network* $N=(G,R,U)$, $R=\{r\}$, $|r|=k(\geq 2)$, $U=\{U_r\}$, $U_r=V$. *We can have an initial acyclic graph such that the Distributed MUTEX works correctly under* $N=(G,R,U)$, *if there exists a canonical circuit-cover with the cardinality* k, $\Omega^{\beta}=\{(c_1,\beta_1),(c_2,\beta_2),\cdots(c_k,\beta_k)\}$ *on* $G=(V,E)$.

(3) Multiple Kinds and Single Quantity

This case is represented by $\boldsymbol{R}=\{r_1,r_2,...,r_h\}$, $\boldsymbol{U}=\{U_{r_i}|r_i\in\boldsymbol{R}\}$, $|r_i|=1$, $U_{r_i}\subseteq\boldsymbol{V}$, $i=1,2,...,h$. The solution is given as a corollary of Theorem 4.1.

Corollary 4.3 *Let us consider a network* $\boldsymbol{N}=(\boldsymbol{G},\boldsymbol{R},\boldsymbol{U})$, $\boldsymbol{R}=\{r_1,r_2,...,r_h\}$, $\boldsymbol{U}=\{U_{r_i}|r_i\in\boldsymbol{R}\}$, $|r_i|=1$, $U_{r_i}\subseteq\boldsymbol{V}$, $i=1,2,...,h$. *We can have an initial acyclic graph such that the Distributed MUTEX works correctly under* $\boldsymbol{N}=(\boldsymbol{G},\boldsymbol{R},\boldsymbol{U})$, *if there exists a canonical circuit-cover with the cardinality k,* $\Omega^{\beta}=\{(c_1,\beta_1),(c_2,\beta_2),\cdots(c_k,\beta_k)\}$ *on* $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$, *where each oriented circuit* (c_i,β_i) *is composed of the nodes in* U_{r_i} *and the edges needed for the circuit.*

4.6 Concluding Remarks

We have analyzed concurrency of an acyclic graph evolution from graph theoretical point of views, and investigated topological conditions for assuring the number of source nodes less than or equal to k using a new concept, canonical circuit-cover. We have showed that CC(k), "For a given graph, is there a circuit set in which every node belongs to at least one circuit from the set whose cardinality is not greater than k ?", is NP-complete, proposed a method to find a canonical circuit-cover using a genetic algorithm. Moreover, we have extended the Distributed MUTEX to the multiple resource case.

Chapter 5

Adaptation of a Genetic Algorithm to Gender-fair Stable Marriage Problems

We consider the stable marriage problem. Gale and Shapley originally proposed this problem in which two groups to be matched are represented by a men group and a women group. Real applications range over matching software processes to CPUs, matching autonomous robots to battery charger devices, and matching members of two sets of autonomous robots to each other for pairing up in a cooperative work. In each case, preference lists may be constructed according to environment parameters such as cost, distance, capacity, and efficiency. The stability is evaluated by monitoring various properties of the environment.

In this chapter, we consider a gender-fair matching in the stable marriage problem. The gender-fair stable matching defined in this dissertation has the property that the sum of partners' ranks for individual men in their preference lists is as close as possible to the sum of partners' ranks for individual women in their preference lists. The gender-fair stable marriage problem is one of the important open problems among stable marriage problems. The main result consists in a procedure constructed for applying a genetic algorithm (GA) to the gender-fair problem, in which the gender-fair marriage problem is transformed into a graph theoretic problem. This graph theoretic representation is more amenable to the application of GA, since it admits easier coding, and more efficient definition of genetic operators. Computer experiments confirm the effectiveness of the GA solution.

5.1. Introduction

We consider the gender-fair matching in the stable marriage problem. Gale and Shapley introduced this problem in 1962 in their seminal

paper [17] "College Admissions and the Stability of Marriage". This problem must be regarded as a fundamental problem of resource management, such as job assignment in employment markets, dancing pairs of boys and girls, software processes and CPUs, autonomous robots and battery charger devices, even members of two sets of autonomous robots pairing up for cooperative work, and many other combinatorial problems. The most famous application is to match, as interns, graduating medical students with hospitals in USA. Many researchers in computer sciences, mathematics, economics, and operations research dealt with this problem in various ways [18,19]. There are many variations of the basic problems, a marriage instance of unequal size, declaration of unacceptable partners, existence of indifference, and so on [18].

The basic problem is explained as follows: We consider two sets of equal size n, the man group and the woman group,

$$\boldsymbol{m} = \{m_1, m_2, \ldots, m_n\}$$
$$\boldsymbol{w} = \{w_1, w_2, \ldots, w_n\}.$$

Each member in both sets possesses a strictly ordered preference list PL, containing all the members of the opposite sex. Let p be a person (man or woman) and q, r be members in the opposite sex of p. A person p prefers q to r if q precedes r on p's preference list. Two pairs (m_i, w_i) and (m_j, w_j) are said to be "unstable" if conditions

m_i prefers w_j to w_i, and

w_j prefers m_i to m_j

hold. When unstable, there is a high probability of break-up and a new pair formation (m_i, w_j). A matching $\boldsymbol{M} = \{(m_1, w_1), (m_2, w_2), \ldots, (m_n, w_n)\}$ is said to be "stable" if no two pairs in $\boldsymbol{M}$ are unstable.

Example 5.1 An instance of the problem is shown in Fig. 5.1. For example, the top of m_1's PL is w_5, the second w_2. The top of w_2's PL is m_7, the second m_4, and so on. Let

$\boldsymbol{M}_1 = \{(1,1),(2,5),(3,2),(4,3),(5,6),(6,8),(7,7),(8,4)\}$, and

$\boldsymbol{M}_2 = \{(1,5),(2,1),(3,2),(4,3),(5,6),(6,8),(7,7),(8,4)\}$.

Since pairs $(1,1),(2,5)$ are unstable, matching $\boldsymbol{M}_1$ is not stable. On the other hand, since no two pairs in $\boldsymbol{M}_2$ are unstable, $\boldsymbol{M}_2$ is a stable matching.

Gale and Shapley presented an algorithm to find a stable matching in their paper [17]. The stable matching found by their algorithm is extremal among many (for the worst case, in exponential order) stable matchings in such that each member in the male group gets the best

m_1	5	2	8	6	4	1	7	3	w_1	2	4	5	8	3	7	1	6
m_2	1	4	3	6	2	7	8	5	w_2	7	4	1	6	3	5	2	8
m_3	2	5	7	4	3	1	8	6	w_3	1	3	7	4	8	2	5	6
m_4	3	4	5	1	8	2	7	6	w_4	8	6	4	7	1	3	5	2
m_5	1	5	6	7	4	3	8	2	w_5	4	3	1	6	5	2	8	7
m_6	4	1	8	5	7	6	3	2	w_6	3	1	7	5	6	2	4	8
m_7	1	7	5	4	6	2	8	3	w_7	4	6	5	7	8	1	3	2
m_8	4	6	8	5	7	3	1	2	w_8	5	2	6	1	8	4	7	3

(a) Preference List PL

M_0 = {(1,5), (2,1), (3,2), (4,3), (5,6), (6,8), (7,7), (8,4)}
M_1 = {(1,5), (2,1), (3,2), (4,3), (5,7), (6,8), (7,6), (8,4)}
M_2 = {(1,2), (2,1), (3,5), (4,3), (5,6), (6,8), (7,7), (8,4)}
M_3 = {(1,5), (2,1), (3,2), (4,3), (5,8), (6,7), (7,6), (8,4)}
M_4 = {(1,2), (2,1), (3,5), (4,3), (5,7), (6,8), (7,6), (8,4)}
M_5 = {(1,2), (2,1), (3,3), (4,5), (5,6), (6,8), (7,7), (8,4)}
M_6 = {(1,2), (2,1), (3,5), (4,3), (5,8), (6,7), (7,6), (8,4)}
M_7 = {(1,6), (2,1), (3,5), (4,3), (5,7), (6,8), (7,2), (8,4)}
M_8 = {(1,2), (2,1), (3,3), (4,5), (5,7), (6,8), (7,6), (8,4)}
M_9 = {(1,6), (2,1), (3,5), (4,3), (5,8), (6,7), (7,2), (8,4)}
M_{10} = {(1,6), (2,1), (3,3), (4,5), (5,7), (6,8), (7,2), (8,4)}
M_{11} = {(1,2), (2,1), (3,3), (4,5), (5,8), (6,7), (7,6), (8,4)}
M_{12} = {(1,3), (2,1), (3,6), (4,5), (5,7), (6,8), (7,2), (8,4)}
M_{13} = {(1,6), (2,1), (3,3), (4,5), (5,8), (6,7), (7,2), (8,4)}
M_{14} = {(1,3), (2,1), (3,6), (4,5), (5,8), (6,7), (7,2), (8,4)}

(b) All Stable Matchings

	sm	sw	sm+sw	|sm-sw|
M_0	: 13	25	38	12
M_1	: 17	23	40	6
M_2	: 15	22	37	7
M_3	: 22	20	42	2
M_4	: 19	20	39	1
M_5	: 20	19	39	1
M_6	: 24	17	41	7
M_7	: 22	17	39	5
M_8	: 21	17	41	7
M_9	: 27	14	41	13
M_{10}	: 27	14	41	13
M_{11}	: 29	14	43	15
M_{12}	: 34	12	46	22
M_{13}	: 32	11	43	21
M_{14}	: 39	9	48	30

(c) Evaluations

Fig. 5.1 Instance of Size 8

partner over all stable matchings, and then each female gets the worst partner. We say Gale-Shapley algorithm can search the man-optimal(or woman-pessimal) stable matching. On exchanging each other's role between men and women, this same algorithm produces the woman-optimal(or man-pessimal) stable matching. Therefore it is quite interesting to find a fair stable matching(fair to the both genders), which is listed in [18,19] as one of important open problems related with the stable marriage problem, and there are no known polynomial time algorithms for it.

If we are searching a stable matching with a particular property, like gender-fair, then a brute force algorithm that examines all stable matchings requires exponential time complexity in the worst case since the number of stable matchings can grow exponentially with the group size of the marriage instance [18]. In [20] an algorithm is proposed to find an "egalitarian" stable matching which minimizes the sum of the partner's preference rank(or position) in each person's PL. However, it

is still possible for one gender to fare much better than the other in the egalitarian stable matching.

We in this chapter define and discuss a gender-fair stable matching that possesses the fairness property, that is, the sum of the men scores (rank of the partner in PL) is as close as possible to the sum of the women scores.

Example 5.2 All stable matchings of the marriage instance in Fig. 5.1(a) are shown in (b). Fig. 5.1(c) shows score evaluations of each stable matching. The column indicated by sm (sw) represents the sum of the rank of every man's (woman's) partner in his (her) preference for each matching. M_0 is man-optimal since it minimizes sm and maximizes sw. On the contrary, M_{14} is woman-optimal. M_2 is the solution of the egalitarian stable matching since it minimizes $sm+sw$. M_4 and M_5 are the gender-fair stable matchings since they minimize $|sw-sm|$. Note that in general the egalitarian stable matching is not gender-fair.

In this chapter, we present a method for adapting GA to the gender-fair stable marriage problems. The performance evaluation by computer experiment shows that for randomly generated marriage instances, the GA obtained the best solutions with high probability.

In Section 5.2 the preliminaries for this problem are given. Almost of all contents in Subsection 5.2.2 are from the book [18]. In Section 5.3, the main of this chapter, we give a method using a genetic algorithm (GA) for the gender-fair problems, that is, propose a method for adapting GA to the problem. In the method we transform the gender-fair stable marriage problem into combinatorial problem of graphs. This transformation makes application of GA easier and more effective. In Section 5.4, the performance of the GA solution is evaluated with computer experiment. Our approach provides a good example of GA strategy in that GA should be applied, not to the original but to the transformed problem of combinatorial natures.

5.2. Preliminaries

In this section, the preliminaries of this chapter, we define formally the gender-fair stable matching and summarize one of the most notable known results of the stable marriage problem, a lattice structure of all stable matchings, described in [18]. In the next section, we transform the gender-fair stable marriage into graph problem using the properties of the lattice structure.

5.2.1 Gender-fair Stable Matching

As stated in the previous section, the number of stable matchings can grow exponentially with its size n in the worst case. The Gale-Shapley algorithm finds the man(or woman)-optimal stable matching over all stable matchings. We in this chapter are interested in the gender-fair stable matching in which the sum of the man scores is as close as possible to that of the woman scores.

We define the man score, $sm(\boldsymbol{M})$, and the woman score, $sw(\boldsymbol{M})$, in the stable matching $\boldsymbol{M}$ as follows:

$$sm(\boldsymbol{M}) \triangleq \sum_{(m,w)\in\boldsymbol{M}} mr(m,w)$$

$$sw(\boldsymbol{M}) \triangleq \sum_{(m,w)\in\boldsymbol{M}} wr(m,w),$$

where $mr(m,w)$ means the rank of w in m's PL and $wr(m,w)$ the rank of m in the w's PL. Then, the man-optimal matching minimizes $sm(\boldsymbol{M})$ and maximizes $sw(\boldsymbol{M})$ over all stable matchings. The woman-optimal matching does the reverse. Moreover we define the score of matching $\boldsymbol{M}$, $s(\boldsymbol{M})$, to be

$$s(\boldsymbol{M}) \triangleq sw(\boldsymbol{M}) - sm(\boldsymbol{M}) \ .$$

We now can define a gender-fair matching.

Definition 5.1 Let the gender-fair stable matching be the matching $\boldsymbol{M}$ giving

$$\min_{\boldsymbol{M}\in\mathbf{M}} |s(\boldsymbol{M})|,$$

where $\mathbf{M}$ is the set of all stable matchings.

5.2.2 The Structure of All Stable Matchings

In [18,19] it is shown that the set of all stable matchings forms a distributed lattice under a natural ordering relation, and that the man-optimal and the woman-optimal matchings represent the minimum and

the maximum elements of the lattice, respectively. Therefore Gale-Shapley algorithm can find the extremes of the stable matchings, the man-optimal or the woman-optimal. The other stable matchings are situated between these two extremal matchings in a lattice structure to be explained shortly. We are now interested in gender-fair stable matchings that should be located in the middle of the structure.

Definition 5.2 Stable matching M is said to dominate stable matching M', represented by $M \preceq M'$, if every man either prefers the partner in M to in M' or has the same partner.

Theorem 5.1[18,19] *For a given instance of the stable marriage problem, the partial order $(M, \preceq)$ forms a distributive lattice, with the meet and join of M and M' denoted by $M \wedge M'$ and $M \vee M'$, where $M \wedge M'$ $(M \vee M')$ is the stable matching in which each man receives the better (poorer) of his partners in M and M'.*

As described above, the man-optimal matching and the woman-optimal matching represent the minimum and the maximum of the lattice, respectively.

Example 5.3 The lattice structure of the instance in Fig. 5.1 is shown in Fig. 5.2. Each stable matching is described by a vector of length 8, where the number in position i of the vector indicates the partner of m_i in the stable matching. Below is a vector indicating the rank of the partner of m_i in his PL. Each edge denotes a dominate relation. The label on each edge should be ignored for now.

Definition 5.3 Let M be a stable matching. For any man m let $s_M(m)$ be the first woman w on m's PL such that w strictly prefers m to her partner in M. Let $next_M(m)$ be the partner in M of woman $s_M(m)$.

Definition 5.4 Let us define the rotation ρ be a sequence of pairs, $(m_0, w_0)(m_1, w_1)...(m_{r-1}, w_{r-1})$, in a stable matching M such that, for each m_i, $m_{i+1 \bmod r}$ is $next_M(m_i)$.

As shown in [18], if M is a stable matching and if ρ is a rotation in M, then the matching, denoted by M/ρ, in which each man m_i appearing in ρ matches $w_{i+1 \bmod r}$ and the others match the same partners as in M, is stable and dominated by M.

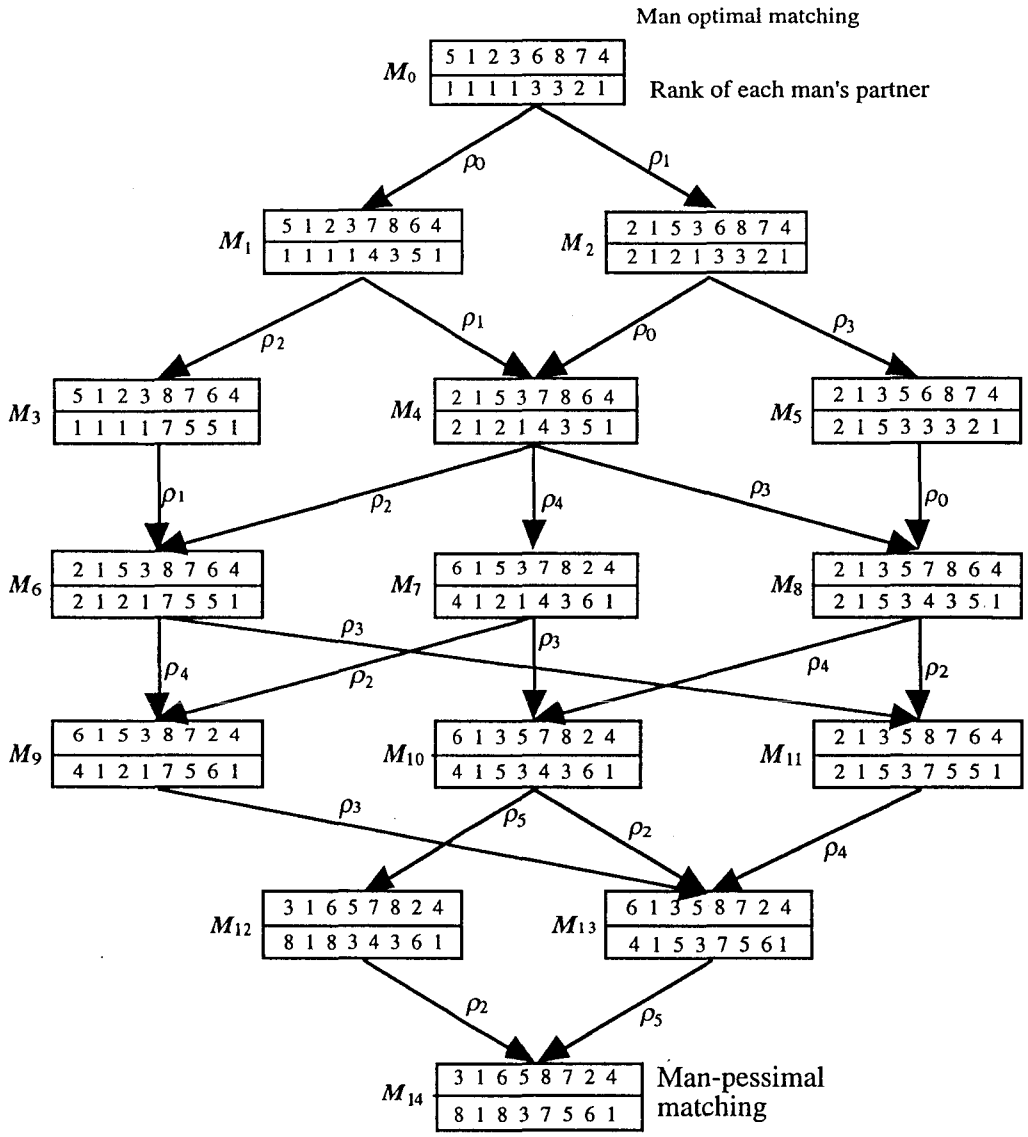


Fig. 5.2 Lattice structure of the instance in Fig. 5.1

Lemma 5.1[18] *If ρ is any rotation in a stable matching M, then M/ρ is a stable matching dominated by M.*

Example 5.4 Fig. 5.2 shows the lattice structure of the stable matchings shown in Fig. 5.1 and the label on each edge represents a rotation. We can see that $M_1=M_0/\rho_0$, $M_4=(M_0/\rho_0)/\rho_1$, and so on.

The rotation digraph $\vec{G}$ is acyclic and composed of the set of all rotations in the instance (as the node set), and the edge set which is

defined in [18]. The definition of the edge is omitted in this chapter since we do not need it. The rotation digraph is obtained from preference lists in $O(n^2)$ time, where n is the instance size. The detail is described in [18].

Definition 5.5 A subset S of the node set of $\vec{G}$ is said to be *closed* if there is no element, in the complementary set of S, that precedes an element in S.

The following lemma [18] is one of the most important results in the stable marriage problem.

Lemma 5.2[18] *There is a one-one correspondence between the closed subsets of $\vec{G}$ and the stable matchings of M, where M is the set of all stable matchings.*

In the lattice structure of an instance, for each stable matching M, the rotations on the path from M_0 to M constitutes the closed subset corresponding to M.

Example 5.5 The rotation digraph $\vec{G}$ of the instance of Fig. 5.1 is shown in Fig. 5.3. A closed subset $\{\rho_0, \rho_1\}$ corresponds to M_4. The rotations in this subset corresponds to the path from M_0 to M_4 in Fig. 5.2.

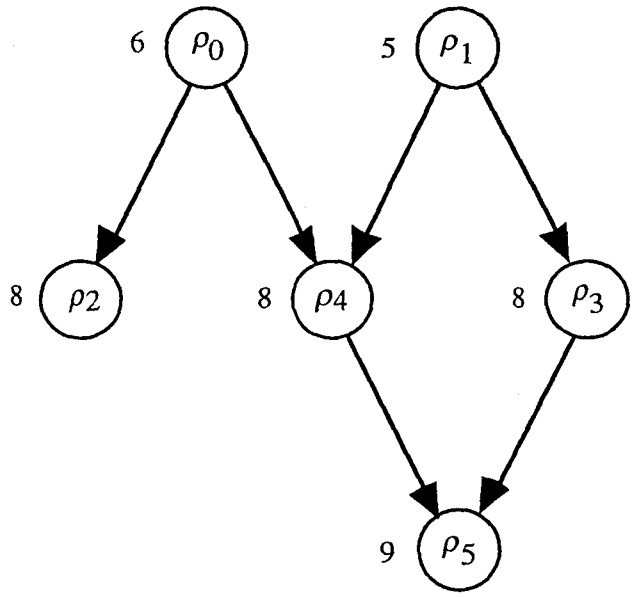


Fig. 5.3 Rotation digraph of the instance in Fig. 5.1
The number beside each node represents its weight

5.3. Adaptation of a Genetic Algorithm

In this section, we propose a method to adapt GA to the gender-fair problem. In Subsection 5.3.1, we transform the gender-fair marriage into a graph problem. In APPENDIX, we define a decision problem, CCS, associated with the gender-fair problem and show its NP-completeness. However, the graph problem representation is very suitable for GA solution because of the stability being transformed into combinatorics of the closed subset of graphs. In 5.3.2, we show a GA solution and its effectiveness.

5.3.1 Graph Problem Representation

For a rotation $\rho = (m_0, w_0)(m_1, w_1)...(m_{r-1}, w_{r-1})$, we define the weight of ρ, $s(\rho)$, to be

$$s(\rho) \triangleq \sum_{i=0}^{r-1} (mr(m_i, w_{i+1 \bmod r}) - mr(m_i, w_i)) - \sum_{i=0}^{r-1} (wr(m_i, w_{i+1 \bmod r}) - wr(m_i, w_i)).$$

Note that $s(\rho) \geq 2$ when ρ is not empty.

Example 5.6 In Fig. 5.3, the number beside each node represents the weight of a rotation.

We have the following lemma.

Lemma 5.3 *If ρ is a rotation of a stable matching M, then $s(M / \rho) = s(M) - s(\rho)$.*

Proof: By Lemma 5.1,

$$M / \rho = M \setminus \{(m_0, w_0), (m_1, w_1), ..., (m_{r-1}, w_{r-1})\} \cup \{(m_0, w_1), (m_1, w_2), ..., (m_{r-1}, w_0)\}.$$

Therefore,

$$\begin{aligned} s(M / \rho) &= s(M) - \sum_{i=0}^{r-1} (wr(m_i, w_i) - mr(m_i, w_i)) \\ &\quad + \sum_{i=0}^{r-1} (wr(m_i, w_{i+1 \bmod r}) - mr(m_i, w_{i+1 \bmod r})) \\ &= s(M) - s(\rho) \end{aligned}$$

Q.E.D

By the fact that there is a one-one correspondence between the stable matching M and the closed subset, CS, of the rotation digraph $\vec{G}$, the following lemma is immediately obtained.

Lemma 5.4 *If CS is a closed subset of $\vec{G}$ associated with stable matching M, then*

$$s(M) = s(M_0) - \sum_{\rho_i \in CS} s(\rho_i).$$

Proof: It is obvious by Lemma 5.2, 5.3. Q.E.D.

By Lemma 5.4 and Definition 5.1, we have the following theorem.

Theorem 5.2 *The gender-fair stable matching is the matching generated by a closed subset of $\vec{G}$, CS, such that $\sum_{\rho_i \in CS} s(\rho_i)$ is as close as possible to $s(M_0)$.*

Example 5.7 In Fig. 5.3, we can find closed subsets $\{\rho_0, \rho_1\}$ and $\{\rho_1, \rho_3\}$ whose weight sum, 11 and 13, respectively, are closest to $s(M_0)(=12)$.

For a given marriage instance, we can get its rotation digraph in $O(n^2)$ time. However, there is no known polynomial time algorithm for the gender-fair matching. In the next subsection, we show a genetic algorithm as an approximate algorithm.

5.3.2 A Genetic Algorithm for Gender-fair Matching

In APPENDIX, we show the decision problem CSS which is associated with the gender-fair marriage is NP-complete. It means that it is difficult to solve the gender-fair stable matching problem. In this section we show a GA for obtaining approximate solutions of the gender-fair stable matching. Since genetic algorithms have received much attention for various optimization problems [14,15], we attempted first to apply GA to the original gender-fair stable marriage problem without success, due primarily to coding difficulty of stability property. The graph problem representation is found much more suitable for GA solution than the original.

For a given marriage instance, that is, PLs, first we obtain the rotation digraph and the man-optimal solution M_0. Secondly, the GA tries to find the closed subset CS such that $\sum_{\rho_i \in CS} s(\rho_i)$ is as close as possible to $s(M_0)$. Thirdly we generate in polynomial time the

corresponding stable matching from the CS and M_0 (See Lemma 5.2, Example 5.5, and Ref.[18] for the detail).

The most important point of GA is to design the string representation for the search objects. The string representation used in this chapter is simple: Each node in an obtained directed graph $\vec{G}$ (rotation digraph) corresponds to a bit in the string. A node i corresponds to bit b_i, i-th(starting from zero) component of the bit string $\boldsymbol{b} = (b_0, b_1, \ldots, b_{n-1})$. If node i belongs to a closed subset CS, b_i equals 1. We regard every predecessor j of a node i such that $b_i = 1$ as also included in the CS. This is for that every string representation in the GA space should correspond to a legal search object, that is, a CS. If not so, there may occur numerous bit-strings corresponding to illegal search objects, that is, non-closed subsets. Note that a bit corresponding to such a predecessor remains zero though we define that the corresponding nodes belong to the CS. These bits are called "hidden zero bits", which may be a part of better solutions in future generations.

Example 5.8 For an instance in Fig. 5.3, there are 6 rotations and hence 6 nodes in $\vec{G}$, bit representations 100110, 110110 correspond to the same closed subset $\{\rho_0, \rho_1, \rho_3, \rho_4\}$. The second bit b_1 of the first string is a hidden zero bit.

The fitness function typically represents the objective function that we want to optimize in the problem. The function in this chapter is defined as follows:

$$f(\boldsymbol{CS}) = \left| s(\boldsymbol{M}_0) - \sum_{\rho_i \in \boldsymbol{CS}} s(\rho_i) \right|$$

The reproduction in this chapter is based on the roulette wheel selection. Thus, CS with a lesser value of the fitness function will survive with higher probability.

In this chapter, crossover and mutation are used as the genetic operators.

Crossover: Let us consider two strings P_1 and P_2. (1) Select randomly crossover point where we cut the string into two parts for each string. (2) Exchange the bottom parts of P_1 and P_2. The new two strings are generated.

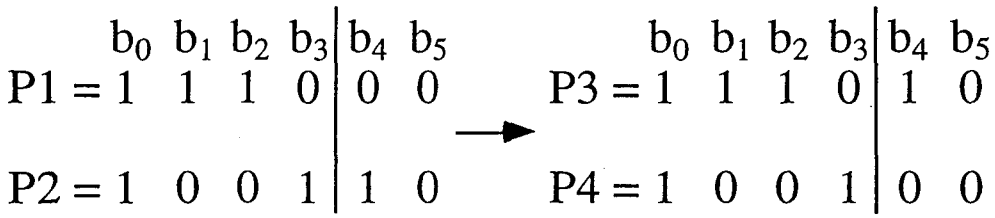


Fig. 5.4 Crossover Operation

Example 5.9 An example of the crossover is shown in Fig. 5.4. The crossover point is between b_3 and b_4 of P_1 and P_2. The new strings P_3 and P_4 are created.

Mutation: Let us consider a string P. (1) Select randomly a mutation point. (2) Reverse the bit of the selected point.

The GA in this chapter is simple and consists of the following steps:

«The Genetic Algorithm»

1° Initialization: An initial population is randomly generated.

2° Evaluation of the fitness function: The fitness value of each string is calculated according to the fitness function.

3° Genetic operations: For randomly selected strings, the crossover and/or the mutation are applied.

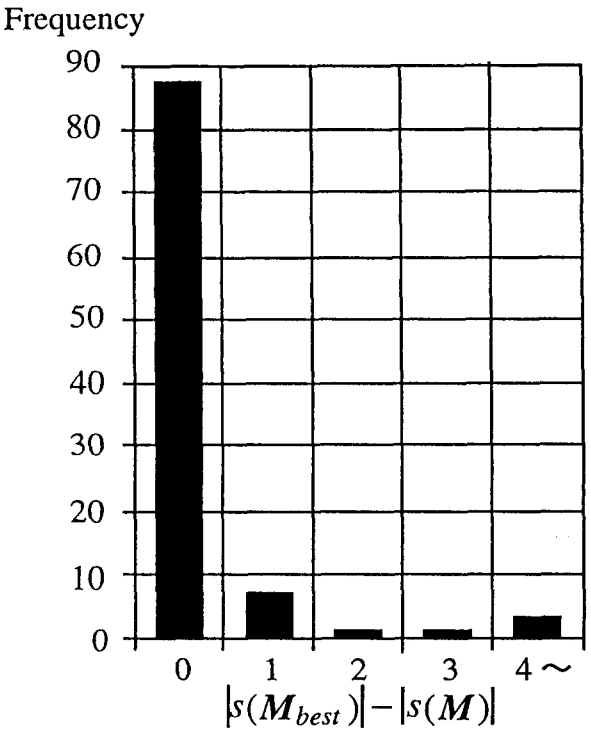
4° Reproduction: The population for the new generation is generated by the roulette wheel selection.

5° If time is up, stop and return the best string, if not, go to 2°.

5.4. Experimental Evaluation

We evaluate the performance of the GA by experiments on a Sun workstation ELC. The GA evaluation system is implemented by C language. One hundred marriage instances, that is, 100 sets of 30x30 men-to-women and women-to-men PLs, are generated randomly. For each marriage instance, the rotation digraph is obtained by the method described in [18] with $O(n^2)$ time complexity. And the following parameters are used throughout the experiment:

- population size = 50
- crossover probability = 1.0



Conditions
- population size = 50
- crossover probability = 1.0
- mutation probability = 0.3
- generation span = 50

abcissa: deviation from the best solution
(average of 10 times executions)
ordinate: frequency in 100 instances

Fig. 5.5 Evaluation of the GA

- mutation probability = 0.3
- generation span = 50

For each rotation digraph, the GA is executed 10 times. The computation of one execution for one marriage instance required less than 1.0 second (for this case, a brute force algorithm required more than 5.0 minutes). At every marriage instance we could get the best solution at least once during 10 executions. Fig. 5.5 shows the number of instances for each deviation (average of 10 times) from the best solution. There are 88 percents of 100 marriage instances for the

deviation 0 exactly, that is, every execution could obtain the best solution and 97 percents for the deviation below 3.

5.5. Concluding Remarks

In this chapter we discussed a gender-fair matching in the stable marriage problem. The stable matching is defined gender-fair when the sum of the man scores is as close as possible to the sum of the woman scores. We proposed a method for adapting GA to the gender-fair marriage. By experimental evaluation, we showed the effectiveness of the GA.

APPENDIX

We define a decision problem associated with the gender-fair stable matching problem and show its NP-completeness.

Definition 5.A1 A decision problem CSS is defined as follows:
Instance: An acyclic directed graph $\vec{G}$ in which every node is weighted with positive integer, a positive integer B.
Question: Is there a closed subset CS of $\vec{G}$ such that the sum of the individual weights of nodes in CS is equal to B ?

Definition 5.A2 A decision problem SS (Subset Sum) is defined as follows:
Instance: Finite set A, weight $w(a) \in Z^+$ for each $a \in A$, positive integer b.
Question: Is there a subset S such that the sum of the weights of elements in S is exactly b?

SS is well known to be NP-complete[23]. We will transform SS to CSS for proving NP-completeness of CSS in a proof of the following theorem.

Theorem 5.A1 *The decision problem CSS (Closed Subset Sum) is NP-complete.*

Proof: Since there exists a nondeterministic algorithm that needs polynomial time to assemble a closed subset CS of $\vec{G}$ and to check its weight sum of the nodes in CS is equal to B, CSS is in NP.

. Next, we transform in polynomial time SS to CSS. The elements of A are nodes of $\vec{G}$. The weight of each element is transformed to the weight of its corresponding node. Let node p with weight one be introduced as a node of instance of CSS. Define $\vec{G}$ by introducing a directed edge from p to the others. That is $\vec{G}=(V,E)$ where $V=A\cup\{p\}$, $E=\{(p,a)|\forall a\in A\}$. And $B=b+1$. Now we have an instance of CSS.

Suppose there be a subset S of A such that the sum of the element weights in S is exactly b, a solution of SS. Then the sum of the node weights in $S\cup\{p\}$ is exactly B. Moreover $S\cup\{p\}$ is a closed subset of $\vec{G}$.

Suppose there be a closed subset $CS=S\cup\{p\}$ such that the sum of the node weights in CS is exactly B. Note that S is a subset of A. Since the weight of p is one, the sum of the element weights in S is exactly b, say SS solution. Q.E.D.

Chapter 6

Genetization of Heuristic-Knowledge in Genetic Algorithms and Its Application to Multiprocessor Scheduling Problems

We propose a genetized-knowledge genetic algorithm (abbreviated as gkGA), and discuss its application to multiprocessor scheduling problems. In this approach, the heuristic itself is represented by genes, and by means of GA selection superior heuristics survive for a given problem, while others die out during the selection process. Moreover, by the crossover of heuristic genes themselves, hybridized heuristics can also be generated. This is a novel strategy in the GA field of combinatorial problems; although the idea of reinforced GAs was already proposed, heuristics have not yet been explicitly genetized. The effectiveness of our proposed strategy for multiprocessor scheduling problems is proved through computer evaluation.

6.1 Introduction

By many studies [14,16,51,52], genetic algorithms have been proved to possess potentialities for combinatorial optimization problems, like the multiprocessor scheduling. Some papers [14,16,21,51] pointed out genetic algorithms are good at global searches but not at local one, and then proposed a reinforcement of genetic algorithms with a heuristic technique tuned to local search effectiveness, so-called hybrid GAs.

In a reinforced GA, a prescribed domain knowledge is employed to produce an efficient string that leads to better solutions compared with ordinary solutions by a simple genetic algorithm. In this chapter we develop a novel GA in which domain knowledge is genetized and is included in a string as genes. A given set of heuristic techniques constitutes a gene pool of knowledge, and crossover of knowledge genes A and B generates hybrid genes, AB. The new method is called "genetized-knowledge GA," in short gkGA.

The multiprocessor scheduling problem is known NP-hard even if various restrictions are added to simplify the problem [22-24]. Therefore, many researchers have tried to develop approximation algorithms rather than to exact the optimization [21,25,26]. From many years' research, most effective solutions are known to be based on the list scheduling scheme[22,26]. In this scheme quality of a schedule is known greatly dependent on a prescribed priority list to be used, and several methods for organizing priority lists were proposed as summarized in [22]. The critical path (CP) method organizes a priority list in such that a task is arranged in a sequence according to its time depth [22]. Kasahara [27] proposed a modified CP method, called CP/MISF(Critical Path / Most Immediate Successors First), in which a task having the most number of immediate successors has higher priority over multiple tasks possessing the same time depth. Nakasumi [28] tried ordering tasks of the same time depth by appealing to genetic algorithms. These modified CP algorithms were reported to produce good quality schedules for wide range of task graphs [22,27,28]. However, it was revealed that for certain kinds of task graphs, these modifications were led to not good schedules. We in this chapter propose a better solution to the problem over much wider range of task graphs even though it dissipates more computation time.

We first introduce four reinforced GA based on four heuristics from ordinary idea of scheduling research. No heuristics of these are uniformly superior to the others, because performance of the heuristics is problem dependent. In the gkGA, the heuristics are represented as genes in the string, and hybrid genes generated by the crossover process offer new heuristics that may prove to be more efficient to a given task graph. This is a novel strategy in the GA field although there are studied some reinforced GAs in which heuristics are not genetized. Moreover, our new method uses a modified list scheduling rather than the original one to overcome its weakness and to be more suitable for GA-based methods. By computer experiment, we prove the proposed gkGA can generate better schedules compared with the CP/MISF and Nakasumi's GA.

This chapter is composed of five sections. In Sect. 6.2 we give the formal description of the problem and a small example for readability. In Sect. 6.3, we introduce four heuristics and explain how to generate a schedule. In Sect. 6.4, we propose the genetized-knowledge GA, the gkGA. The conclusion is described in Sect.6.5.

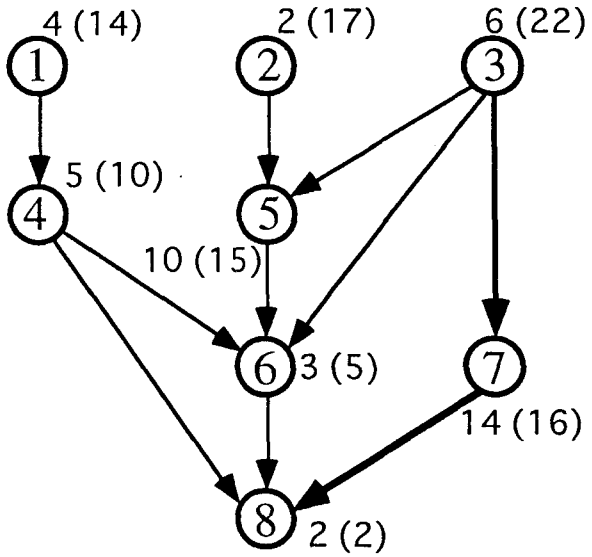
6.2 Preliminaries

6.2.1 Problem Definition

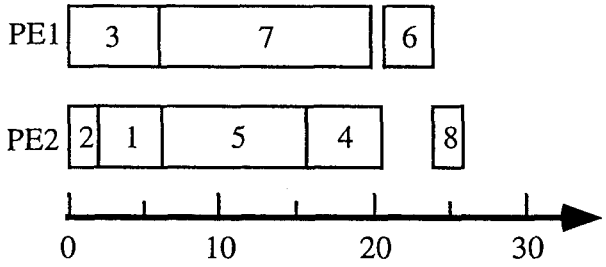
A process is represented by an acyclic digraph (directed graph) of a single sink called a task graph, $TG=(V,E)$ with a set of vertices, $V=\{T_1, T_2, ..., T_k\}$, and a set of directed edges, $E=\{e_{ij}\}$. Each vertex, T_i, represents a task to be executed, and is given its execution time, t_i. An edge represents a precedence relation between its two incident vertices. The edge from T_i to T_j means a precedence relation (>), $T_i > T_j$, meaning T_i must be completed before T_j is initiated. The multiprocessor scheduling is to assign each task of the graph to a free processor from the pool of p processors in such that each processor executed only one task at a time without preemption while maintaining the precedence relation. The time to the last task(sink) completed is called the finish time of the schedule. The multiprocessor scheduling problem requires to construct the schedule of the earliest finish time.

Example 6.1: Figure 6.1(a) shows an example of a task graph. A number attached to a vertex represents the execution time of its associated task. When the number of processors is two, Figure 6.1(b) is the optimum schedule, that is, the schedule giving the earliest finish time.

In a task graph, the path that includes the maximum number of tasks, among the paths from a task T_i to the sink task T_z, is called the deepest task path of T_i, and this maximum number is called "the depth of Ti," denoted by $d(Ti)$. The longest timed path of a task T_i is a path from T_i to the sink Tz such that the sum of the execution times of the tasks on



(a) Task Graph
the number beside a node: its execution time
and (time depth)



(b) The Optimal Schedule
(2 processors)

Fig. 6.1 Problem Instance

the path is largest. The largest sum for a task T_i is called "the time depth of T_i ," and denoted by $td(T_i)$.

6.2.2 Task Assignment Scheme TAS

In this subsection, we describe first the conventional list scheduling, and next show a method suitable for our purpose. The list scheduling scheme assigns an available task to the processor that becomes free earliest. If multiple tasks are available to assign, the prior task in a given priority list is first assigned. The rule is iterated until all the tasks

are assigned. The original list scheduling scheme is likely to yield good, even optimal, schedules with a high probability when it is given an appropriate priority list.

However, there are some weak points of its own as pointed out in [22]. In the original list scheduling scheme, when a processor becomes idle, the standard rule dictates the processor must be assigned an executable task when possible. This "no-forced-idle" rule sometimes leads to a bad schedule.

Example 6.2[22]: Figure 6.2 shows an example. Figure 6.2(a) is the task graph, 6.2(b) the schedule generated by the original list scheduling from the priority list organized by the CP/MISF, and (c) the optimum schedule. Note that for the task graph the original list scheduling cannot generate the optimum for any priority list.

To remedy this hindsight of "no-forced-idleness," we use a simple scheme, denoted by "TAS."

«TAS»

1° Let t = the top of the priority list PL and remove it from PL.

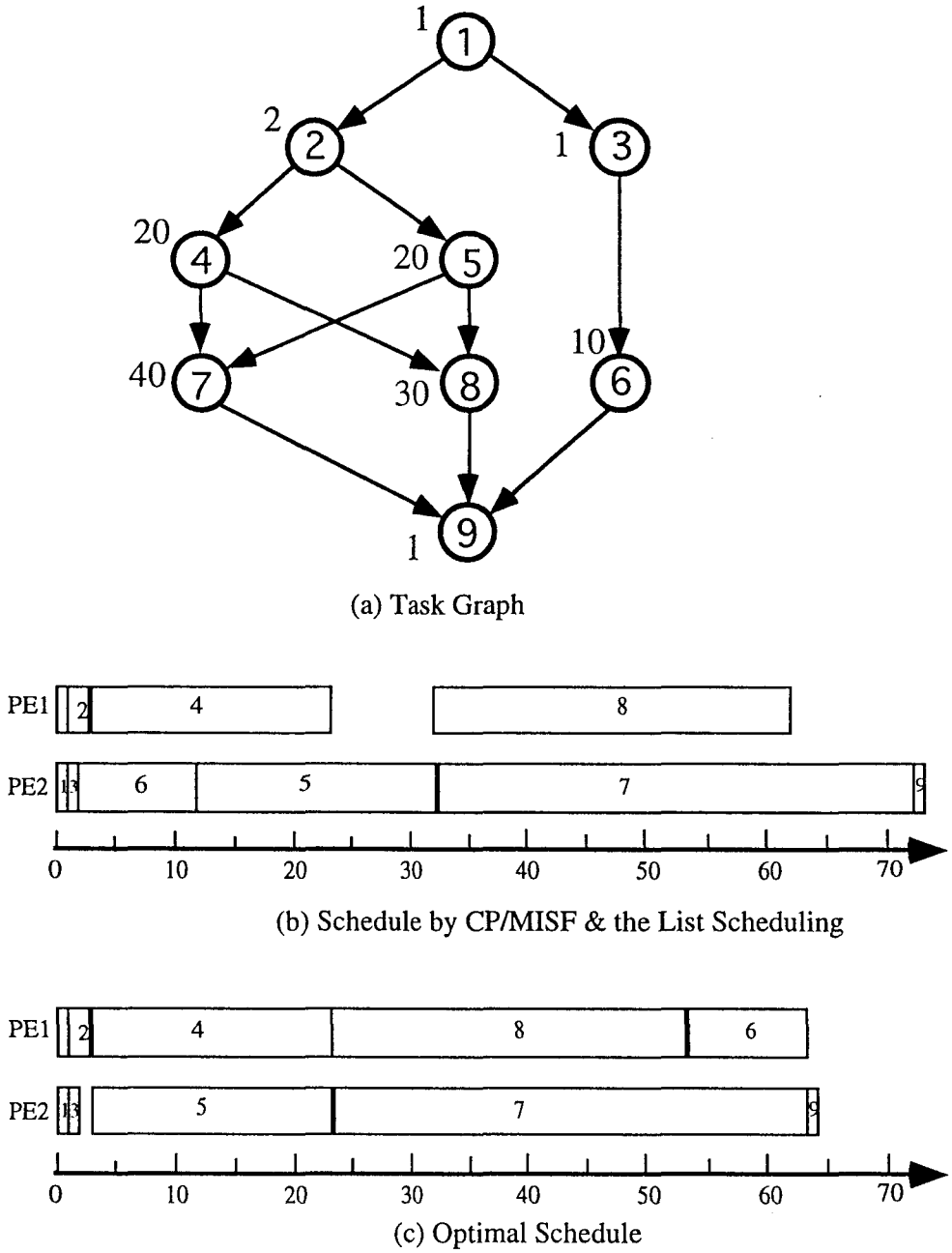
2° If all the predecessors of t are already assigned, t is assigned to an idle processor available. The initiating time of t is defined by: $\max(\min_i \{idle\_t(PE_i)\}, \max_j\{c\_t(T_j) | T_j$ is an immediate predecessor task of $t\})$ where $idle\_t(PEi)$ is the time that PE_i becomes idle and $c\_t(T_j)$ is the time that T_j is completed. Otherwise, exit with failure.

3° If PL is empty, we have a schedule. Otherwise, return 1°.

Example 6.3: For the task graph shown in Fig. 6.2, TAS generates the optimum schedule when we use a priority list:

1 2 3 4 5 8 7 6 9.

Note that the original list scheduling scheme produces a schedule that is not optimum. Showing that every schedule generated by the original one is also generated by TAS is easy. As to be described in Sect. 6.3, every priority list organized by our proposed method can be converted into a schedule by TAS, that is, no failure at Step 2. Since more priority lists should be converted into one schedule by the original list scheduling method than by TAS, TAS can generate more variety of schedules from the strings in GA pool than the original one. Moreover, TAS has lesser



(a) Task Graph

(b) Schedule by CP/MISF & the List Scheduling

(c) Optimal Schedule

Fig. 6.2 Weak Point of the List Scheduling[1]

computational complexity than the original one. Therefore, we can say TAS is more suitable to methods using genetic algorithm techniques.

6.3 Four Reinforced Genetic Algorithms

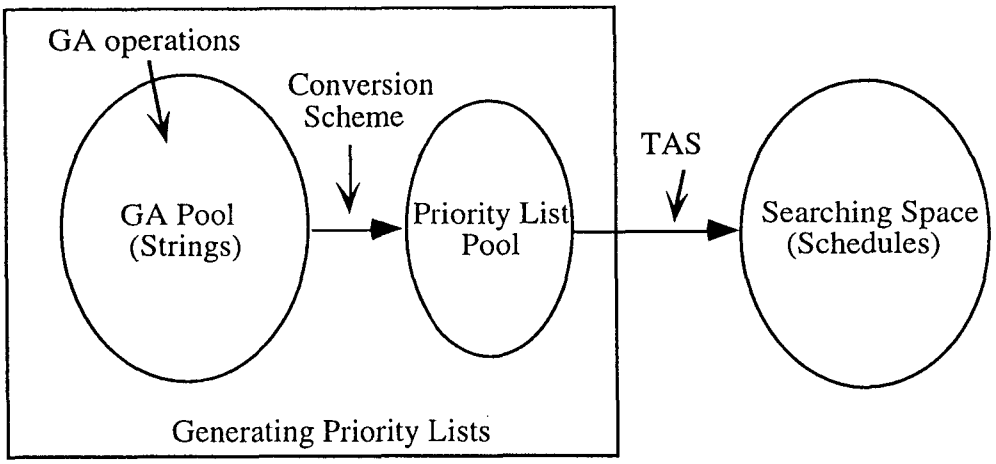


Fig. 6.3 Outline of the Proposed Method

In this section we introduce four heuristic reinforcements of GA as prelude to the gkGA. In Sect. 6.4, the main part of this chapter, we propose the gkGA by genetizing a knowledge pool of heuristics.

Figure 6.3 shows the outline of our solution. The strings in the GA pool (population) are converted and reshaped by a conversion scheme based on a heuristic into a priority list, and then a schedule is constructed by TAS.

By experimental evaluation, we prove that the proposed reinforcement GAs performs generally better than the CP/MISF and Nakasumi's method.

6.3.1 String Representation and Genetic Operations

(1) String Representation and Initial Population

A gene can be a task identifier or knowledge. However, in this section we consider only a task identifier. We call a string in this section a task string since a string is sequence of the task identifiers. For a given task graph, the strings in the initial population are randomly generated in such that the task identifier is arranged monotonously in a sequence in accord with the depth of the task, the largest the first. This is called the depth monotonicity. That is, for a string $T_1, T_2, ..., T_k$, $d(T_1) \geq d(T_2) \geq ... \geq d(T_k)$.

Example 6.4: The following is an example of the task string for the task graph in Fig. 6.1:

1 3 2 | 5 4 | 7 6 | 8

Here the vertical bar represents the border between adjacent two different depths. The tasks in {1,3,2} are of depth 4, those in {5,4} of depth 3, and so on.

(2) Crossover of Two Strings

We use the OX method [16,51], as a crossover operation, to maintain the depth monotonicity of the generated strings. In the OX method, given a randomly generated 0-1 mask pattern, new two strings of the depth monotonicity are generated from given two initial strings.

«OX method»

1° Choose two strings randomly from the GA population. Let us denote them by P1 and P2, and create a new empty string C1 of the same length.

2° Generate randomly a 0-1 bit pattern of the same length as the string (equal to the number of tasks).

3° For each task identifier of $P1$, if the bit corresponding to the task is one, then the task is copied into the same gene place of a new string $C1$.

4 ° Pick up the tasks corresponding to the zero bits of the mask from $P1$.

5° Copy the picked up genes to the empty gene place of $C1$ in the same order in $P2$.

6° Generate a new string $C2$ by exchanging the roles of $P1$ and $P2$.

Example 6.5: An example of the OX method is as follows: Here M represents a randomly generated mask pattern.

$P1$: 1 2 3 4 5 6 7 8

$P2$: 2 1 3 5 4 7 6 8

M : 0 0 0 1 0 0 1 1

$C1$: 2 1 3 4 5 6 7 8

$C2$: 1 2 3 5 4 7 6 8

(3) Mutation

«Mutation»

1° Choose randomly one string, say P, from the GA population.

2° Choose randomly two genes, say T_i and T_j, of P at the same depth.
3° Exchange the places of T_i and T_j.

Lemma 6.1: *For an initial population satisfying the task monotonocity, the operations of crossover and mutation preserve the depth monotonicity.*

Proof: It is obvious since no change of tasks beyond task depth boundaries occurs in the crossover and the mutation. Q.E.D.

6.3.2 Conversion of Strings into Priority Lists by Heuristics

In this subsection, we describe how to convert a string, s, to a priority list P. We in this chapter use the following four conversion schemes based on heuristics.

«Scheme O»

1° Let P=s. That is, a string is simply a priority list.

«Scheme A»

1° Decompose a string s into substrings $s1$, $s2$, ..., sk such that the genes in s_i have the depth i. Note that $s=s1 \mid s2 \mid ... \mid sk$, where " | " denotes concatenation. Put the genes of s_i into a queue q_i from the left to right, for $1 \le i \le k$.
2° Make P empty.
3° Choose the top of each q_i, $1 \le i \le k$, whose predecessors are already assigned.
4° Append the task, T_j, that has the longest execution time, $t(T_j)$, among the tops chosen at 3° to the tail of P. Remove T_j from the associated queue.
5° Unless every substring is empty, return 3°.

«Scheme B»

Scheme B is the same as Scheme A except for Step 4.

4° Append the task, T_j, that has the most immediate successor tasks, $suc(T_j)$, among the tops chosen at 3° to the tail of P. Remove T_j from the associated queue.

«Scheme C»

Scheme C is the same as Scheme A except for Step 4.

4° Append the task, Tj, that has the largest time depth, $td(Tj)$, among the tops chosen at 3° to the tail of P. Remove Tj from the associated queue.

For any two tasks in a priority list, when the task at more left side is not a successor of the other in the task graph, the priority list is called "sequentially left executable list." Apparently, a schedule can be generated from a sequentially left executable priority list by TAS.

Lemma 6.2: *A priority list is sequentially left executable when it is arranged monotone decreasing of task depth.*

Proof: Suppose the j-th task in the priority list be a predecessor task of the i-th task, $i<j$. This is contradiction for the monotonous decreasing of their depth. Q.E.D.

Theorem 6.1: *The strings generated by the crossover and the mutation defined in Sect. 6.3.1 are converted into sequentially left executable priority lists by Scheme O, A, B, and C.*

Proof: For Scheme O, it is obvious by Lemma 6.1, 6.2. For A, B, and C, we check the predecessor relation at step 3 of each scheme. Q.E.D.

6.3.3 Fitness Function and Description of the Reinforced GA

The strings generated by one of the schemes shown in 6.3.2 are converted into priority lists. Then, by TAS, schedules are generated from the priority lists. In our method, the fitness function of the GA is defined by the length of a schedule as follows:

$$f(s)=Large\_num\text{-}\max tpj$$

where s is a string, tpj denotes the complete time of a processor PEj in the schedule generated from s, and $Large\_num$ means a large positive number. Apparently, larger value of the fitness corresponds to better schedule.

Now we have the description of the proposed GA as follows:

«Reinforced GA»

1° Initialization: An initial population of task strings is randomly generated by (1) in 6.3.1.

2° Evaluation of the fitness function: The fitness value of each string is calculated according to the fitness function defined above.

3° Genetic operations: For randomly selected strings, the crossover (2) in 6.3.1 and/or the mutation (3) in 6.3.1 is applied.
4° Reproduction: The population for the new generation is generated by the roulette wheel selection[26].
5° If time is up, stop and return the best string, if not, go to 2°.

6.3.4 Experimental Evaluation

We evaluate experimentally our methods and compare with the CP/MISF and Nakasumi's method. The evaluation system is constructed on Sun SS5(75MHz) workstation by C language. In the experiment, for the two cases, the number of tasks = 50 and 100, task graphs in which each task has from 1 to 5 successors are randomly generated. The parameters are as follows:

The number of processors = 4, 6,
The number of populations = 50,
The generation span = 50,
The crossover probability = 1.0,
The mutation probability = 0.03, 0.06.

The computation time of the CP/MISF method is less than 1.0 seconds, and those of our methods are about 15 seconds. Tables 6.1, 6.2, 6.3, and 6.4 show the results for randomly picked up eight from generated problems, task50.1-5 for the number of tasks =50, and task100.1-3 for that of tasks = 100. The contents of the tables shows the length of the schedules. These are the average of 10 times executions. The shaded values mean the optimal among 7 solutions. The gkGA is to be proposed in Sect. 4, therefore, its results should be ignored for now.

We can observe that (i) our methods generally obtain better schedules than the CP/MISF and Nakasumi's method, and (ii) performance of our four methods depends on the problem instance. The reason of (i) is that our method can search wider searching space because Nakasumi's method does try to search only the schedules generated from the priority list having monotonous property of time depth. From (ii) we need a new methodology of automatically heuristic adaptation to any problems. In this line of thought, we propose in the next section a new strategy, the genetized-knowledge GA.

Table 6.1 Experimental Evaluation (mutation_rate=0.03, 4 processors)

	CP/MISF	Nakasumi	Scheme O	Scheme A	Scheme B	Scheme C	gkGA
task50.1	142	140	140	136	140	135	135
task50.2	158	152	152	147	152	152	144
task50.3	149	147	146	148	150	147	146
task50.4	128	126	126	122	126	126	122
task50.5	128	126	126	122	126	126	122
task100.1	273	272	272	267	272	272	268
task100.2	263	261	261	255	261	261	255
task100.3	265	260	260	261	260	260	260

Table 6.2 Experimental Evaluation (mutation_rate=0.03, 6 processors)

	CP/MISF	Nakasumi	Scheme O	Scheme A	Scheme B	Scheme C	gkGA
task50.1	101	98	98	94	98	93	93
task50.2	111	107	107	102	107	107	100
task50.3	115	115	115	115	115	115	112
task50.4	128	89	89	84	89	89	85
task50.5	128	89	89	85	89	89	84
task100.1	187	184	185	181	185	184	181
task100.2	183	179	179	173	179	180	173
task100.3	181	175	175	176	175	175	175

Table 6.3 Experimental Evaluation (mutation_rate=0.06, 4 processors)

	CP/MISF	Nakasumi	Scheme O	Scheme A	Scheme B	Scheme C	gkGA
task50.1	142	140	140	136	140	135	135
task50.2	158	152	152	146	152	152	144
task50.3	149	147	147	149	149	147	146
task50.4	128	126	126	122	126	126	122
task50.5	128	126	126	122	126	126	122
task100.1	273	272	272	267	272	272	268
task100.2	263	261	261	255	261	261	255
task100.3	265	260	260	260	260	260	260

Table 6.4 Experimental Evaluation (mutation_rate=0.06, 6 processors)

	CP/MISF	Nakasumi	Scheme O	Scheme A	Scheme B	Scheme C	gkGA
task50.1	101	98	98	97	98	93	94
task50.2	111	107	107	101	108	108	99
task50.3	115	115	115	116	115	115	111
task50.4	92	89	89	84	89	89	84
task50.5	92	89	89	85	89	89	85
task100.1	187	184	185	182	185	185	181
task100.2	183	179	179	173	179	180	172
task100.3	182	175	175	176	175	175	175

6.4 Genetized-Knowledge Genetic Algorithm: gkGA

In this section, we propose a novel strategy for constructing a priority list and its execution. This strategy is amalgam of the genetic algorithm as an execution engine and a set of multiple heuristics as the knowledge pool, called the genetized-knowledge GA and in short, gkGA.

6.4.1 Basic Idea of the gkGA

In Sect. 6.3, we have proposed reinforced GAs for the multiprocessor scheduling problems. In each reinforced GA, one string conversion scheme based on a heuristic is applied to the strings for converting into a priority list. We have observed that performance of four schemes is problem dependent and that there could not be found uniformly best. The new strategy to be proposed in this section is based on automatic selection of a string conversion scheme best suited for a given problem, realized by the genetic adaptation mechanism of GA. That is, information on heuristics also is represented as genes and even heuristics is evolved by hybridizing multiple heuristics during the GA execution. This idea of the new strategy is applicable to not only the multiprocessor scheduling problems but also other GA applications.

In the method proposed in Section 6.3, a string is composed of a sequence of the task identifiers. We prepare three binary genes and attach them before the string representation described in Section 6.3 to represent four heuristic schemes O, A, B, and C. We call the first three genes of the strings "heuristic substrings" and the rest "task substrings." The genes in a heuristic substring corresponds to Scheme A, B, or C from left to right, respectively. For example, 100-xxxx..xxx corresponds to the priority list converted by applying Scheme A to the string xxxx..xxx. The substring 000- corresponds to Scheme O. Furthermore, we introduce here hybridizing of heuristics. The string 110- means applying hybrid heuristics of Scheme A and B, say Scheme AB. We hope that such hybridized heuristics may convert to a more appropriate priority list than a simple one. The following shows how to convert a string into a priority list by hybrid heuristics.

«Converting Strings into Priority Lists by Hybridized Heuristics»

Let the heuristic substrings = $x_a x_b x_c$-.

1° Decompose a task substring s into substrings $s1$, $s2$, ..., sk such that the genes in s_i have the depth i. Note that $s=s1 \mid s2 \mid ... \mid sk$, where " | " denotes concatenation. And put the genes of s_i into a queue q_i from the left to right, for $1 \le i \le k$.

2° Make P empty.

3° Choose the top of each q_i, $1 \le i \le k$ whose predecessors are already assigned.

4° Sort the chosen tasks by descending order of the execution time and for each T_j, define the score, $St(T_j)$, as the position of T_j in the above sort. And sort them by descending order of the number of the immediate successor tasks and define the score, $Ssuc(T_j)$, as the position of T_j in the previous sort. And sort them by descending order of the number of the time depth and define the score, $Std(T_j)$, as the position of T_j in the previous sort. Let $SCORE(T_j) = x_a * St(T_j) + x_b * Ssuc(T_j) + x_c * Std(T_j)$.

5° Append the smaller $SCORE(T_j)$ task, T_j, among the tasks chosen at 3° to the tail of Q. Remove T_j from the associated queue. When tie occurs, the deeper task first.

6° Unless every substring is empty, return 3°.

Note that if the heuristic substring = 000, then the scheme behaves as Scheme O.

6.4.2 Genetic Operations

We define genetic operations for new strings in the new strategy. In the new strategy, different operations for both substrings are defined. For task substrings, we use the same operations as the method described in Section 6.3. Therefore, here we define them only for heuristic substrings.

(1) Initial Populations

Initial population is randomly generated such that only one or no bit should be "1" in the heuristic substrings and for the task substrings, follow (2) in Section 6.3.1.

(2) Crossover for heuristic substrings

The following two -points crossover[16] is used for heuristic substings.

«Two-points Crossover»

1° Choose two strings, say $P1$ and $P2$, randomly from the GA pool.

2° Determine randomly two crossover points $q1$ and $q2$ on the heuristic substring. These two may not be distinct.

3° Exchange the genes between $q1$ and $q2$ in $P1$ and $P2$.

4° We now have new substrings $C1$ and $C2$.

Example 6.6: Let $P1$ = 100- and $P2$ = 010-, corresponds to Scheme A and B, respectively. If $q1$ =1 and $q2$ =2 then we have $C1$ = 110- and $C2$ = 000-, they correspond to AB and O, respectively.

(3) Mutation for heuristic substrings

The mutation is defined as follows:

«Mutation for Heuristic Substrings»

1° Choose a mutation point q randomly on the heuristic substring.

2° The bit of the q-th gene is reversed.

Now we have the description of the new strategy as follows:

«gkGA»

1° Initialization: An initial population is randomly generated by (1) in Sect. 6.4.2.

2° Evaluation of the fitness function: The fitness value of each string is calculated according to the fitness function (5) in Section 6.3.3. (A string is converted into a priority list by the scheme in Sect 6.4.1.)

3° Genetic operations for task substrings: For randomly selected strings, the crossover (3) in Section 6.3.1 and/or the mutation (4) in Section 6.3.1 is applied.

4° Genetic operations for heuristic substrings: For randomly selected strings, the crossover (2) in Sect. 6.4.2 and/or the mutation (3) in Sect. 6.4.2 is applied.

5° Reproduction: The population for the new generation is generated by the roulette wheel selection[26].

6° If time is up, stop and return the best string, if not, go to 2°.

6.4.3 Experimental Evaluation

We evaluate the new strategy by experiment. The parameters of GA are set as follows:

The number of processors = 4, 6,
The number of populations = 50,
The generation span = 50,
The crossover probability for task substrings = 1.0,
The mutation probability for task substrings
= 0.03, 0.06,
The crossover probability for heuristic substrings = 0.0,
The mutation probability for heuristic substrings = 0.0.

We set the probabilities for heuristic substrings to 0.0. In this case, no hybridizing of multiple heuristics takes place, that is, Schemes AB, BC, ABC, etc., were not generated but only O, A, B, and C. In preliminary experiment, we set the probabilities for heuristic substrings to some positive real numbers, and then we got the almost same result as the case 0.0 with more computation time. However, we guess for more complex and larger problems effectiveness of the operations for the heuristic substrings appears clearly.

Tables 6.1, 6.2, 6.3, and 6.4 include the results, not described in Sect. 6.3. We can observe that the new strategy obtains uniformly best schedules for every instance of the problem, which should be effectiveness of genetizing heuristics.

6.5 Concluding Remarks

We have proposed a genetized-knowledge genetic algorithm (abbreviated as gkGA), and discussed its application to multiprocessor scheduling problems. In this approach, the heuristic itself is represented by genes, and by means of GA selection superior heuristics survive for a given problem, while others die out during the selection process. Moreover, by the crossover of heuristic genes themselves, hybridized heuristics can also be generated. This is a novel strategy in the GA field of combinatorial problems; although the idea of reinforced GAs was already proposed, heuristics have not yet been explicitly genetized. The effectiveness of our proposed strategy for multiprocessor scheduling problems has been proved through computer evaluation. As a future problem, we have experimental evaluation of the gkGA with

non-zero probabilities for heuristic substrings for more complex and larger problem instances.

Chapter 7

Concluding Remarks

We have considered three kinds of resource assignment type problems, the mutual exclusion problem in distributed environments, the stable marriage problem, and the multiprocessor scheduling problem. New results have been derived using net theory and genetic algorithm techniques.

First, we treated the mutual exclusion problem in distributed environments in Chapters 3 & 4.

In Chapter 3, we considered a mutual exclusion problem in an autonomous distributed environment. The environment is characterized by autonomy of the node, the variability of the network structure, and needs of local message exchange for information gain. We have designed a dynamic mutual exclusion protocol, called Distributed MUTEX, based on the Chandy-Misra protocol, that is, the privilege of resource use is given in accord with the partial order relation induced by an acyclic graph logically set up by means of message token distribution. We have obtained the communication feasibility condition that makes mutual exclusion possible, namely that a Hamiltonian circuit must exist in the graph G representing the communication structure in order to be able to design the initial acyclic graph for mutual exclusion of a single shared resource. Moreover, we have proposed entry and exit protocols for individual nodes.

In Chapter 4, we discussed properties of the evolution of acyclic graphs. The mutual exclusion protocol proposed in Chap 3, Distributed MUTEX, is based on the evolution of acyclic graphs. We have analyzed the concurrency of an acyclic graph evolution using graph theory, and investigated topological conditions for assuring the number of source nodes is less than or equal to k throughout the evolution by using a new notion, canonical circuit-cover of the communication graph G. We have shown that a decision problem CC(k), "for a given graph, is there a

circuit set in which every node belongs at least to one circuit from the set whose cardinality is not greater than k?", is NP-complete, and proposed a method to find a canonical circuit-cover using a genetic algorithm. Moreover, we have extended the Distributed MUTEX to the mutual exclusion problems for multiple resource cases.

The following items require future attention:

(1) Control of concurrency by other means than canonical circuit-cover.
Concurrency analysis should be performed, by using not only the notion of the canonical circuit-cover as we did in this dissertation but also, for example, by means of the independent set of the graph. Development of multiple synthesis techniques is beneficial for providing multiple design options coupled with various cost-performance criteria.

(2) Analysis of the periodicity of an acyclic graph evolution.
The period of an evolution trajectory represents how often each node is scheduled to fire. Given the number of shared resources, the initial acyclic set-up determines the entire course of the evolution trajectory starting from it, and hence the period is decidable. The quality of a given initial set-up is measurable by its period, shorter the better.

(3) Implementation of fault tolerance in the Distributed MUTEX.
Is it possible for the Distributed MUTEX to function correctly if a node/ link breaks down physically or logically? Fault tolerance is important for protocols in distributed environments. The analyses in Chapter 3 need to be extended and the fault tolerance mechanism should be to meet certain criteria added to.

(4) Development of network software for implementing the Distributed MUTEX.
We have only discussed theoretical aspect of this protocol. The development of software and its evaluation in a real system are next steps we are currently undertaking.

Secondly, we defined and treated the gender-fair stable marriage problem in Chapter 5. We have proposed a method for adapting a genetic algorithm to its solution. In this method we transform the problem into a graph problem, a closed subset sum problem, whose

decision version is shown to be NP-complete. The graph representation reflects the lattice structure of the stable matchings and facilitates GA application, by allowing easier encoding, and more effective definition of genetic operators. Computer experiment has confirmed the effectiveness of the GA solution.

Areas for future study include:

(5) Introduction of fuzzy theory into preference lists.

The preference list in the original problem is required to be strictly ordered. However, preference is seldom so well-defined, and is more fuzzy in nature. We should devise a design methodology of arriving at more natural preference by means of fuzzy theoretic measurements.

(6) Extension of the matching protocol to distributed environments.

In the distributed stable marriage problem, each member only possesses his or her own preference list. Members must find stable partners by message exchanges. This extension improves privacy and widens the range of possible applications.

(7) Parallelization of the GA execution.

A genetic algorithm is safely said to have opened up a new methodology for solving combinatorial optimization. Many papers have reported its superior efficiency over classical methods. However, GA execution is quite time consuming and its high computation overhead must be relieved. Parallelization of GA execution is one of the immediate cures.

Thirdly, we have considered multiprocessor scheduling problems and their GA solution in Chapter 6. We have proposed genetized-knowledge GA, gkGA. The heuristic technique itself is also represented as genes, and by means of GA selection superior heuristics for a given problem survive and others die out during execution. Moreover, by the crossover of heuristic genes, hybridized heuristic techniques can also be generated. This is a novel strategy in the GA field, extending the idea of reinforced GAs. Experimental evaluation proved the effectiveness of our proposed strategy.

Future extensions of this work include:

(8) More performance evaluations comparing with other conventional methods.

In addition to the comparison made here of gkGA with conventional approaches such as CP/MISF and Nakasumi's GA method, we should compare with other methods such as branch-and-bound searching.

(9) Application of the gkGA to other problems.

We have demonstrated the gkGA's uniformly better performance over a wide range of problem instances when applied to the multiprocessor scheduling problem, but the gkGA methodology is easily applicable to other combinatorial optimization problems.

Acknowledgment

First of all, I would like to express my sincere appreciation to Professor Isao Shirakawa of Osaka University for his accurate criticisms, invaluable suggestions, encouragements, and serving as the chair of my dissertation committee.

I am grateful to Professors Hiroaki Terada, Sadatoshi Kumagai, Norihisa Komoda, Shojiro Nishio, for serving as members on my dissertation committee.

I would like to express my sincere appreciation to Professor Kenji Onaga of University of the Ryukyus for his continuous guidance, accurate criticisms, and unceasing encouragements throughout this research. It would be impossible to accomplish this work without his invaluable guidances and helpful suggestions.

I would like to thank Professor Seiki Kyan of University of the Ryukyus for providing me with constructive suggestions, accurate criticisms, and continuous encouragements throughout this research.

I would like to express my deep gratitude to Professor Shoji Shinoda of Chuo University and Professor Toshimasa Watanabe of Hiroshima University for their invaluable advices and discussions.

I would like to thank Professor Tsuyoshi Kawaguchi of Ohita University and Professor Zensho Nakao of University of the Ryukyus for their instructive suggestions and warm encouragements.

Grateful thanks must be paid to those Professors and Staffs at the Departments of Electric and Electronics Engineering and Information Engineering of University of the Ryukyus, who have supported me throughout this work.

Furthermore, thanks are also due to all the members of the Professor Onaga's laboratory and the Professor Kyan's laboratory.

Finally, my great thanks are due to my mother for her support, and my family members and friends for their concerns and encouragements.

References

[1] Raynal, M., *Distributed Algorithms and Protocols*, John Wiley & Sons, 1992.

[2] Wirth, N., *Algorithms+Data Structures=Programs*, Prentice-Hall, Inc., 1976.

[3] Bellman, R., Esogbue, A. O., and Nabeshima, I., *Mathematical Aspects of Scheduling & Applications*, Pergamon Press, 1982.

[4] Maekawa, M., *Operating Systems*, Iwanami-Shoten, 1988 (in Japanese).

[5] Le Lann, G., "Distributed systems, towards a formal approach," *Proc. of IFIP Congress*, pp. 155-160, 1977.

[6] Lamport, L., "Time, clocks and the ordering of events in a distributed system," *Comm. ACM*, vol. 21, no. 7, pp. 558-565, 1978.

[7] Ricart, G. and Agrawala, A. K., "An optimal algorithm for mutual exclusion in computer networks," *Comm. ACM*, vol. 24, no. 1, pp. 9-17, 1981.

[8] Suzuki, I. and Kasami, T., "A distributed mutual exclusion algorithm," *ACM Trans. Computer Systems*, vol. 3, no. 4, pp. 344-349, 1985.

[9] Maekawa, M., "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," *ACM Trans. on Computer Systems*, vol. 3, no. 2, pp. 145-159, 1985.

[10] Akhil, K., "Hierarchical quorum consensus: A new algorithm for managing replicated data," *IEEE Trans. on Computers*, vol. 40, no. 9, 1991.

[11] Fuchi, T., "An improved $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," *IEICE Trans. on Information and Systems*, vol. J75-D-I, no. 5, 1992 (in Japanese).

[12] Kakugawa, H., Hujita, S., Yamashita, M., and Ae, T., "A protocol for distributed k-mutual exclusion," *IEICE Technical Report*, COMP91-28, 1991 (in Japanese).

[13] Chandy, K. M. and Misra, J., "The drinking philosophers problem," *ACM Trans. on Programming Languages and Systems*, vol. 6, no. 4, pp. 633-646, 1984.

[14] Davis, L. ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.
[15] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
[16] Kitano, H., *Genetic Algorithms* , Sangyo-tosho, 1993 (in Japanese).
[17] Gale, D. and Shapley, L. S., "College admissions and the stability of marriage," *American Mathematical Monthly*, vol. 69, pp. 9-15, 1962.
[18] Gusfield, D. and Irving, R. W., *The Stable Marriage Problem Structure and Algorithms*, MIT Press, 1989.
[19] Knuth, D. E., *Marriages Stables*, Les Presses de l'Universite de Montreal, 1976.
[20] Irving, R. W., Leather, P., and Gusfield, D., "An efficient algorithm for the 'optimal' stable marriage," *Journal of ACM*, vol. 34, pp. 532-543, 1987.
[21] Ibaraki, T., "Combinatorial optimization and scheduling problems: new algorithm and their perspective," *Journal of Society of Instrument and Control Engineers*, vol. 34, no. 5, pp. 340-346, 1995 (in Japanese).
[22] Kasahara, H., *Parallel Processing Technology* , Corona Publishing, 1991 (in Japanese).
[23] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1991.
[24] Lenstra, J. K. and Kan, A. H. G. R., "Complexity of scheduling under precedence constraints," *Operations Research*, vol. 26, no. 1, pp. 22-35, 1978.
[25] Gonzalez, M. J., "Deterministic processor scheduling," *Computing Surveys*, vol. 9, no. 3, pp. 173-204, 1977.
[26] Coffman, E. G., *Computer and Job-shop Scheduling Theory*, John Wiley & Sons, 1976.
[27] Kasahara, H. and Narita, S., "Practical multiprocessing scheduling algorithms for efficient parallel processing," *IEEE Trans. Computers*, vol. C-33, no. 11, pp. 1023-1029, 1984.
[28] Nakasumi, M., "A process scheduling by parallel genetic algorithm," *Proc. of Fifth Meeting of Special Interest Group on*

Parallel Processing for Artificial Intelligence, pp. 1-4, 1994 (in Japanese).

[29] Ozaki, H. and Shirakawa, I., *Theory of Graphs and Networks*, Corona Pub. Co., LTD., 1973 (in Japanese).

[30] Ore, O., *Theory of Graphs*, American Mathematical Society, 1962.

[31] Murata, T., "Petri nets: properties, analysis and applications," *Proc. of IEEE*, vol. 77, no. 4, pp. 541-580, 1989.

[32] Raynal, M., *Algorithms for Mutual Exclusion*, North Oxford Academic, 1985.

[33] Raynal, M., *Networks and Distributed Computation Concepts and Algorithms*, North Oxford Academic, 1985.

[34] Dijkstra, E. W., "Cooperation sequential processes, in programming languages," F.Genuys(Eds), *Academic Press*, pp. 43-112, 1965.

[35] Crocus (group name), *Systems d'exploitation des ordinateurs*, Dunod, 1975.

[36] Shaw, A. C., *The Logical Design of Operating Systems*, Prentice-Hall, 1974.

[37] Gottlieb, A., Lubachevsky, B. D., and Rudolph, L., "Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors," *ACM Toplas*, vol. 5, no. 2, pp. 164-189, 1983.

[38] Morris, J. M., "A starvation-free solution to the mutual exclusion problem," *Information Processing Letters*, vol. 8, no. 2, pp. 76-80, 1979.

[39] Reiter, R., "Scheduling parallel computations," *Journal of ACM*, vol. 15, no. 4, pp. 590-599, 1968.

[40] Sifakis, J., "Performance evaluation of systems using nets," *Net Theory and Applications, Lecture Notes in Computer Science,* vol. 84, pp. 307-319, 1980.

[41] Carller, J., Chretienne, P., and Girault, C., "Modeling scheduling problems with timed petri nets," in *G. Rosenberg, ed., Advance in Petri Nets, APN '84, Springer-Verlag, LNCS189*, pp. 62-84, 1985.

[42] Carller, J. and Cheretienne, P., "Timed petri net schedules," in *G. Rosenberg, ed., Advance in Petri Nets, APN'87, Springer-Verlag, LNCS34*, pp. 62-82, 1988.

[43] Magott, J., "Performance evaluation of concurrent systems using petri nets," *Information Processing Letters*, vol. 1, no. 1, pp. 7-13, 1984.

[44] Magott, J., "Performance evaluation of systems of cyclic sequential processes with mutual exclusion using petri nets," *ibid.*, vol. 21, no. 5, pp. 229-232, 1985.

[45] Magott, J., "New NP-complete problems in performance evaluation of concurrent systems using petri nets," *IEEE Trans. on Software Engineering*, vol. SE-13, no. 5, 1987.

[46] Ramamoorthy, C. V. and Ho, G. S., "Performance evaluation of asynchronous concurrent systems using petri nets," *ibid.*, vol. SE-6, pp. 440-449, 1980.

[47] Leu, D. and Murata, T., "Properties and applications of the token distance matrix of a marked graph," *Proc. IEEE International Symposium on Circuits and Systems*, pp. 1381-1385, 1984.

[48] Lue, D. and Murata, T., "On maximum concurrency in a decision-free concurrent systems," *Proc. of International Computer Symposium*, pp. 1065-1071, 1984.

[49] Leu, D., Lee, S. H., and Murata, T., "Finding optimum initial markings to achieve desired concurrency in a marked graph," *Proc. on 31st Midwest Symposium on Circuits and Systems*, pp. 1137-1140, 1988.

[50] Nakamura, M., Onaga, K., and Kyan, S., "Concurrency analysis of acyclic graph evolution driven by node firing," *Proc. of 7th IEICE Karuizawa Workshop on Circuits and Systems*, pp. 103-108, 1994.

[51] Tamaki, H., "Conbinatorial optimization by genetic algorithms," *Journal of Society of Instrument and Control Engineers*, vol. 34, no. 5, pp. 347-352, 1995 (in Japanese).

[52] Huppe, B. S. and Okitani, I., "Application of genetic algorithm to the manpower scheduling problem," *Trans. of IEE Japan*, vol. 114-C, no. 4, pp. 450-455, 1994 (in Japanese).

List of Publications by the Author

1. Papers accepted in Transactions

1. Kawaguchi, T., Nakamura, M., "An OR parallel execution scheme of prolog on loosely coupled multiprocessor systems," *IEICE Trans. on Information and Systems*, vol. J75-D-I, no. 6, pp. 380-384, 1992 (in Japanese).

2. Onaga, K., Nakamura, M, and Kyan, S., "Design of a dynamic mutual exclusion algorithm for a distributed network of autonomous nodes," *IEICE Trans. on Fundamentals*, vol. E76-A, no. 3, pp. 387-398, 1993.

3. Nakamura, M., Onaga, K., Kyan, S., and Shinoda, S., "An autonomous distributed mutual exclusion protocol for multiple network resources and its marked graph analysis," *Trans. of IEE Japan* , vol. 114-C, no. 9, pp. 898-906, 1994 (in Japanese).

4. Nakamura, M., Onaga, K., and Kyan, S., "Concurrency and periodicity analysis of acyclic-graph evolution driven by node firing", *IEICE Trans. on Fundamentals*, vol. E78-A, no. 3, pp. 371-381, 1995.

5. Nakamura, M., Onaga, K., and Kyan, S., "Sex-fair stable marriage problem and its GA solution", *ibid.*, vol. E78-A, no. 6, pp. 664-670, 1995.

2. Papers accepted in Conferences and Workshops

6. Kawaguchi, T., Nakamura, M., and Masuyama, H., "Asynchronous parallel execution of prolog on message passing multiprocessor systems," *Proc. of Joint Technical Conference on Circuits/Systems, Computers and Communications*, pp. 96-101, 1990.

7. Nakamura, M., Onaga, K., and Kyan, S., "Minimum cost path algorithms for networks with piece-wise linear time-varying travel costs and transit delays," *ibid.*, pp. 279-284, 1991.

8. Onaga, K., Nakamura, M., and Kyan, S., "A solution of mutual exclusion problems in autonomous decentralized networks," *Proc. of 5th IEICE Karuizawa Workshop on Circuits and Systems*, pp. 291-296, 1992 (in Japanese).

9. Nakamura, M., Onaga, K., and Kyan, S., "Polynomial time minimum cost path algorithms for acyclic networks of c-depth bridges with piece-wise linear time-varying travel costs and transit delays," *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 1879-1882, 1992.

10. Onaga, K., Nakamura, M., and Kyan, S., "Fault tolerance and entry & exit protocols for one shared resource mutual exclusion in decentralized autonomous networks," *Proc. of Joint Technical Conference on Circuits/Systems, Computers and Communications*, pp. 537-542, 1992.

11. Onaga, K. and Nakamura, M., "Design of social matching algorithms for distributed and autonomous environments," *Proc. of International Symposium on Autonomous Decentralized Systems*, pp. 345-350, 1993.

12. Nakamura, M., Onaga, K., and Kyan, S., "Minimum cost path algorithms for networks with piece-wise linear time-varying travel costs and transit delays," *Proc. of 6th IEICE Karuizawa Workshop on Circuits and Systems*, pp. 255-259, 1993.

13. Nakamura, M., Onaga, K., and Kyan, S., "A mutual exclusion algorithms for a distributed network of autonomous nodes," *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 2733-2736, 1993.

14. Onaga, K., Nakamura, M., and Kyan, S., "A protocol for social matching in distributed environments," *Proc. of Joint Technical Conference on Circuits/Systems, Computers and Communications*, pp. 857-862, 1993.

15. Nakamura, M., Onaga, K., and Kyan, S., "Firing concurrency of acyclic-graph evolution driven by node firing," *Proc. of 7th IEICE Karuizawa Workshop on Circuits and Systems*, pp. 103-108, 1994.

16. Nakamura, M., Onaga, K., Kinjo, H., and Kyan, S., "Sex-equal matching in stable marriage problems," *Proc. of Joint Technical Conference on Circuits/Systems, Computers and Communications*, pp. 555-560, 1994.

17. Nakamura, M., Shimabukuro, K., Onaga, K., and Kyan, S., "A solution for multiprocessor scheduling problems using genetic algorithm," *Proc. of 8th IEICE Karuizawa Workshop on Circuits and Systems*, pp. 407-412, 1995 (in Japanese).

18. Nakamura, M., Onaga, K., Kyan, S., and Silva, M., "A genetic algorithm for sex-fair stable marriage problems," Special Session on Combinatorics on Advanced CAS, *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 509-512, 1995.

19. Nakamura, M., Sunagawa, K., Onaga, K., and Kyan, S., "Design of initial acyclic graphs for k-MUTEX," *Proc. of Joint Technical Conference on Circuits/Systems, Computers and Communications*, pp. 774-777, 1995.

3. Technical Reports

20. Nakamura, M., Kawaguchi, T., and Kyan, S., "An OR parallel execution scheme of prolog on loosely coupled multiprocessor systems multiprocessor systems," *IEICE Technical Report*, CPSY90-20, pp. 47-52, 1990 (in Japanese).

21. Nakamura, M., Kawaguchi, T., and Kyan, S., "A method for prolog OR execution and its evaluation results," *Record of Joint Conference of Electrical and Electronics Engineers in Kyusyu*, no. 572, 1990 (in Japanese).

22. Nakamura, M., Kyan, S., and Onaga, K., "Minimum-cost path problems in time-varying network," *ibid.*, no. 1225, 1991 (in Japanese).

23. Sakima, A., Nakamura, M., Kyan, S., and Onaga, K., "A minimum-cost path algorithm for time-varying networks," *ibid.*, 1992 (in Japanese).

24. Sakima, A., Nakamura, M., Kyan, S., and Onaga, K., "A minimum-cost path algorithm for time-varying edge characteristic networks - time complexity analysis for pwc transit delays and pwl travel costs-," *Proc. of IEICE Fall Conference*, 1993 (in Japanese).

25. Arakaki, R., Kyan, S., and Nakamura, M., "Psuedonoise generator using dynamic number sieve," *ibid.*, no. A.205, 1993 (in Japanese).

26. Yamashiro, M., Nakamura, M., Onaga, K., and Kyan, S., "A mutual exclusion protocol in autonomous distributed environments," *Record of Joint Conference of Electrical and Electronics Engineers in Kyusyu*, no. 1340, 1993 (in Japanese).

27. Miyagi, O., Nakamura, M., Sakima, A., Onaga, K., and Kyan, S., "A minimum-cost path algorithm for time-varying edge characteristic networks," *ibid.*, no. 1341, 1993 (in Japanese).

28. Onaga, K., Nakamura, M., and Kyan, S., "A mutual exclusion protocol with privilege transfer function for autonomous distributed environments," *Proc. of SICE Symposium on Decentralized Autonomous Systems*, pp.201-206, 1993 (in Japanese).

29. Yamashiro, M., Nakamura, M., Sunagawa, K., Onaga, K., and Kyan, S., "Firing concurrency and periodicity of acyclic graph evolution," *IEICE Technical Report*, CPSY94-36, pp. 73-79, 1994 (in Japanese).

30. Miyagi, O., Nakamura, M., Onaga, K., and Kyan, S., "Classification of GA space," *Proc. of IEICE Fall Conference*, A5, 1994 (in Japanese).

31. Nakamura, M., Miyagi, O, Onaga, K., and Kyan, S., "A method using genetic algorithm for searching sex-fair stable matching," *ibid.*, A3, 1994 (in Japanese).

32. Yamashiro, M., Nakamura, M., Onaga, K., Kyan, S., "Periodicity analysis of acyclic graph evolution under unison firing mode," *ibid.*, A1, 1994 (in Japanese).

33. Nakamura, M., Onaga, K., Kyan, S., "Concurrency bound of acyclic graph evolution," *IEICE Technical Report*, CAS94-68, pp. 93-99, 1994 (in Japanese).

34. Kinjo, H., Nakamura, M., Maeda, M., Onaga, K., and Kyan, S., "A genetic algorithm for sex-fair stable marriage problems and its computational evaluation," *ibid.*, CAS94-92, pp. 61-67, 1995 (in Japanese).

35. Sunagawa, K., Nakamura, M., Yamashiro, M., Onaga, K., and Kyan, S., "Cuncurrency of acyclic graph evolution - initial acyclic graph design for the maximum concurrency k-," *ibid.*, CAS94-83, pp. 75-82, 1995 (in Japanese).

36. Shimabukuro, K., Nakamura, M., Arakaki, K., Onaga, K., and Kyan, S., "A method using genetic algorithms for multiprocessor scheduling problems," *ibid.*, CAS94-91, pp. 53-59, 1995 (in Japanese).

37. Miyagi, O., Nakamura, M., Onaga, K., and Kyan, S., "Improvement of a genetic algorithm for knapsack problems," *ibid.*, CAS94-82, pp. 67-74, 1995 (in Japanese).

38. Nakamura, M., Kinjo, H., Onaga, K., and Kyan, S., "Gale-Shapley based algorithm for distributed stable marriage problems," *Proc. of IEICE Spring Conference*, A393, 1995 (in Japanese).

39. Shimabukuro, K., Nakamura, M., Onaga, K., and Kyan, S., "A method generating priority lists using genetic algorithm," *ibid.*, A392, 1995 (in Japanese).

5. Submitted Papers

40. Nakamura, M., Onaga, K., and Kyan, S., "A new type solution to multiprocessor scheduling problems: genetized-knowledge genetic algorithm (gkGA)," submitted to Journal of Artificial Intelligence of Japan.

41. Nakamura, M., Miyagi, O., Onaga, K., and Kyan, S., "A hybrid genetic algorithm with reliefing unfeasible string for the 0-1

multiple knapsack problems," submitted to Journal of Artificial Intelligence of Japan (in Japanese).