STUDIES ON SOLVING METHOD OF SOME COMBINATORIAL PROBLEMS BY GA

by
Katsumi Hirayama

Course in Engineering of Social Development

at
TOTTORI UNIVERSITY
TOTTORI,JAPAN

March 1997

PREFACE

When we consider effective production and management systems, a lot of problems can be dealt with through combinatorial problems. Various combinatorial optimization problems applied to production and management systems have been studied in the field of Operations Research(OR). One typical OR methods is mathematical programming, which optimizes some function called an objective function, subject to some constraint equations. Combinatorial optimization problems are classified into the most difficult kind of problems in mathematical programming. Combinatorial optimization problems have a vast sums of combinations which can not be enumerated even with a super computer.

We consider two types of combinatorial optimization problems, which are categorized into two types of problems, One is to layout geometrical materials, which is called packing problem, and the other is to decide combination of natural number in Markov Decision Process(MDP). Packing problems can be applied to iron and steel industries, glass industries and paper industries. Many optimal control problems of stochastic systems can be formulated by MDP, e.g., queuing systems, reliability systems and reservoir operation systems.

Some techniques to give an optimal solution of a combinatorial problem have been developed. Dynamic programming, branch and bound, and integer programming are typical methods to solve combinatorial optimization problems. It is ,however, impossible to find an optimal solution of combinatorial problems in real world manufacturing problems within a reasonable time, even though appealing to above operational research techniques. Computational time and effort will grow exponentially with problem size which means the number of decision variables and the number of constraint equations. Almost combinatorial optimization problems are known as Non-deterministic Polynomial(NP) complete problems. NP complete problems are defined as the problem which cannot be given an optimal solution within a polynomial expression time.

We apply Genetic Algorithms (GAs) to these problems. GAs are a typical meta-heuristics and have been proposed by John Holland. GAs are search procedures to find an approximate or possibly an optimal solution, which are based on the mechanics of natural selection and natural genetics. Our GAs are categorized into Hybrid GAs which utilize the other optimization methods, greedy algorithms, heuristics and Policy Iteration Method (PIM).

Contents

Chapter 1.

INTRODUCTION

1.1. Combinatorial Optimization Problem

When we consider effective production and management systems, a lot of problems can be dealt with through combinatorial problems. Various combinatorial optimization problems applied to production and management systems have been studied in the field of Operations Research (OR). One typical OR method is mathematical programming, which optimizes some function called an objective function, subject to some constraint equations.

Combinatorial optimization problems are classified into the most difficult kind of problems in mathematical programming. Combinatorial optimization problems have vast sums of combinations which can't be enumerated even with a super computer. A combinatorial optimization problem is generally stated as follows,

$$\min_x f(x)$$

subject to $x \in F$

Here, variable x means a decision variable which is usually expressed as a permutation of natural numbers, and F is a set of feasible solutions and/or a set of permutations.

Some techniques to give an optimal solution of a combinatorial problem have been developed. Dynamic programming, branch and bound, and integer programming are typical methods to solve combinatorial optimization problems. It is, however, impossible to find an optimal solution of combinatorial problems in real world manufacturing problems within a reasonable time, even though appealing to the above operational research techniques. Computational time and effort will grow exponentially with problem size, which means the number of decision variables and the number of constraint equations. Almost all combinatorial optimization problems are known as Non-deterministic Polynomial (NP) complete problems. NP complete problems are defined as the problems which can't be given an optimal solution within a polynomial expression time.

1.1.1. Heuristics on Combinatorial Optimization Problems

Various heuristics are developed to find feasible solutions of combinatorial optimization problems. Heuristics are a powerful technique to give some feasible solutions, which are very difficult to find in problems

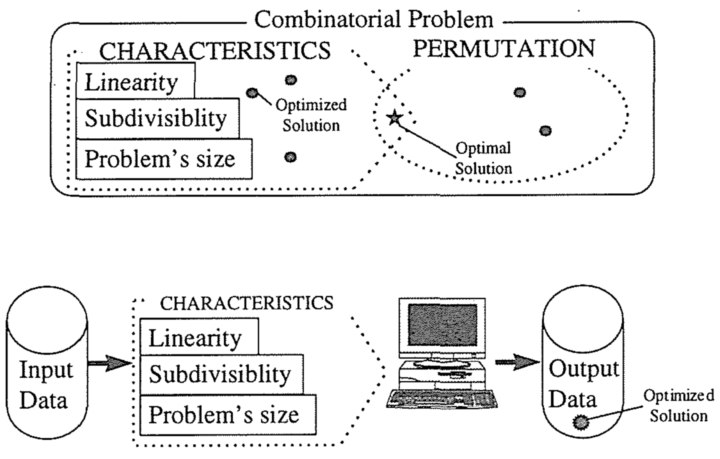


Figure 1.1 A concept of heuristics in combinatorial problem

of large size. Heuristics incorporate the experience of experts and utilize the problem's characteristics. Here, the problem's characteristics include the problem's size, linearity, subdivisiblity and so on. However, a feasible solution obtained by heuristics is not always an optimal solution, and is only an approximate solution which disregards objective criteria. Heuristics are what we call an expert system in the real world system.

Here, we define an optimal solution to be a feasible solution which maximizes or minimizes objective function. That is, there is no other feasible solution which is better than an optimal solution. On the other hand, we define an optimized solution to be an approximate solution in some sense. Figure 1.1 shows the relationship between a concept of heuristics and a combinatorial optimization problem.

1.1.2. Meta-Heuristics on Combinatorial Optimization Problems

Recently, meta-heuristics attract the attention of a great many researchers as a solving method of combinatorial optimization problems. Typical methods of meta-heuristics are neural-networks, simulated annealings and genetic algorithms. Meta-heuristics are a high rank concept of standard heuristics, which is an effective method when we are not satisfied with an optimized solution obtained by heuristics. Meta-heuristics have an advantage of utilizing standard heuristics and/or mathematical programming for local search. That is, meta-heuristics inherit the advantage of mere heuristics and/or mathematical programming. We are convinced that a concept such as meta-heuristics can make an application of software catch up with hardware. Meta-heuristics is the solving method of combinatorial optimization problems in the sense of global optimization.

Figure 1.2 shows a concept of meta-heuristics in a combinatorial problem.

1.1.3. Genetic Algorithms

Genetic Algorithms (GAs) are typical meta-heuristics [21]. They have been proposed by John Holland, whose research goals are to solve complex problems in many areas, including machine learning, the simulation of autonomous behavior and combinatorial optimization problems. GAs

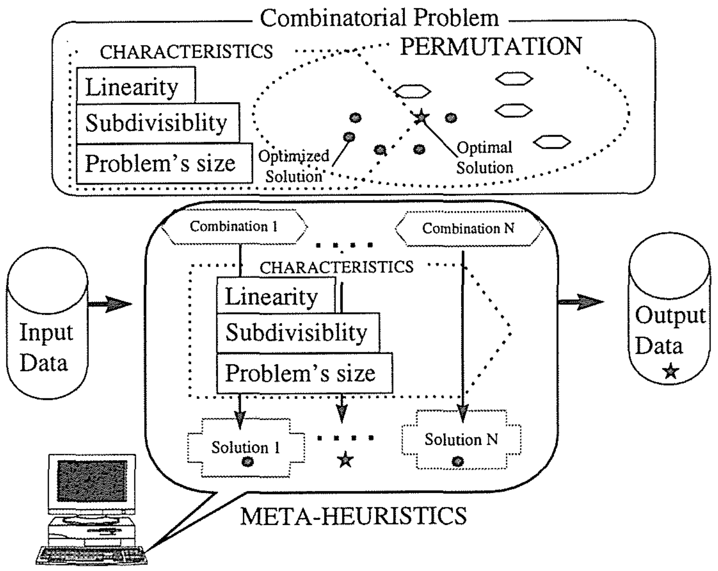


Figure 1.2 A concept of meta-heuristics in combinatorial problem

are search procedures to find an optimized solution, and they are based on the mechanics of natural selection and natural genetics.

GAs are algorithms of multi-start local searches which can deal with plural solutions simultaneously. GAs take advantage of utilizing usual heuristics and/or mathematical programming which are suitable to each problem. We particularly deal with what we call hybrid GAs which utilize usual heuristics and/or mathematical programming. This paper considers two types of combinatorial optimization problems, that is, packing problems and MDP problems, which are formulated by some combinatorial programming and proposed solving methods using hybrid GAs.

1.1.4. Geometrical Combinatorial Optimization Problems

Combinatorial optimization problems are categorized into two types of problems. One is to layout geometrical materials and the other is to decide combinations of natural numbers. Combinatorial optimization problems to layout geometrical materials are called packing problems. Packing problems can be applied to iron and steel industries, glass industries and paper industries. In these industries, a typical packing problem is how to distribute small rectangular pieces in a large rectangular piece. In this problem, it is very difficult to find even one feasible solution. There are various types of packing problems, that is, rectangular packing problems, knapsack problems and bin-packing problems. In this paper, we deal with rectangular packing problems and bin-packing problems in various combinatorial problems to layout geometrical materials. We propose solving methods of packing problems and bin-packing problems by using hybrid GAs. On applying GAs to packing problems, the most important thing is how to constitute genetic information. We propose a constituting method suitable for each packing problem.

1.1.5. Natural Number Combinatorial Optimization Problems

Typical combinatorial optimization problems to decide combinations of natural numbers are scheduling problems, traveling salesman problems, Markov Decision Process (MDP) problems and so on. We deal with MDP problems in various combinatorial problems to decide combinations of natural numbers. Many optimal control problems of stochastic systems can be formulated by MDP, e.g., queuing systems, reliability systems and reservoir operation systems. In this paper, we treat MDP with constraints, and propose solving methods using hybrid GAs.

MDP is constituted of state space, action space, one step transition probability and cost or reward associated with each state and action. This problem is to find a policy, that is, a combination of actions for each state, which optimize time average expected cost or reward in an infinite time horizon. MDP with constraints doesn't have any solving method for finding an optimal solution. In this paper, we propose a solving method of MDP with a constraint, and with multiple constraints by using hybrid GAs.

We first analyze MDP with a constraint as a general framework, and next analyze an optimal control problem of reservoir against drought as a real world problem.

1.2. Outline of Dissertation

Chapter 2 and Chapter 3 discuss packing problems, which are to decide geometrical combinatorials. Chapter 2 treats rectangular packing problems using hybrid GAs interwoven with greedy algorithms. Chapter 3 deals with two-dimensional bin-packing problems, which are so-called slab design problems in iron and steel industries. We apply hybrid GAs interwoven with heuristic algorithms to this problem. A rectangular packing problem's object is to minimize trim-loss of a large rectangle. On the other hand, a two-dimensional bin-packing problem's object is to minimize the number of large rectangular pieces. Those objective functions are different between rectangular packing problems and two-dimensional bin-packing problems. Therefore, the way of modeling and suitable local search are different for each problem. We propose different ways of interweaving GAs in Chapter 2 and Chapter 3. We propose hybrid GAs combined with greedy algorithms in Chapter 2, and hybrid GAs combined with heuristics in Chapter3.

Chapter 4 and Chapter 5 study MDP problems with constraints, which are to decide combinations of natural numbers. Chapter 4 investigates a solving method of a MDP with a constraint, proposes applying the Policy Iteration Method (PIM), and analyzes by applying hybrid GAs which combine PIM. Chapter 5 considers a MDP problem with multiple constraints, which is a single reservoir operation optimization problem with multiple reliability constraints. Chapter 5 extends a solving method of a MDP with a constraint. However, even a MDP with a constraint is too difficult to derive an optimized solution for. A single reservoir operation optimization model is a more realistic problem than a solving method of a MDP model with a constraint.

1.3. Historical Review of Literature

Combinatorial problems have been discussed by a great number of researchers since the middle of the nineteenth century. Here, we introduce some significant historical literature on packing problems and MDP problems.

There are various types of packing problems such as two-dimensional rectangular packing problems, three-dimensional cuboid packing problems, non-rectangular packing problems, strip-packing problems, bin-packing problems, and so on. We discuss two-dimensional rectangular packing problems and two-dimensional bin-packing problems as an application of GAs.

Two-dimensional packing problems are available in many real world problems. In regard to two-dimensional rectangular packing problems, Kantorovitch [1] published the earliest paper "Mathematical methods of organizing and planning production" in 1939. He extended one-dimensional packing problems to two-dimensional packing problems. Paull and Walter [7] proposed an application of linear programming to this problem in 1954. Their work can be applied only to very small-size problems. After that, many variants of the two-dimensional rectangular packing problem are proposed. Gilmore and Gomory [2] proposed to apply the column generation method to one-dimensional rectangular packing problems in 1961. Their success in one-dimensional problems stimulated research into the two-dimensional case.

In regard to bin-packing problems, D.S.Johnson [9] proposed Fast algorithms for bin-packing in 1974. After that, many researchers entitled two-dimensional rectangular bin-packing problems. Almost all of their algorithms were heuristics such as the Best-Fit algorithm, First-Fit-Decreasing algorithm and Best-Fit-Decreasing algorithm. B.S.Baker and E.G.Coffman [8] proposed Next-Fit-Decreasing on bin-packing in 1981. Moreover, Coffman, Garey and Johnson [10] surveyed Approximation algorithms for bin-packing problems in 1984. Packing problems have been surveyed by Dowsland [3] in 1992. In 1994, Ahmed el-bouri et al. [6] presented heuristics for two-dimensional bin-packing problems. In their paper, a heuristic algorithm combining priority rules with a restricted search procedure is presented for solving two-dimensional bin-packing problems.

Rectangular packing problems are also known as cutting stock problems. Here, cutting stock problems mean two-dimensional rectangular packing problems with guillotine cut constraint. After Gilmore's research, cutting stock problems attracted many industries where many shapes are cut from a piece of raw material. Wang [11] proposed two algorithms for constrained two-dimensional cutting stock problems in

1983. Tokuyama and Ueno [4] discussed the cutting stock problem for large sections in the iron and steel industries in 1985. Agrawal [5] proposed a method to minimize trim loss in cutting rectangular blanks of single size from a rectangular sheet using guillotine cuts in 1993.

The original form of a Markov Decision Process (MDP) was introduced by Bellman [12] in 1957. In 1960, an excellent guide book "Dynamic Programming and Markov Processes" was published by Howard [13]. He gave the Policy Iteration Method to find a stationary pure optimal policy. Each stationary pure policy is a combination of actions for each state. Hence, the problem to find an optimal pure policy is a kind of combinatorial optimization problem. MDP has received increasing attention during the last three decades. Up to the present time, however, only a limited amount of work has been done on MDP with constraints. Hordijk and Kallenberg [14] investigated MDP with multiple constraints in 1984. They analyzed the problem in the range of mixed policy and formulated it by linear programming. Beutler and Ross [15] considered MDP with single constraint and proposed a solving method by applying Lagrange multipliers in 1985.

Regarding a reservoir operation problem, since Moran's pioneering work [26] on 'Stochastic Reservoir Theory' in 1954, extensive research has been done to analyze the reliability performance of reservoir systems [27], [28]. In particular, remarkable progress has been made in the estimation of relevant indices. These indices specify the probability that water is available from the reservoir (Reliability index) and the expected duration of that following a drought (Expected Duration index). Hashimoto et al. [29] in 1982 proposed to use 'reliability', 'resiliency' and 'vulnerability' criteria for water resource system performance evaluation.

Hashimoto's indices cover the concepts of drought 'frequency', 'duration' and 'magnitude' but fail to deal explicitly with the inflow distribution. In order to introduce these indices into a stochastic programming model, a consistent theoretical basis for formulating indices is required. The problem was analyzed in the range of Mixed policy and was formulated by linear programming.

Little's pioneering work on reservoir policy appeared in 1955, which provided a basis for what is now called Stochastic Dynamic Programming (SDP). Since then, much work has been done to apply SDP to real world reservoir operation problems [30],[31]. Most of the work in this direction used a simple single-stage loss function which is assumed to be a decreasing function of reservoir release (an ordinary single stage loss function). A model of this type commonly takes the form of expected loss minimization. These models could not explicitly consider the indices mentioned above. As a result, those models tend to ignore the trade-off among the various reliability indices. To overcome this problem, several approaches have been developed. These include the one that treats 'chance' or 'reliability' of drought as a constraint [32], [33], [34], [35]. However, even chance/reliability constraint models could not take into account the duration of drought in an explicit manner.

Tatano et al. discussed that in their paper [36] giving a reservoir operation rule. The model was formulated in the form of a stochastic linear programming model which minimizes expected loss per period subject to two kinds of reliability constraints of drought frequency and expected drought duration. In that model, state variables were defined at maximum available amounts for release and occurrence of drought.

Chapter 2.

TWO-DIMENSIONAL RECTANGULAR PACKING PROBLEM

2.1. Introduction

This chapter describes a solving method for two-dimensional rectangular packing problems which are a kind of geometrical combinatorial problem. Two-dimensional rectangular packing problems are the packing of rectangular pieces (order plates) into a larger rectangular container (mother plate). Two-dimensional rectangular packing applications are required in industries such as glass, paper and metal in order to produce effectively. Two-dimensional rectangular packing problems are subdivided into two problem areas; that of determining a permutation of order plates' allocation while minimizing wastage (the trim-loss problem), and that of determining the number of mother plates and each mother plate's cutting pattern to meet a given order while minimizing the number of mother plates (the assortment problem).

In this chapter, we deal with trim-loss problems which are also called cutting problems. In a production system, order plates are produced by cutting patterns according to layout order plates on a mother plate. The trim-loss problem is dominated by permutation of order plates and a geometrical allocation on the plane. Therefore, it is important to determine which order plate is the best to pack into the mother plate next, and which allocation of points is the best to locate next.

However, it is impossible to calculate all permutations of order plates and feasible allocation points within a reasonable calculation time limit when the amount of order plates is large. Then, various heuristic algorithms are developed to solve two-dimensional packing problems. These heuristic algorithms consist of cutting pattern constraint, order plate's size and/or production rule-based algorithms. For example, one approach is proposed from a mathematical perspective to a production rule-based one, the other approach is to use heuristics consisting of ordering and placement rules for the required rectangular pieces. However, characteristics of production rules are different for each individual factory and production environment. Therefore, production systems which are made by heuristics are not general purpose systems, when the production environment changes.

There are several approaches to find an optimized solution in this problem. These approaches adopt predetermined theories or rules that are subject to the problem itself. It is very difficult to determine which order plates to allocate first. Moreover, the allocation position depends on packing order. Since this determination affects the whole rectangular packing performance, a two-dimensional rectangular packing problem is a combinatorial problem.

We consider that in the trim-loss problem it is possible to control two evaluation functions expressed by this problem's characteristics. We propose an approach which applies Hybrid GAs in order to guide a search process effectively and to obtain an optimized allocation. Genetic information in a trim-loss problem is expressed by the weighted coefficients of evaluation function. Permutation and placement of order plates is improved according to the progressing of genetic information. This method is more general-purpose than the usual rule based approach or heuristics. Kawakami and Kakazu [16] proposed that a three-dimensional packing strategy is acquired through a hierarchical tuning. We basically reference their hierarchical tuning method on applying GAs. However, they never discuss about two-dimensional rectangular packing problems and allocation direction of rectangles.

We consider the allocation direction of rectangular pieces on two-dimensional packing problems. Considering the allocation direction of rectangular pieces means whether it is permitted to revolve a rectangular piece 90 degrees or not. In case of permission to revolve the rectangular pieces 90 degrees, the number of rectangular piece's combinations is equal to the square number of primals. Moreover, the problem's characteristics are suddenly complicated because we must add the decision of whether the rectangular pieces are revolved 90 degrees or not. Therefore, we define whether the allocated rectangular pieces are revolved 90 degrees or not as the binary genetic information. This binary genetic information lets us find the optimized solution, even if the problem's size is large.

A numerical example was also presented to examine the computational efficiency of this proposed approach. Our new algorithm found an optimal solution in a few minutes by 200mips workstation .

2.2. Two-dimensional Rectangular Packing Problem

In the previous section we proposed that a two-dimensional rectangular packing problem is represented by two evaluation functions that dominate the packing procedure. In this section, we discuss modeling and procedure on two-dimensional rectangular packing problems. Two-dimensional packing procedure consists of two steps. First, the most suitable position in the mother plate is determined by a position evaluating function. Second, when an order plate is placed in a predetermined next allocation position, the most suitable order plate is chosen from an unallocated order plates' set according to an order permuting evaluating function. These two steps are executed recursively until allocation space satisfies the termination conditions. We define gene in GAs to be weighted coefficients which are controlled by those evaluation functions. Calculation results on two-dimensional rectangular packing problems largely depend on a combination of weighted coefficients of two evaluation functions. Our object is to give an optimized solution while minimizing trim loss by finding which weighted coefficients suit the problem.

Here, we define some notations about the size of the container and rectangular pieces, that is, the mother plate and order plates.

$$\text{Mother plate size:} \qquad (W, L) \tag{2.1}$$

$$i \text{ th order plate size:} \quad P_i = (w_i, \ell_i), \quad i = 1, 2, \cdots, n \tag{2.2}$$

Where, P_i is the i-th order plate evaluation value, w_i and ℓ_i are dimensions of the i-th order plate, and n is the number of order plates. First, we allocate a mother plate at initial point $(0, 0)$. Here, allocation spaces (x, y) are restricted respectively to $0 \leq x \leq W$, $0 \leq y \leq L$.

Moreover, objective function (i.e., fitness on GAs) is as follows:

$$R = \sum_{i=1}^{m} \left(\frac{w_i \cdot \ell_i}{W \cdot \mathrm{L}} \right) \tag{2.3}$$

That is, R means the area ratio which covered order plates in a mother plate, and m is the number of allocated order plates which is possible to allocate to an order plate. Our object is to choose a suitable order plate and to determined the order plate's allocation point and direction in a mother plate at each step, while maximizing the function R. That is, our object is to layout order plates in a mother plate while minimizing wastage (the trim loss).

2.3. Solving Method by Greedy Algorithm

We propose the greedy algorithm for determining a permutation of order plates and an allocation point. The greedy algorithm is a solving method which chooses the biggest element first in order to press the overall solution space, then the next biggest element in the solution space that is left. This algorithm presses solution space while choosing the most suitable solution according to progress in the search process. This search process is similar to a human's intellect. For example, when we pay an amount of money by using some kinds of coins and bills, we unconsciously use this algorithm.

We define evaluation function Q_i to measure i-th order plate, weighted coefficient e_k to control allocation and evaluation item f_k to evaluate the order plates. In this paper, we consider two evaluation items.

2.3.1. Evaluation Function to Choose a Rectangular Piece

Evaluation function Q_i, weighted coefficient e_k and the evaluation items f_k are as follows:

$$i^* = \arg\max_i Q_i, \tag{2.4}$$

$$Q_i = \sum_{k=1}^{2}(e_k \cdot f_k), \tag{2.5}$$

$$f_1(w_i, \ell_i) = \left(\frac{w_i}{W}\right)^2 + \left(\frac{\ell_i}{L}\right)^2, \tag{2.6}$$

$$f_2(w_i, \ell_i) = \left(\frac{w_i \cdot \ell_i}{W \cdot L}\right)^2. \tag{2.7}$$

Where evaluation item f_1 and f_2 show non-similarity and area ratio between mother plate and order plate, respectively. Function Q_i is a liner equation for f_k which is biased against weight coefficient e_k. In case e_1 is larger than e_2, evaluation item f_1 has strong effect on choosing non-similar rectangular pieces. In the opposite case, that is, e_1 is smaller than e_2, evaluation item f_2 has strong effect on choosing large-sized rectangular pieces. Therefore, this evaluation function allocates a rectangular piece which has the highest evaluation value.

2.3.2. Evaluation Function to Select an Allocation Position

We define evaluation function P_j which measures j-th order plate's feasible allocation positions. Evaluation function P_j chooses the most suitable allocation position (x^*, y^*) of a set of allocable positions (x_j, y_j). Here, an evaluation value P_j and the most suitable allocation position (x^*, y^*) are given as follows:

$$(x^*, y^*) = \arg\min_j P_j, \tag{2.8}$$

$$P_j = e_3 \cdot x_j^2 + e_4 \cdot y_j^2. \tag{2.9}$$

Where (x_j, y_j) is a set of the j-th feasible allocation positions, and e_3 and e_4 are weighted coefficients. Function P_j is a linear equation for a set of the j-th feasible allocation positions (x_j, y_j) which is biased against weight coefficient e_3, e_4.

When j-th rectangular piece with dimension (w_j, ℓ_j) is located at allocation point (x, y), we add two points, $(x + w_j, y)$ and $(x, y + \ell_j)$ to a set of allocation positions, then delete the allocation point (x, y) from an allocation position set. That is, the number of elements in j-th feasible allocation position set is j. Here, the first allocation position is $(0, 0)$.

Figure 2.1 shows the most suitable and feasible allocation points when j is 3. The first allocation point (x_1, y_1) is the lower left point O (0,0). When the 11th white order plate with dimension (w_{11}, ℓ_{11}) which is chosen by evaluation function Q_i is allocated at the first allocation point (0,0), the second allocation points (x_2, y_2) are $(0, \ell_{11})$ and $(w_{11}, 0)$. According to Figure 2.1, the 6th deep gray order plate is allocated at $(w_{11}, 0)$. Therefore, the third allocation points (x_3, y_3) are $(0, \ell_{11}), (w_{11}, \ell_6)$ and $(w_{11} + w_6, 0)$. According to Figure 2.1, the 8th light gray order plate is allocated at $(0, \ell_{11})$. The fourth allocation points (x_4, y_4) are $(0, \ell_{11} + \ell_6), (w_6, \ell_{11}), (w_{11}, \ell_6)$ and $(w_{11} + w_6, 0)$.

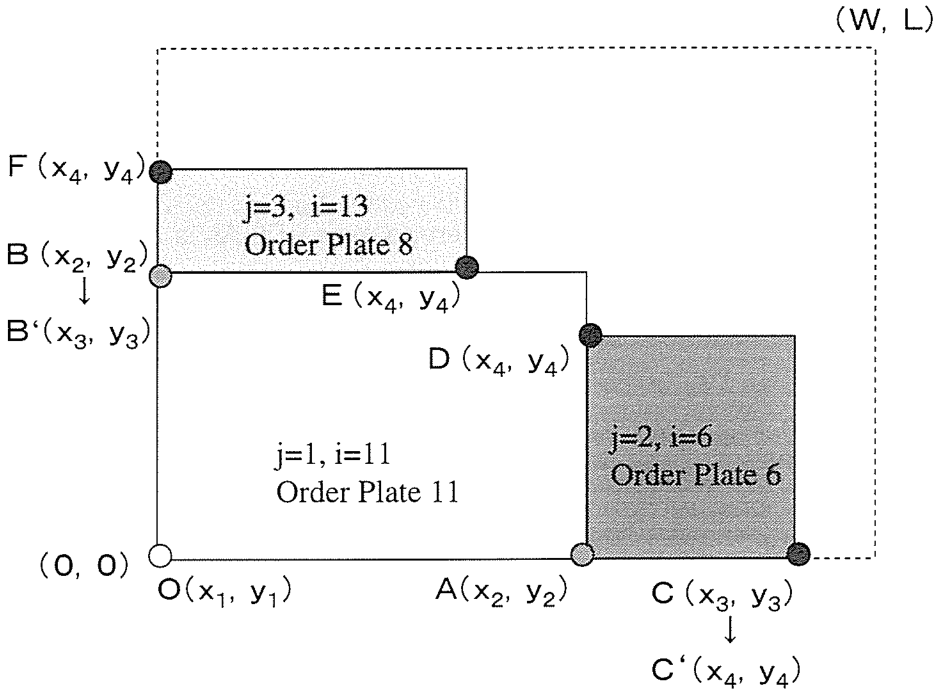


Figure 2.1 Allocation position

In case e_3 is larger than e_4, allocation points along the x-axis direction, while maximizing the evaluation item P_j, have strong effect on choosing. In the opposite case, that is, e_3 is small than e_4, allocation points along the y-axis direction have strong effect on choosing.

Accordingly, each parameter e_1, e_2, e_3 and e_4 is an element of combinatorial information on a two-dimensional rectangular packing problem. The different combination of those parameters leads to the different allocation results. An optimized solution in a two-dimensional rectangular packing problem is given by tuning those parameters while minimizing trim-loss.

2.4. Parameter Tuning Method by Genetic Algorithms

In this section, we propose a solving method of tuning parameters in greedy algorithm using GAs. Here, a gene of GAs is defined as a string which is lined with four parameters in greedy algorithm. Each parameter, e_1, e_2, e_3 and e_4 is expressed by the binary code (0 or 1). Population of GAs consists of the different genes, that is, the different parameter sets. Figure 2.2 shows population, that is, the gene's set which expresses four parameters as sixteen bits. According to Figure 2.2, e_1 of string 1 is 0101 in binary code. 0101 means $2^3 \times \underline{0} + 2^2 \times \underline{1} + 2^1 \times \underline{0} + 2^0 \times \underline{1} = 5$. We normalize 0101 to $\frac{5}{16}$, therefore $e_1 = \frac{5}{16}$. Similarly, $e_2 = \frac{10}{16}, e_3 = \frac{10}{16}$ and $e_4 = \frac{4}{16}$. As a result, string1 means allocation information which has weighted coefficients, $e_1 = \frac{5}{16}, e_2 = \frac{10}{16}, e_3 = \frac{10}{16}$ and $e_4 = \frac{4}{16}$. Similarly, other strings respectively mean different allocation information:

Genes	e_1	e_2	e_3	e_4
String 1	0101	1010	1010	0100
String 2	1101	0101	0000	0010
⋮	⋮	⋮	⋮	⋮
String n	1101	1110	1110	0100

Figure 2.2 A concept of population

Here, we explain the binary code which determined whether pieces revolved 90 degrees or not as follows:

$$j = \begin{cases} 0 & \text{same as primal data} \quad (w, \ell) = (w_i, \ell_i), \\ 1 & \text{revolved 90 degrees} \quad (w, \ell) = (\ell_i, w_i). \end{cases} \tag{2.10}$$

Figure 2.3 shows population, that is, the gene's set which expresses allocation information and allocation direction whether revolving 90 degrees or not. According to Figure 2.1 and Figure 2.3, String 1 shows that the first chosen order plate $i = 11$ is allocated the same direction as primal data (w_{11}, ℓ_{11}), the second chosen order plate $i = 6$ is allocated with dimension (ℓ_6, w_6) which revolved 90 degrees and the third chosen order plate $i = 13$ is allocated the same direction as primal data (w_{13}, ℓ_{13}). Here, we must set m to a number large enough to allocate order plates. Similarly, other strings respectively mean different allocation and direction information:

Genes	e_1	e_2	e_3	e_4		123	...	i	...	m
String 1	0101	1010	1010	0100		010	...	0	...	1
String 2	1101	0101	0000	0010		101	...	1	...	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	+	$\vdots$	$\vdots$	0	$\vdots$	0
String n	1101	1110	1110	0100		011	...	0	...	1

Figure 2.3 A concept of population

2.5. Numerical Experiment

Based on the proposed method described above, some numerical experiments were carried out on the two-dimensional rectangular packing problem.

2.5.1. Experimental Data

We prepared two types of data, data 1 and data 2 which are generated by random and artificially made. Data 1(Table 2.1) is created by random values, and data 2(Table 2.2) is created artificially. The number of data 1 and data 2 order plates are 71 and 31, respectively. Data 1 has more than enough plates to cover a mother plate. However, the amount of order plates' area in data 2 is the same as the mother plate's area.

Table 2.1 Data 1

i	w_i	ℓ_i	pieces	i	w_i	ℓ_i	pieces
1	1	4	2	16	4	3	3
2	2	5	1	17	3	6	3
3	3	4	3	18	4	3	1
4	2	3	3	19	1	4	3
5	5	6	3	20	6	5	1
6	6	5	2	21	5	2	2
7	3	4	1	22	2	1	1
8	6	3	3	23	1	2	3
9	1	2	3	24	4	1	3
10	4	3	3	25	1	4	2
11	3	2	3	26	2	1	3
12	6	1	3	27	5	6	2
13	1	6	2	28	4	5	2
14	2	3	3	29	1	4	1
15	3	6	1	30	4	1	3

2.5.2. Experimental Types of GAs

We tried three types of GAs - GA1, GA2 and GA3 - on two types of data. Each GA is shown as follows:

Table 2.2 Data 2

i	w_i	ℓ_i	pieces	i	w_i	ℓ_i	pieces
1	7	4	1	11	1	1	2
2	7	3	2	12	2	1	2
3	10	3	1	13	3	1	2
4	14	2	2	14	4	1	2
5	16	2	1	15	5	1	2
6	18	2	1	16	6	1	2
7	20	2	1	17	7	1	2
8	4	2	1	18	8	1	2
9	6	2	1	19	9	1	2
10	8	2	1	20	10	1	1

- GA1

  GA1 is simulated by Greedy Algorithm and GAs which have weighted coefficients for Greedy Algorithm.

- GA2

  GA2 is simulated by Greedy Algorithm using GAs which have weighted coefficients for Greedy Algorithm. GA2 prepares dummy data exchanging width to length, and chooses the most suitable rectangle from both primal data and dummy data.

- GA3

  GA3 is simulated by Greedy Algorithm using GAs which have weighted coefficients for Greedy Algorithm and binary genetic information on whether to permit 90 degrees revolution or not. Both genetic information independently execute genetic operation, crossover and mutation.

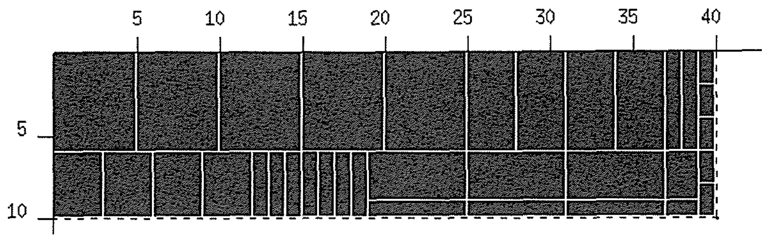


Figure 2.4 GA1 Allocation at generation 0

2.5.3. Experimental Results

The allocation process for each generation from GA1 to GA3 is shown in Figure 2.4 to Figure 2.17. Here, population size is 20 and generation is 50. We show experimental results and the processes of each GA as follows:

- Results of GA1

  In GA1, data 1 achieved objective function (area ratio) $R = 100\%$ at initial generation, however, data 2 didn't achieve $R = 100\%$ at even generation 50. Figure 2.4 indicates that GA1 could get one optimized solution from data 1. Here, data 1 has several kinds of optimized solutions. It is seen that Figure 2.4 could pack well from the bigger order plate to the smaller order plate.

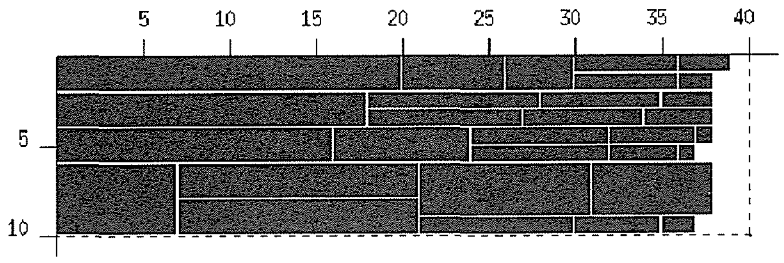


Figure 2.5 GA1 Allocation at generation 0

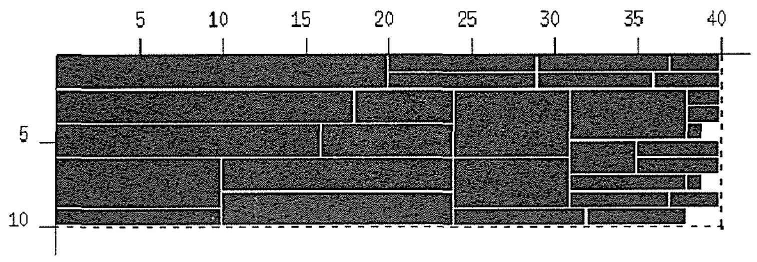


Figure 2.6 GA1 Allocation at generation 50

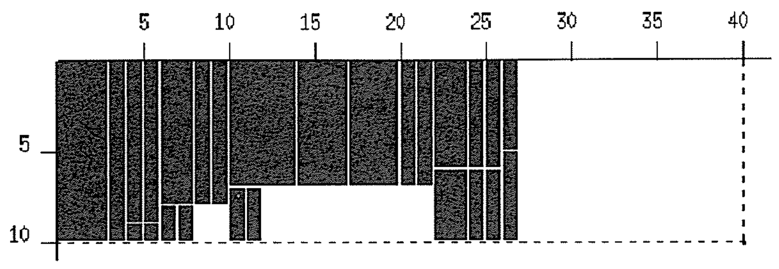


Figure 2.7 GA1 Allocation at generation 0

However, data 2 doesn't have a plural number of optimized solutions. Figure 2.5 and Figure 2.6 show GA1's allocation of results at generation 0 and 10, respectively. Results of GA1 on data 2 were $R = 94.8\%$ at generation 0 and $R = 99.0\%$ at generation 50. There are many differences between Figure 2.5 and Figure 2.6. These results show GA1 tried to search various combinations of allocation, while maximizing objective function.

Figure 2.7 shows the results when it exchanges width and length of primary data. This experiment didn't improve the allocation of order plate from initial generation. It is seen that data 2 is more difficult than data 1.

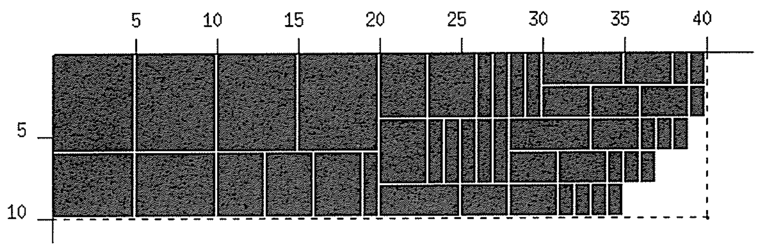


Figure 2.8 GA2 Allocation at generation 0

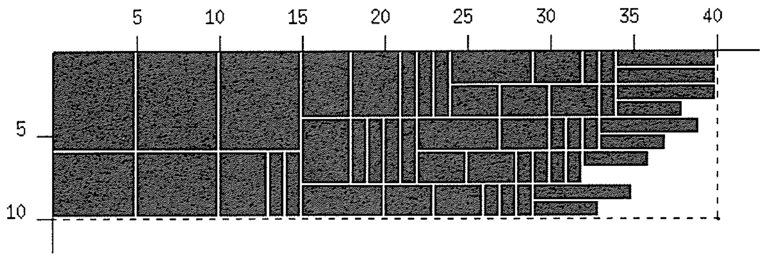


Figure 2.9 GA2 Allocation at generation 10

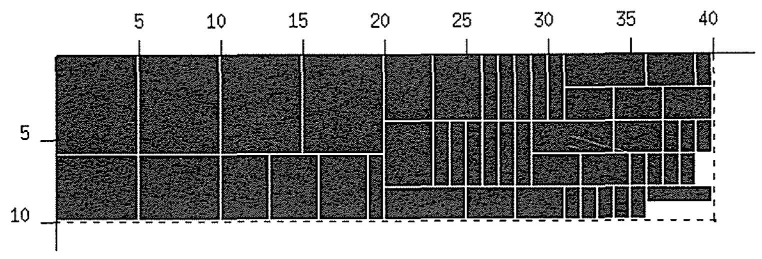


Figure 2.10 GA2 Allocation at generation 50

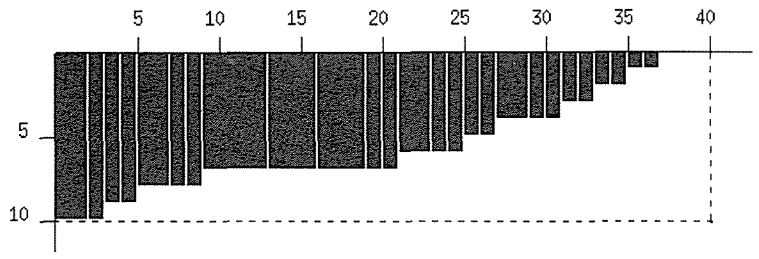


Figure 2.11 GA2 Allocation at generation 0

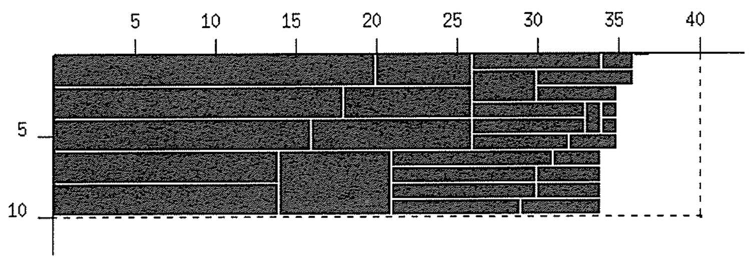


Figure 2.12 GA2 Allocation at generation 21

- Results of GA2

  Results of objective function of GA2 on data 1 were $R = 92.5\%$ at initial generation, $R = 95.5\%$ at generation 10 and $R = 98.5\%$ at generation 100. Each allocation process of GA2 on data 1 is given in Figure 2.8 to Figure 2.10. Those results show that GA2, which permitted revolving the order plate 90 degrees, is more difficult than GA1, which didn't permit revolving, since GA2 has two times more feasible order plates than GA1. It means that the number of the next choice of order plate which is chosen by evaluation function Q_i is equal to the square number of GA1. Therefore, GA2 on data 2 allocated smaller order plates than GA1 on data 1 when j was small, then couldn't allocate the bigger order plates later.

Results of the objective function of GA2 on data 2 were $R = 41.0\%$ at initial generation, $R = 73.0\%$ at generation 6 and $R = 90.0\%$ at generation 21. Each allocation process of GA2 on data 2 is given in Figure 2.11 and Figure 2.12. In data 2, the order plate's width is longer than its length. Moreover, evaluation function Q_i, which has the same weighed coefficient, tends to choose similar order plates. Figure 2.11 would indicate that GA2 evaluates order plates which revolved at 90 degrees as a higher score of Q_i at initial generation. Then, wider order plates (revolving 90 degrees) were chosen faster at initial generation. However, longer order plates were chosen faster at generation 6. Direction of order plates is similar in both cases. These processes seem to indicate that GA2 is not a more effective algorithm than GA1. However, GA1 can't allocate to revolve the order plate at 90 degrees. It means GA1 can solve only small problems for the two-dimensional rectangular packing problem. GA2 can solve complicated problems, but it is not a more effective algorithm than GA1 on data 1.

- Results of GA3

  GA3 has objective function $R = 99.5\%$ at initial generation, then achieves $R = 100.0\%$ at generation 1. This results shows that GA3 is a more effective algorithm than GA2. Each allocation process of GA3 on data 2 is given in Figure 2.13 and Figure 2.14.

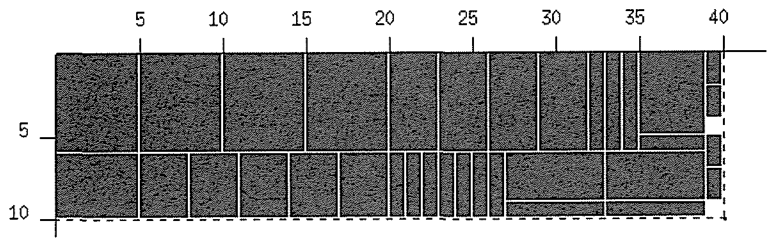


Figure 2.13 GA3 Allocation at generation 0

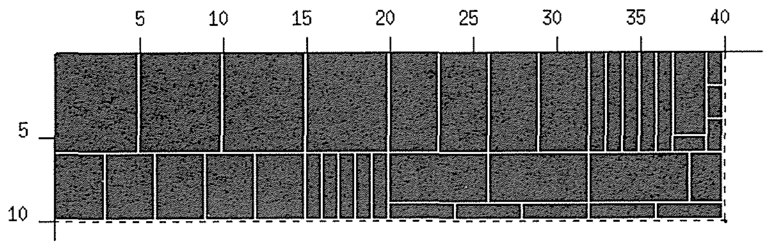


Figure 2.14 GA3 Allocation at generation 1

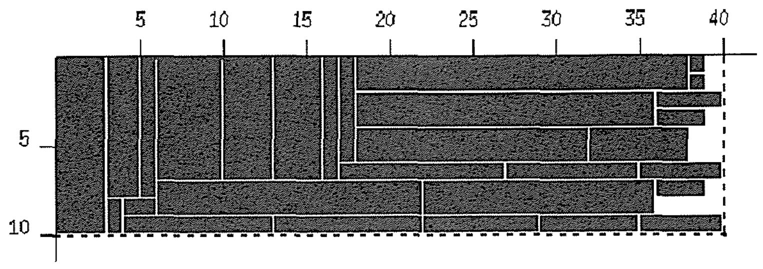


Figure 2.15 GA3 Allocation at generation 0

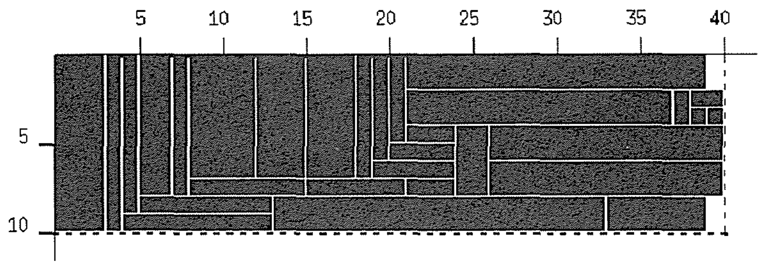


Figure 2.16 GA3 Allocation at generation 9

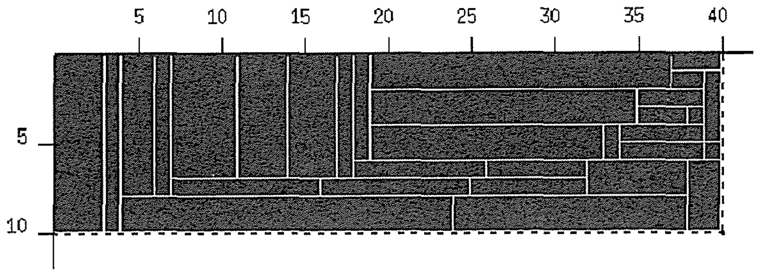


Figure 2.17 GA3 Allocation at generation 15

Results of the objective function of GA3 on data 2 were $R = 97.0\%$ at initial generation, $R = 99.0\%$ at generation 9 and $R = 100.0\%$ at generation 15. Each allocation process of GA3 on data 2 is given in Figure 2.15 to Figure 2.17. GA3 has more complexity allocation than GA1 or GA2. These processes would indicate that gene whether the gene revolving order plates 90 degrees or not works effectively.

2.6. Conclusion

In this paper, we proposed Hybrid Genetic Algorithms for the solving method of two-dimensional packing problems. Hybrid Genetic Algorithms have two kinds of genetic information; one is weighted coefficients of evaluation functions for determining allocation position and choosing a rectangular piece, the other is allocation of direction whether to revolve order plates 90 degrees or not. Our approach indicates that Genetic Algorithms have automatic tuning mechanisms for two-dimensional packing problems, moreover, improvement of genetic information is effective according to the characteristics of the problem. Some experimental results show the usefulness of our approach.

Chapter 3.

SLAB DESIGN PROBLEM

3.1. Introduction

Recent tendency in heavy industries is toward smaller lot sizes, shorter delivery times and higher grades. Under this tendency, the effective production system which attains a high-assortment yield ratio within a reasonable time is required.

When we consider the effective production system, a lot of problems must be dealt with through the bin-packing problems. The bin-packing problems are one of the typical packing problems which are categorized into geometrical combinatorial optimization problems. For example, a bin-packing application which packs rectangular pieces into a large rectangle is required by industries such as glass, metal and circuit layout application in LSI design.

The general bin-packing problem assumes a collection of equal capacity bins and a list of pieces which are to be packed into the bins subject to the requirement that the capacity of no bin be exceeded. In regard to the bin-packing problem, D.S.Johnson [9] published the paper "Fast algorithms for bin-packing" in 1974. After that, many researchers tried to study the various bin-packing problems. Many of these variations are surveyed by Coffman, Garey and Johnson [10]. However, almost all former bin-packing solving methods in a real industrial systems are heuristics, because the size of the bin-packing problem in a real industrial system is large enough to solve and has complexity to satisfy many constraints.

A class of problems with a slightly different objective is two-dimensional bin-packing problems. The two-dimensional bin-packing problems are usually a set of bins of fixed width and height with the objective of fitting the given pieces into a minimum number of bins. In regard to the two-dimensional bin-packing problem, Coffman and Shor [17] studied the distinction in 1990.

In this chapter, we deal with the two-dimensional bin-packing problems, which can be dealt with through a slab design problem in iron and steel industries. Slab design is an important problem to improve the production system in iron and steel industries.

We tried to apply the slab design problem to hybrid GAs which combined with heuristics for two-dimensional bin-packing problems. We show some results that hybrid GAs exceeded the former algorithms at generation 30, and raised yield ratio 0.6%. Moreover, this system applied by hybrid GAs works good enough to run on the available time.

3.2. Slab Design Problem

In iron and steel industries, H.Saito et.al. [19] proposed the two-dimensional assortment problem in the production of heavy steel plate in 1988. Moreover, B. Watson et.al. [20] proposed the rectangular cutting-stock problem by genetic algorithm in 1992. However, their problem is not a bin-packing problem but a cutting-stock problem. A cutting-stock problem is easier than a bin-packing problem, because a cutting-stock problem doesn't require packing all of the order plates into mother plates. The difficulty of the bin-packing problem is to allocate all of the order plates into mother plates. In the previous chapter, we showed that data 1, which has more than enough order plates to cover a mother plate, is easier than data 2 whose amount of order plates' area is the same as a mother plate.

Though the slab design problem is often known as a cutting stock problem, we define slab-design problems as two-dimensional bin-packing problems.

For heavy steel plates' effective production planning and management system, first of all, orders of the same thickness and specification required from customers are grouped into a lot. Then, the size of the rolled mother plates is determined. Finally, the slabs' dimensions are

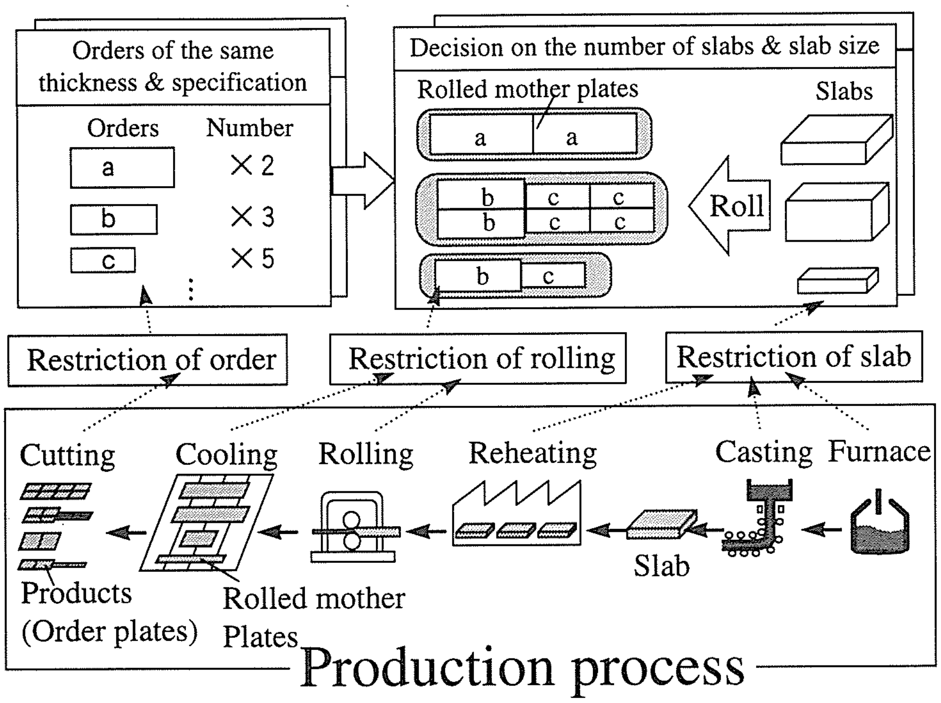


Figure 3.1 A concept of slab design problem

determined for each rolled mother plate. This procedure is called "Slab Design". The top of Figure 3.1 shows a concept of slab design.

The bottom of Figure 3.1 shows the production process in the iron and steel industries. In this process, slabs mean the intermediate material of rolled plates, and are usually produced by a casting machine.

Then, the slabs are heated in a reheating furnace to a set temperature. After being heated, the slabs are milled to mother plates of pre-determined thickness, width and length by a rolling mill. After that, mother plates are transferred to the finishing line and cut in longitudinal and transverse directions according to the cutting pattern of each mother plate, and divided into small plates.

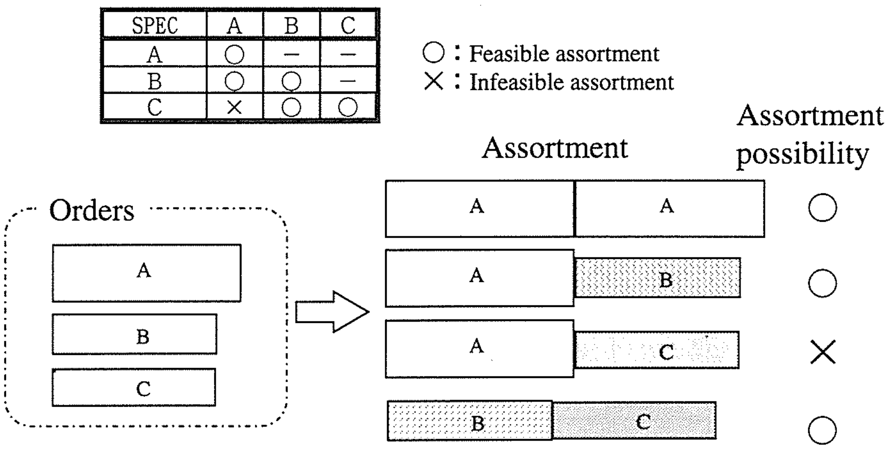


Figure 3.2 Restriction of order

3.3. Production Process of Heavy Steel Plates

The production procedures of slab design lead to three constraints, restriction of order, rolling, and slab. Each restriction is classified as follows.

3.3.1. Restriction of Order

Figure 3.2 shows feasible assortment. Basically, order plates with the same thickness and specification are grouped into a lot. Here, specification means the reheating temperature, quality of material, and cutting equipment which the customers required. However, there are some dif-

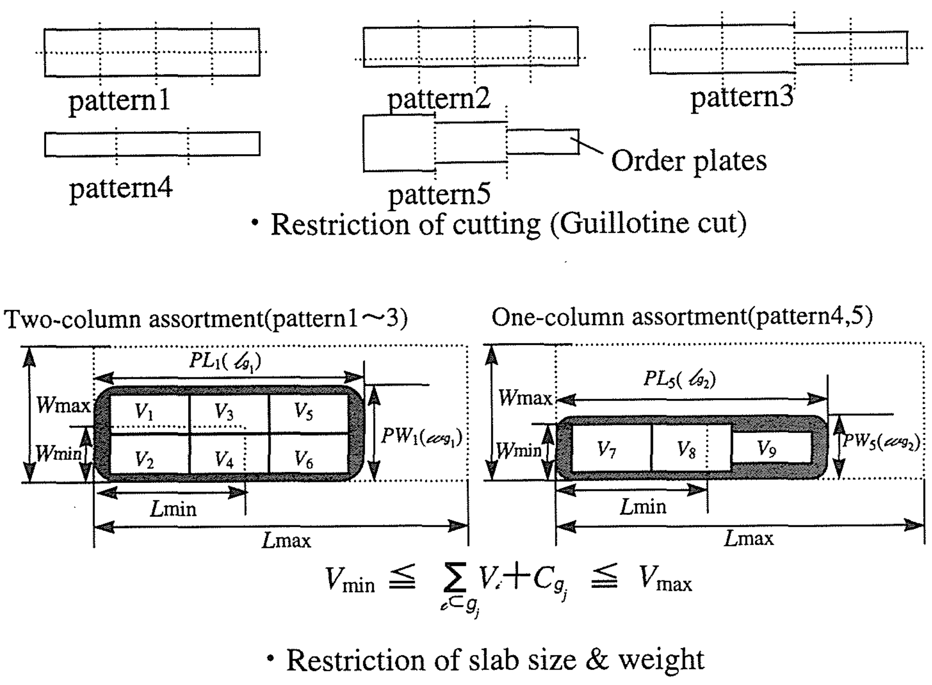
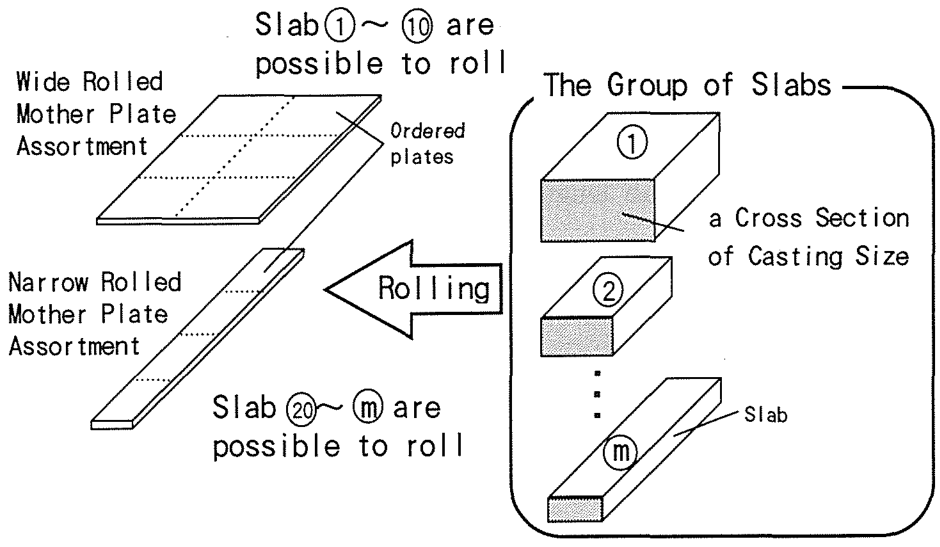

pattern1
pattern2
pattern3
pattern4
pattern5
Order plates
• Restriction of cutting (Guillotine cut)
Two-column assortment(pattern1~3)
One-column assortment(pattern4,5)
PL1(g1)
PL5(g2)
Wmax
Wmin
V1
V2
V3
V4
V5
V6
V7
V8
V9
PW1(g1)
PW5(g2)
Lmin
Lmax
Vmin ≦ Σ Vi+Cgj ≦ Vmax
• Restriction of slab size & weight

Figure 3.3 Restriction of rolling

ferent specification orders in a lot. According to Figure 3.2, even though different specification orders such as A and B are possible to assort, order A and C are impossible to assort. To divide order C into the other group leads to a decrease in the combinatorial possibility. Therefore, we consider order plates in a lot as much as possible.

3.3.2. Restriction of Rolling

Restriction of rolling derives from rolling, cooling and cutting equipment. Figure 3.3 shows restriction of cutting and the rolled mother plate's dimensions. The top of Figure 3.3 shows that cutting restricts only from pattern 1 to pattern 5, what we call guillotine cut (i.e. cuts



Slab ①～ ⑩ are possible to roll
Wide Rolled Mother Plate Assortment
Ordered plates
The Group of Slabs
①
a Cross Section of Casting Size
②
Rolling
Narrow Rolled Mother Plate Assortment
Slab ⑳～ ⓜ are possible to roll
Slab
ⓜ

Figure 3.4 Restriction of slab

from one edge of a previously cut rectangle to the opposite edge). The bottom of Figure 3.3 shows that cooling bed and rolling equipment restrict slab size and weight. We must design order plates smaller than maximum width, length and weight, and larger than minimum width, length, and weight.

3.3.3. Restriction of Slab

Figure 3.4 shows restriction of slab size and weight by a slab cross section of casting size. This restriction is caused from rolling ratio (i.e. divide slab width into rolled mother plate width) on the rolling equipment. According to Figure 3.4, a wide width rolled mother plate can't be rolled from a narrow width slab. A cross section of casting size is not continuous but discontinuous, because it depends on the casting machine. Accordingly, a slab dimension (thickness, width, length) is determined by a rolled mother plate. When we determine the slab dimension, we must check feasible slab length charged in a reheating furnace and feasible cross section rolled on a rolling machine, simultaneously.

3.4. Formulation

In this section, we show the formulations and characteristics of the slab design problem. The problem is formulated as follows:

$$\textbf{Minimize} \quad \sum_j SV_j, \tag{3.1}$$

$$\textbf{subject to.} \tag{3.2}$$

$$SV_j = \sum_{i \in g_j} v_i + C_{g_j}, \tag{3.3}$$

$$SL_j(CS_k) = \alpha \frac{SV_j}{CS_k}, \tag{3.4}$$

$$SV_{min} \leq SV_j \leq SV_{max}, \tag{3.5}$$

$$W_{min} \leq PW_h(w_{g_j}) \leq W_{max}, \tag{3.6}$$

$$L_{min} \leq PL_h(\ell_{g_j}) \leq L_{max}, \tag{3.7}$$

$$SL_{min}(CS_k) \leq SL_j(CS_k) \leq SL_{max}(CS_k). \tag{3.8}$$

where

i	index of orders,
j	index of slabs,
v_i	weight of order i,
SV_j	weight of slab j,
g_j	a set of order index including slab j,
C_{g_j}	margin and waste weight of slab j,
k	index of cross sections,
CS_k	dimensions of cross section k,
α	coefficient,
w_i	width of order i,
ℓ_i	length of order i,
h	assortment pattern,
V_{min}	minimum weight of slab,
V_{max}	maximum weight of slab,
W_{min}	minimum width of slab,
W_{max}	maximum width of slab,
$SL_{min}(CS_k)$	minimum length of slab,
$SL_{max}(CS_k)$	maximum length of slab,
$PW_h(w_{g_j})$	function of rolled mother plate width determined by a set of order plate g_j,
$PL_h(\ell_{g_j})$	function of rolled mother plate length determined by a set of order plate g_j.

The slab design problem is to find a set of order index g_j including slab j which minimizes Eq. (3.1) subject to Eq. (3.3) to (3.8). Further, the slab dimension which has a slab cross section size and length is determined simultaneously. Thus, the slab design problem can be defined as a set partition problem, which packs all order plates $\forall i$ to all slabs $\forall g_j$

From this formulation, it seems when the objective function $\sum SV_j$ equal to minimize margin and waste weight $\sum C_{g_j}$. C_{g_i} is different for each slab, $\sum C_{g_j}$ changes when the assortment changes. Figure 3.5 shows the margin and waste part of a rolled mother plate. Margin and waste weight consists of the cutting part, waste part, top margin and side

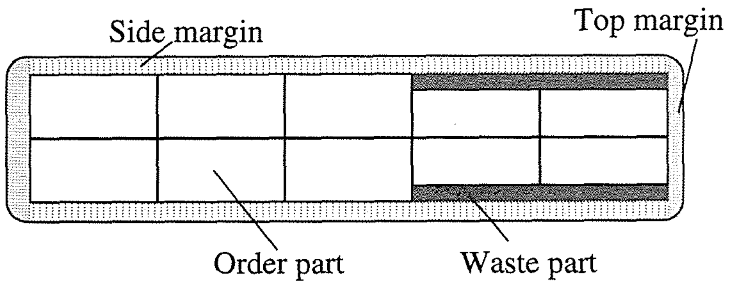


Figure 3.5 Margin and waste weight C_{g_j}

margin weight. Generally speaking, to assort same size order plates into one slab means to decrease margin weight in C_{g_j}, and to increase the number of slabs and total margin weight in $\sum C_{g_j}$. In the opposite way, if different size order plates were assorted into one slab, margin and waste weight in C_{g_j} would increase, and the number of slabs and total margin in $\sum C_{g_j}$ would decrease.

Consequently, trade-off exists between the number of slabs and total slab weight. Therefore, each slab's waste weight C_{g_j} decreases when the number of slabs is minimized, and the number of slabs and total slab weight $\sum SV_j$ increases when each slab's waste weight C_{g_j} is minimized.

3.5. Application Method

3.5.1. Concept of Hybrid Genetic Algorithms

It is too difficult to find the feasible solution within reasonable time under the strict constraints. Thus, we apply hybrid GAs combined with heuristics to the slab design problem. Figure 3.6 shows a concept of a hybrid GAs in a slab design problem. The previous solving method of slab design is given on the right side in Figure 3.6. In this chapter, slab design can be defined for the two-dimensional bin-packing problem on the left side in Figure 3.6. The theoretical bin-packing problem without some restrictions may be given by Next-Fit-Decreasing on bin-packing which B.S.Baker and E.G.Coffman [8] proposed. However, slab design problems in iron and steel industries have many restrictions. We apply Hybrid GAs in order to bridge a gap between theoretical bin-packing and real assortment restrictions. An important point of this research is to develop a method of translation from genotype (genetic information) to phenotype (feasible slab design solution). Two kinds of searching methods of Hybrid GAs in this chapter are as follows:

1. Local Search Conversion from genotype (i.e. the group of order plates) to phenotype (i.e. the group of slabs) by heuristics.

2. Global Search

   Combinatorial global search (minimizing a part of combinatorial

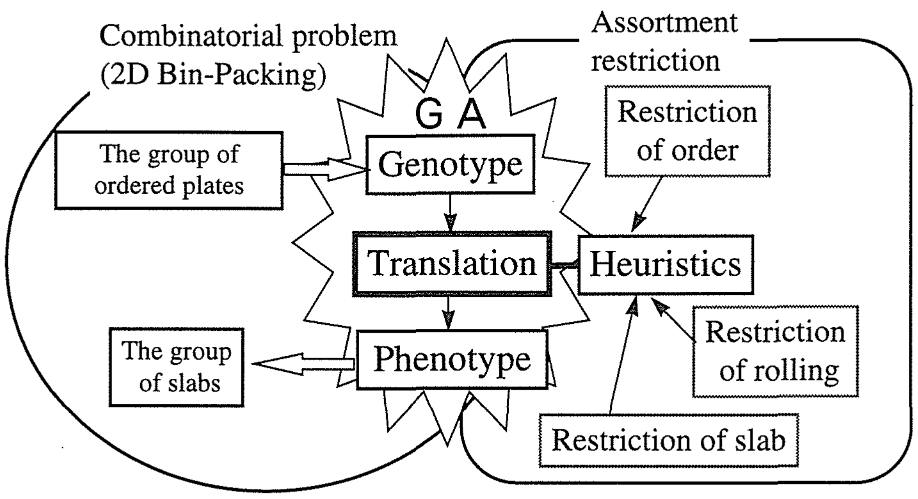


Figure 3.6 A concept of hybrid GAs in slab design problem

solutions enumerated by local search) by GAs.

3.5.2. Procedure of Hybrid GAs on Slab Design

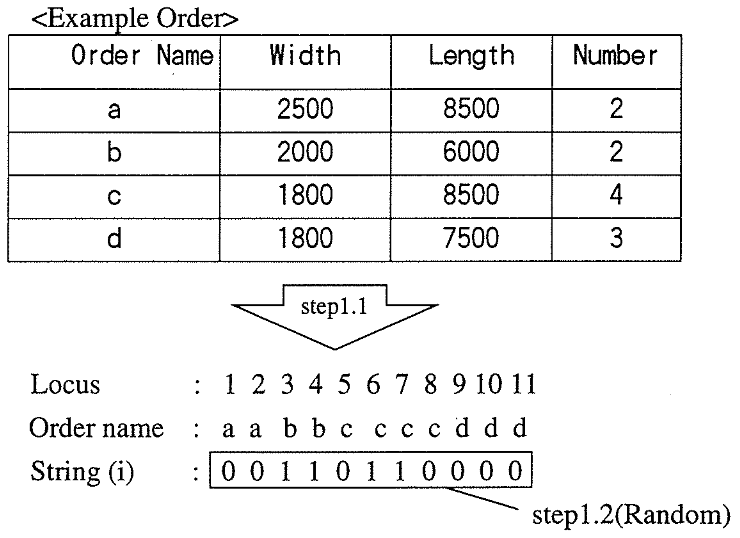
We describe the slab design problem application method as follows.

Step 1 Creating initial population

Mapping locus to order name, and binary code is allocated to locus at random (Figure 3.7).

Step 1.1 Sort order plate's width and length, and spread the number of order plates.

Step 1.2 Set locus 0 or 1 at random as the initial population.



<Example Order>

Order Name	Width	Length	Number
a	2500	8500	2
b	2000	6000	2
c	1800	8500	4
d	1800	7500	3

Locus	:	1	2	3	4	5	6	7	8	9	10	11
Order name	:	a	a	b	b	c	c	c	c	d	d	d
String (i)	:	0	0	1	1	0	1	1	0	0	0	0

Figure 3.7 Example input data

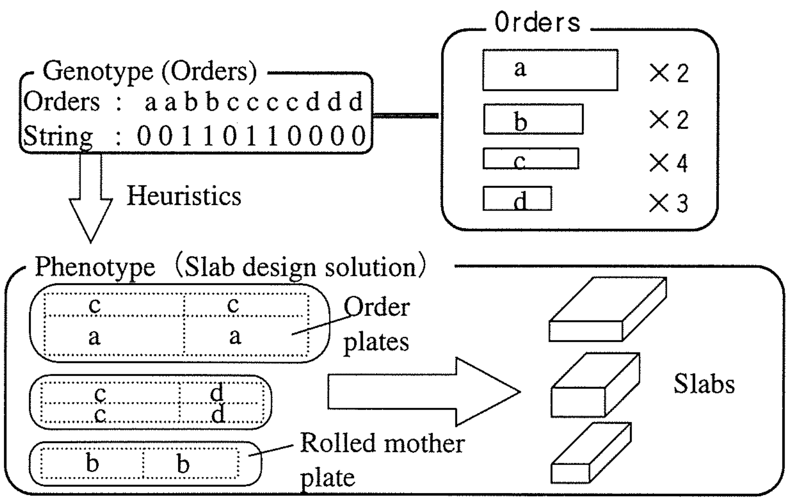


Figure 3.8 Coding method of slab design problem

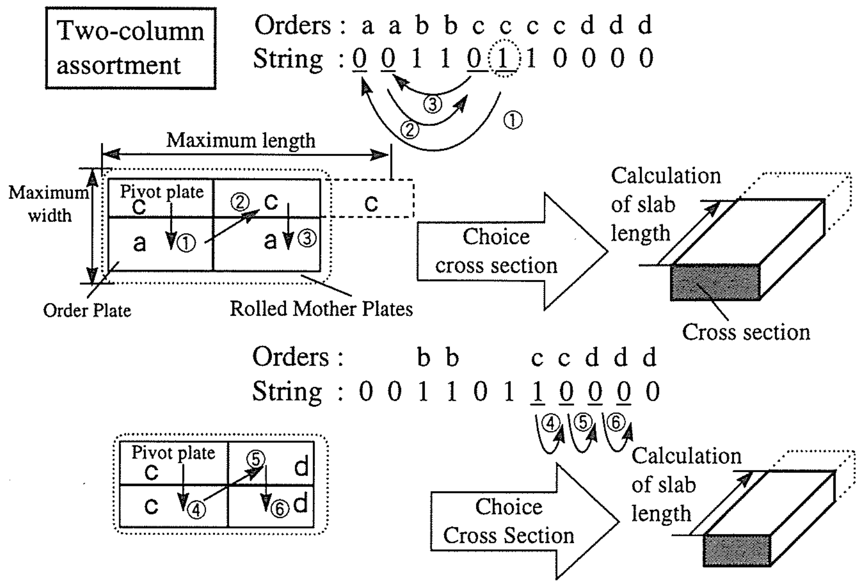


Figure 3.9 Coding method of two-column assortment

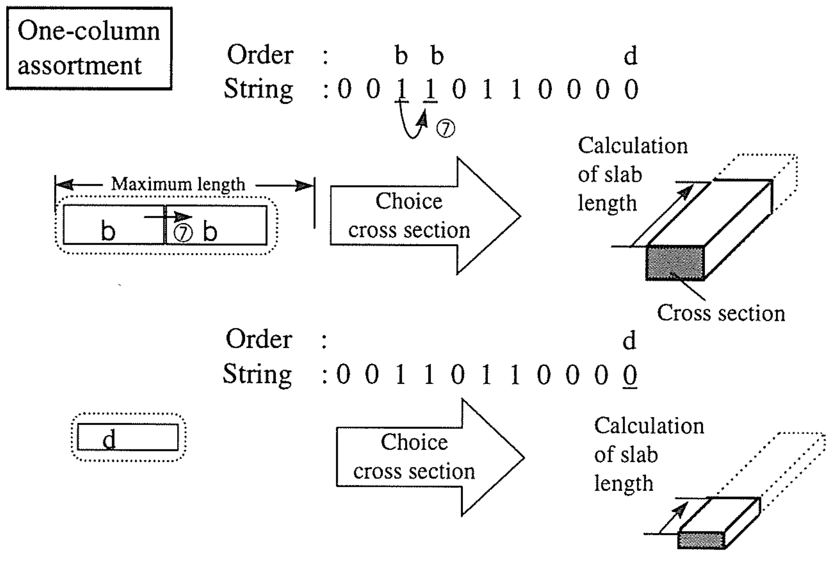


Figure 3.10 Coding method of one-column assortment

Step 2 Coding

Conversion from gene (i.e. the group of order plates) to phenotype (i.e. the group of slabs) by heuristics (Figure 3.8).

Step 2.1 Sort order plate widthwise, lengthwise by turns under the maximum width and length. The only order plate whose locus sets to 1 is to be a pivot order plate in a two-column assortment mother plate, and the other order plate whose locus sets to 0 is sorted following a pivot order plate in two-column assortment (middle of Figure 3.9). Repeat to assort the two-column assortment mother plates until pivot order plate is not in the string (bottom of Figure 3.9).

Step 2.2 Sort left order plate whose locus sets to 1 into one mother plate as one-column assortment. Therefore, sort left order plate whose locus equals to 0 into another mother plate as one-column assortment (Figure 3.10).

Step 3 Genetic Operation

After Step 2 is finished for all strings, choose a pair of strings at random for parents. Operate a pair of strings for crossover, and create a pair of new strings. Choose a string at random, operate a string for mutation, and create a new string. In case a new string is the same as the parent, change the pair of parents, and repeat to create the new strings.

Step 4 Selection

Select higher fitness strings by elitist strategy. Repeat Step 2 and 3 until stopping generation.

3.6. Realistic Application

This section describes the improvement of realistic application for the slab design problem.

3.6.1. Creating Initial Population

Realistic application must find the better solution within reasonable time. Therefore, GA parameter (population size, stopping generation) and quality of initial population are important.

GAs Parameter

First, as we describe the determining method of GA parameter, we define the same characteristic groups as a class. The class is judged from the observational information (order plates' specification, the number of order plates, the kind of order plates' width and length, etc.).

The observational information is classified in Table 3.1 The sensitivity of fitness and run time are given by case study analysis when GA parameters change. Few regulations of increasing state can be observed according to case studies. For example, some class' fitness never increases from initial generation, some class' fitness increases steeply, and some class' fitness increases step-wise.

By those case studies, it is possible to predict problem size and the degree of difficulty from the observational information. Here, the degree of difficulty means a characteristic to judge whether the fitness increases or not. We defined the classes raising the fitness as the 1st (difficult) degree of difficulty when both the population size and the stopping generation increase, the classes raising the fitness as the 13th degree of difficulty when either the population size or the stopping generation increase, and the classes keeping the fitness from initial generation as the 25th (easy) degree of difficulty, even though both the population size and stopping generation change.

Next, we will describe the setting of GA parameter. When GA parameter 1 (population-size 10) takes 50sec (generation 50) until the fitness achieves 95%, GA parameter 2 (population-size 20) takes 40sec (generation 20) until the fitness achieves 95%.

In this case, the latter GA parameter 1 is better. The setting of GA parameter effects run time very much; accordingly the relation between population-size and stopping generation has trade-off. In a realistic system, we would choose a GA parameter minimizing run time from the saddle-points which are given by the same solution.

The slab design system predicts the fitness transition from the observational input data, and dynamically sets the GA parameter which finds the best solution within a reasonable run time.

Table 3.1 Classification of order group

Class	The number of order	The kinds of width	Total order weight	The type of fitness	⋯ ⋯	The size of group	The degree of difficulty
1	large	large	heavy	step-wise	⋯	large	1(difficult)
2	large	middle	heavy	step-wise	⋯	large	1
3	large	small	middle	steep	⋯	middle	2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
32	small	small	light	no-change	⋯	small	25(easy)

Creating Method for Initial Population

We prepared 10 kinds of creating methods for initial population. In this section, we describe a part of them.

CREATING METHOD 1 If more than two same size order plates were in a group and these order plates could assort two-column assortment mother plates, creating method 1 would pack as many same size order plates as possible into a mother plate. In order to create an initial string, count how many order plates can pack into a mother plate. Moreover, count how many mother plates are required to pack all the order plates. Then, set locus to the number of rolled mother plates pivot, 1, and the other locus to 0.

CREATING METHOD 2 Set the same gene (0 or 1) to the same order plate to assort the one-column assortment plate by turns from the widest order plate.

3.6.2. Coding

Coding converts binary code to a slab design solution. Here, the other coding which corresponds a rolled mother plate to an integral number is also possible. However, since the number of slabs j is unknown, it still requires that in order to convert the solution the transaction satisfies the constraints or not to succeed to the next generation. Moreover, if order plates in the same rolled mother plate do not satisfy the constraints such as maximum rolled mother plate's width, length, and weight, the slab design solution not to satisfy the constraint of the strings has risk for decreasing the efficiency of transaction.

Accordingly, we propose the logic that all of the phenotypes satisfy the constraints. Our logic converts a genotype into a phenotype which satisfies the constraints heuristic.

3.6.3. Genetic Operations

Genetic operations mean crossover and mutation. Genetic operation is important to create the new strings as an engine of GAs. Genetic operation is requires two things. The one is to keep variety, and the other is that new strings, like the old strings, should not succeed to the next generation. If this transaction is removed, all of the strings in the generation come to have the same gene. Accordingly, the variety of strings is lost and all of the strings fall into the local optimum solution.

3.6.4. Selection

Few methods of selection are proposed for applying to different types of problems. We apply the elitist strategy which succeeds to the next generation only the higher fitness strings as elites. This method has an advantage of keeping the best solution of the generation. The elitist strategy is suitable in case of finding the best solution within a reasonable run time.

The elite strings of the previous generation need not convert from the binary code to the slab design solution; it is good enough to succeed only the value of fitness. This operation can increase the run time.

3.7. Numerical Example

Figure 3.11 and Figure 3.12 show a part of the case study for determining GA parameter. Figure 3.11 shows the fitness transition when the number of initial pre-optimized strings are 6 and 10. Figure 3.12 shows the fitness transition when the number of initial random strings are 6 and 10.

It resulted that 10 initial strings in a population can create the higher fitness elite from both Figure 3.11 and Figure 3.12. Generally

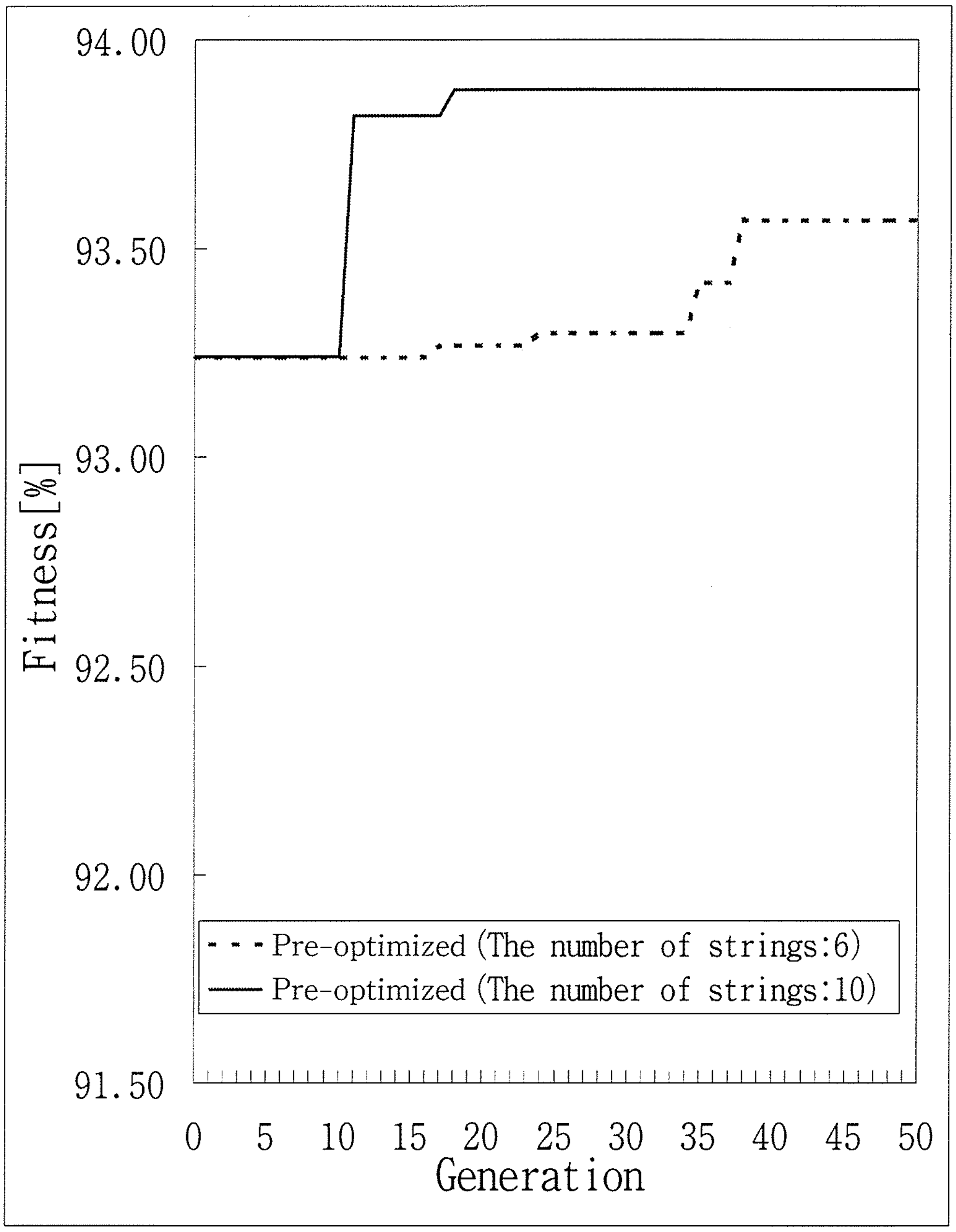


Figure 3.11 The relation between the number of solutions and initial solutions

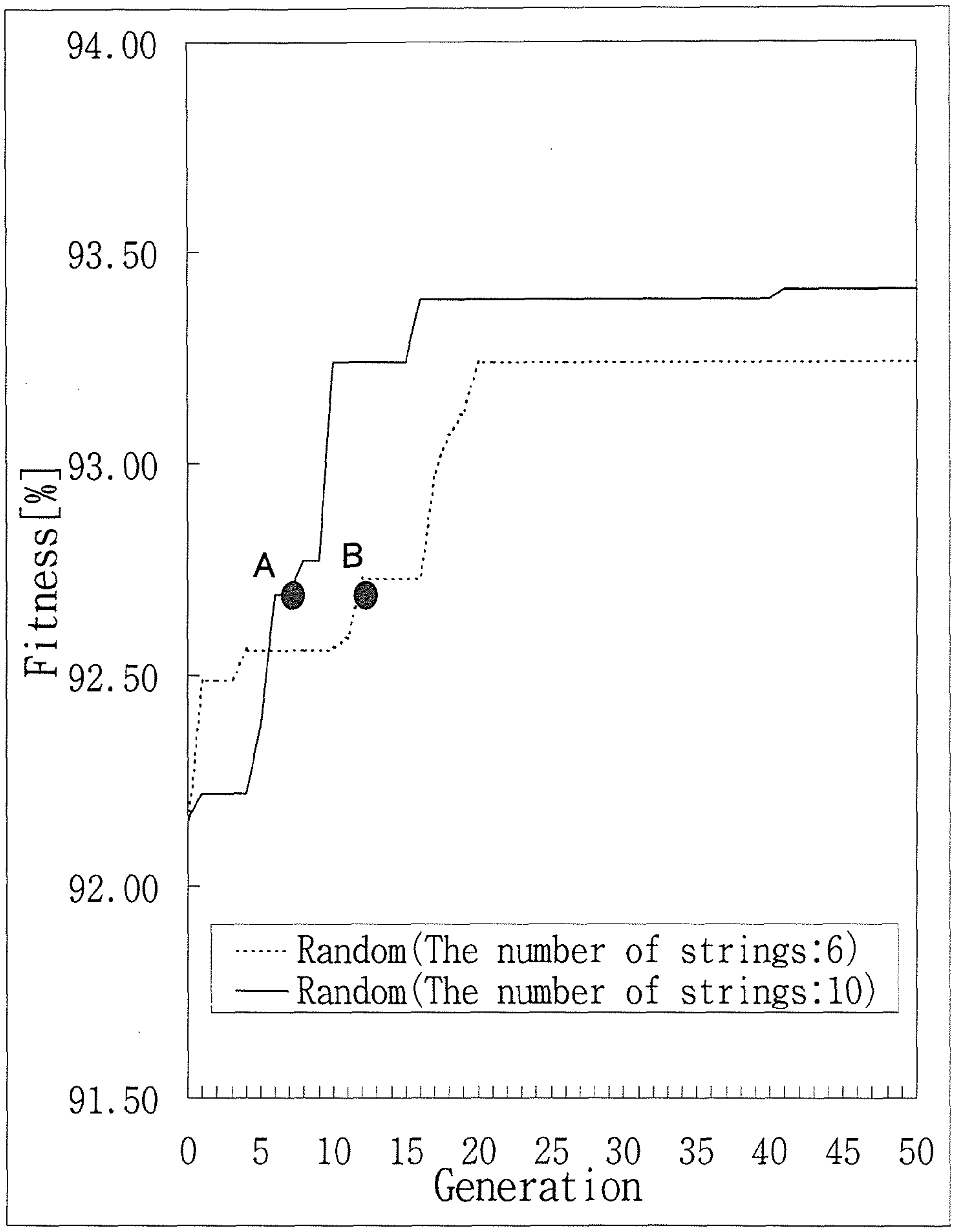


Figure 3.12 The relation between the number of solutions and calculation time

speaking, 10 initial strings in a population are difficult to fall into local optimum solution. This is because 10 initial strings in a population can keep the variety on the meta-search.

However, run time increases when the number of the initial population increases. In spite of point A being earlier than point B in generation, point B is earlier than point A in run time. In the case of realistic application, finding the best solution is required as soon as possible. Accordingly, point B is preferred in Figure 3.12.

The setting of GA parameter is effective for the run time. Therefore, prediction of the fitness transition pattern from observational input data is effective for reduction of run time. Pre-optimized initial population in Figure 3.11 is more effective in the searching process than random initial population in Figure 3.12, when population size is the same. These results show that the improvement of creating initial population method is effective.

Figure 3.13 shows the comparison between former algorithms (branch and bound method) and hybrid GAs, when representative data is input; the kind of order plates are approximately 100, the number of order plates are approximately 300 and the degree of difficulty is high. Hybrid GAs exceed the former algorithms at generation 30. The run time is one second per generation by using a work station (100MIPS). The slab design system must dispose everyday of approximately 5,000 order plates within

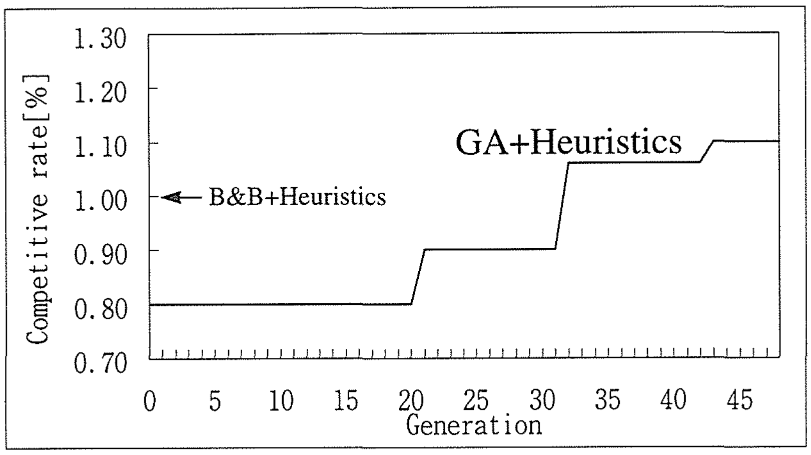


Figure 3.13 The comparison with B&B+heuristics

3 hours. The system applying hybrid genetic algorithms is good enough for the available run time.

The variety of improvement in the previous section can reduce run time 1/8. The setting of GA parameter is flexible for the setting of run time, according to the production environment change.

3.8. Conclusion

In this paper, we proposed hybrid GAs which combined with heuristics on the slab design problem which is a kind of two-dimensional bin packing problem. We have gotten successful results by this variety of improvements. This system has worked in Sumitomo Metal Industries, Kashima Steel works since December in 1995. This system at Kashima Steel works has contributed to a raised yield ratio of 0.6% and other benefits.

Chapter 4.

A MARKOV DECISION PROCESS WITH A CONSTRAINT

4.1. Introduction

Many optimal control problems of a stochastic system, e.g., queuing systems, and reliability systems, can be formulated by Markov Decision Process (MDP). In this chapter, we consider a decision making problem as one of discrete-time MDP problems. This problem has a finite state space, a finite action space and two kinds of immediate rewards. The problem discussed in this chapter is to maximize the time average expected reward generated by one reward stream in the range of pure policy, subject to a constraint that the other average reward is not smaller than a prescribed value. MDP with a single constraint in the range of mixed policy has already been studied by Beulter and Ross [15]. They showed that an optimal policy can be obtained by randomizing at most two pure stationary policies. Mixed policy is, however, very tedious to handle in applying to real world problems. On the other hand, in the case of MDP with no constraint, an optimal pure policy can be obtained by using Policy Iteration Method (PIM). However, for MDP even with a single constraint, any algorithm to find an optimal pure policy has not been discovered.

Here, we apply GAs to find a near optimal, possibly optimal pure policy. GAs proposed in this chapter also fall into a category of a Hybrid GAs in the sense that they combine with PIM. Our new algorithm can find an optimal solution of MDP with a constraint which has 5^{10} feasible solutions in a few minutes by 200mips Work Station .

4.2. MDP with Constraint

4.2.1. Definitions and Notations

Here, we define some notations.

$I = \{0, 1, \cdots, N\}$ the state space,

$D_i = \{1, 2, \cdots, K_i\}$ the action space in state i,

p_{ij}^k one step transition probability when action k is taken in state i,

a_i^k, b_i^k two kinds of immediate rewards when action k is taken in state i,

S set of pure policy, that is, $S = D_0 \times D_1 \times \cdots \times D_N$,

s pure policy, $\mathrm{s} \in S$.

We assume that the Markov chain induced by any pure policy has only one ergodic set. This means that there is a stationary distribution for each policy.

For convenience of expression, let p_{ij}^s, a_i^s and b_i^s be transition probability and immediate rewards when policy s is taken, respectively.

$g(\mathrm{s})$ time average expected reward in an infinite time horizon, with respect to reward a_i^s when policy s is taken,

$h(\mathrm{s})$ time average expected reward in an infinite time horizon, with respect to reward b_i^s when policy s is taken,

π_i^s stationary distribution when policy s is taken.

$g(\mathrm{s})$ and $h(\mathrm{s})$ are derived by the following simultaneous equations which have the unknowns, $g(\mathrm{s}), v_i(\mathrm{s})$ and $h(\mathrm{s}), w_i(\mathrm{s}), i \in I$.

$$\begin{cases} g(\mathrm{s}) + v_i(\mathrm{s}) = a_i^s + \sum_j p_{ij}^s v_j(\mathrm{s}), \quad i = 1, \cdots, n, \\ v_0(\mathrm{s}) = 0, \end{cases} \tag{4.1}$$

$$\begin{cases} h(\mathrm{s}) + w_i(\mathrm{s}) = b_i^s + \sum_j p_{ij}^s w_j(\mathrm{s}), \quad i = 1, \cdots, n, \\ w_0(\mathrm{s}) = 0. \end{cases} \tag{4.2}$$

It is also given by using stationary distribution as follows:

$$g(\mathrm{s}) = \sum_i \pi_i^s a_i^s, \tag{4.3}$$

$$h(\mathrm{s}) = \sum_i \pi_i^s b_i^s, \tag{4.4}$$

$$\pi_j = \sum_i \pi_i^s p_{ij}^s, \quad j \in I, \tag{4.5}$$

$$\sum_i \pi_i = 1. \tag{4.6}$$

Here, our problem is expressed as follows:

$$\max_{s \in S} \quad g(s),$$

subject to

$$h(s) \geq \alpha, \quad s \in S.$$

4.2.2. Mixed And Pure Policy

An optimal mixed policy and an optimal pure policy are shown in Figure 4.1. Thus, a mixed optimal policy is a randomized policy of two pure policies corresponding to clear circles A_1 and A_2. That is, we can find an optimal mixed policy, a clear triangle A, which is the cross point between a vertical line α and a line $\overline{A_1A_2}$. However, an optimal pure policy, a dark star B, is not always on a line $\overline{A_1A_2}$. An optimal pure policy is contained in the triangle area determined by a line $\overline{AA_2}$, a vertical line α and a horizontal line passing through A_2.

After all, searching an optimal pure policy is required to enumerate all pure policies. However, pure policies s exist on the discrete points denoted by dark circles. Moreover, these dark circles have vast sums of the combination of pure policies.

4.3. Genetic Algorithms

Genetic Algorithms are search algorithms based on the mechanics of natural selection and natural genetics [23]. Meta-heuristics [21] such

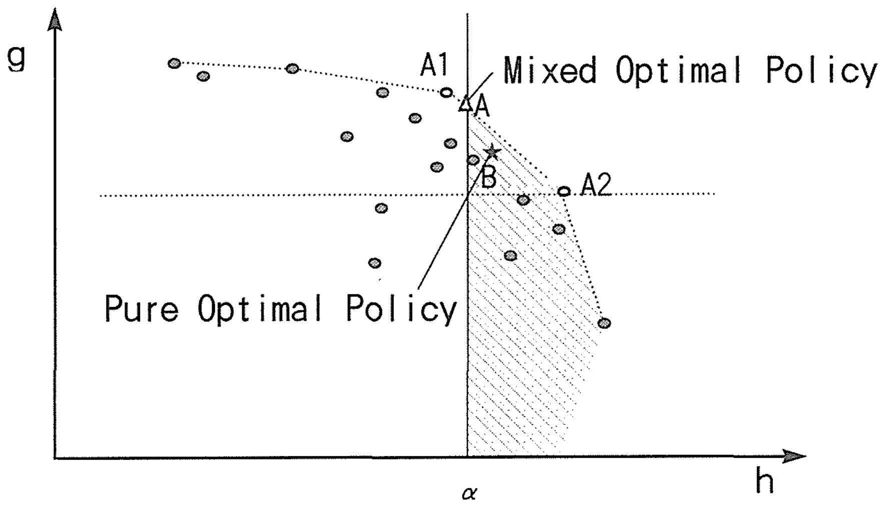


Figure 4.1 Mixed and Pure Policy

as GAs, simulated annealings and neural networks have emerged as the method of choice for applications in engineering optimization problems. Now, many applications are published as optimization algorithms.

4.3.1. Procedure of GAs

In genetics, the gene's function identifies the position of a gene (its locus). In artificial genetic search, strings are composed of a problem's features, which take on different values. Features may be located at different positions on the string [22]. Creatures fall heir to their offspring's superior character by gene, to survive their species. Biological individuals have chromosomes, and each chromosome is constructed with genes. On the computer, a new set of artificial strings is created using bits and pieces of the fittest of the old in every generation. We encode chromosome to policy as follows:

$$\text{strings } \mathrm{s} : k_0, k_1, \cdots, k_N, \quad k_i \in D_i, i \in I. \tag{4.7}$$

String s and gene k_i correspond to policy and action in state i, respectively. Each action is expressed by a decimal number. The GAs work by holding a population of encoded solutions to a search problem, and then try to select and breed new solutions derived from the existing population. The members of the population consist of parents and breed a new population. Typically, parents interbred by exchanging alleles to create a new chromosome using some form of "crossover". In order to maintain good search characteristics a mutation factor is often used. Here, we define $x_m(t)$ to be a m-th string at generation t. String $k_0, k_1, \cdots, k_n$ is a pure policy s in MDP. Gene k_i corresponds to an action in state i. Moreover, we define $X(t)$ to be a population which is a set of different strings at generation t, where M is population size.

$$X(t) = \{x_1(t), x_2(t), \cdots, x_M(t)\}. \tag{4.8}$$

Here, M is population size.

Step 1

By choosing actions for each state at random, we create a policy which is a string on GAs. By the same procedure, we create a set of policies, that is, the initial population $X(0)$.

$$X(0) = \{x_1(0), x_2(0), \cdots, x_m(t), \cdots, x_M(0)\}$$

Step 2

We select better policies which are evaluated by fitness, what we call, objective function. Here, we choose elitist strategy.

Step 3

We create a new population $X(t+1)$ which is a set of new policies by crossover and mutation.

$$X(t+1) = \{x_1(t+1), x_2(t+1), \cdots, x_m(t), \cdots, x_M(t+1)\}.$$

Step 4

We stop or $t = t + 1$ and go to Step 2.

The important parts of this algorithm are how to set the fitness, how to survive the superior policy, and how to create the new policy. Selection plays an important role to induce the direction of successful search. The genetic operator plays an important role to find new search space as a driving force. When these procedures work effectively, GA gives full play to power.

4.3.2. Application of GAs

In this section, we give a way of setting up the fitness, parameter and procedure of GAs. Gene k_i corresponds to an action in state i. String $k_0, k_1, \cdots, k_N$ is a pure policy s in MDP. String length is the number of state $N+1$. s is a target policy. First, calculate π_j^{s} by Eqs. (4.5) and (4.6), then solve $g(\mathrm{s})$, $h(\mathrm{s})$ as phenotype $(h(\mathrm{s}), g(\mathrm{s}))$ by Eqs. (4.3) and (4.4). Important parameters in our GA are reported as follows:

$f(h(\mathrm{s}), g(\mathrm{s}))$	Phenotype of individual (policy),
$N+1$	String length,
M	Population size,
$F(\mathrm{s})$	Fitness of policy s,
L	Allowing generations.

Here, we propose three types of GA; GA1, GA2, and GA3.

GA1

GA1 searches an optimal policy by applying what we call a simple GA. If the time average reward $h(\mathrm{s})$ under policy s is more than α, then average reward $g(\mathrm{s})$ is adopted as the fitness $F(\mathrm{s})$. If $h(\mathrm{s})$ is smaller than or equal to α, the fitness $F(\mathrm{s})$ is zero. Then. this policy does not inherit the next generation. That is, *i)* If $h(\mathrm{s}) \geq \alpha$ then $F(\mathrm{s}) = g(\mathrm{s})$.

ii) If $h(\mathrm{s}) < \alpha$ then $F(\mathrm{s}) = 0$.

GA2

GA2 is a Hybrid GA which combines with PIM in MDP. If the time average reward $h(\mathrm{s})$ is more than or equal to α, then average reward $g(\mathrm{s})$ is adopted as the fitness $F(\mathrm{s})$. If $h(\mathrm{s})$ is smaller than α, then we find a new policy s^* by using PIM, where GA2 has two types of fitness, objective function $F_1(\mathrm{s})$ and a slope of improvement by PIM $F_2(\mathrm{s})$. $F_1(\mathrm{s})$ evaluates objective function $g(\mathrm{s})$, and $F_2(\mathrm{s})$ evaluates a slope of improvement by PIM, that is, $\frac{\Delta g}{\Delta h}$. Accordingly, selection is sorted as the keys of both $F_1(\mathrm{s})$ and $F_2(\mathrm{s})$. If the number of strings which didn't satisfy a constraint has less population, strings which have the higher $F_2(\mathrm{s})$ would survive. That is, the fitness is given as follows:

i) If $h(\mathrm{s}) \geq \alpha$ then, $F_1(\mathrm{s})$ is given by $g(\mathrm{s})$
and $F_2(\mathrm{s})$ is given by 0.

ii) If $h(\mathrm{s}) < \alpha$ then, we solve a simultaneous equation,

$$h(\mathrm{s}) + w_i(\mathrm{s}) = b_i^s + \sum_j p_{ij}^s w_j(\mathrm{s}),$$
$$i \in \{0, \cdots, N\}, \ w_0(\mathrm{s}) = 0.$$

with respect to $h(\mathrm{s}), w_i(\mathrm{s})$.

We apply PIM to the current policy s, that is, we find an action

$$k_i^* = \arg \max_{k \in D_i} \{b_i^k + \sum_j p_{ij}^k w_j(\mathrm{s})\} \text{ for each state } i.$$

We let s^* be a policy $(k_0^*, \cdots, k_N^*)$.

$F_2(\mathrm{s})$ is given by $\dfrac{g(s^*) - g(s)}{h(s^*) - h(s)}$.

iia) If $h(s^*) \geq \alpha$ then, $F_1(\mathrm{s})$ is given by $g(s^*)$.

iib) If $h(s^*) < \alpha$ then, $F_1(\mathrm{s})$ is given by 0.

GA3

GA3 is also a Hybrid GA which combines with PIM. GA3, however, uses penalized fitness. If the time average reward $h(\mathrm{s})$ is more than or equal to α, then the time average reward $g(\mathrm{s})$ is adopted as the fitness $F(\mathrm{s})$. If $h(\mathrm{s})$ is smaller than α, we find a new policy s^* by using the same procedure as GA2. If $h(s^*)$ is more than or equal to α and $g(s^*)$ is more than or equal to $g(s^*)$, we swap current policy s to a new policy s^*. If $h(s^*)$ is more than or equal to α and and $g(s^*)$ is smaller than $g(\mathrm{s})$, we impose a penalty to fitness. If $h(\mathrm{s})$ is smaller than α, we set the fitness zero. By this operation, the ineffective policy isn't selected to the next generation. Thus, the fitness is given by the following procedure:

i) If $h(\mathrm{s}) \geq \alpha$ then, $F(\mathrm{s})$ is given by $g(\mathrm{s})$.

ii) If $h(\mathrm{s}) < \alpha$ then,

a new policy s^* is given by the same procedure as GA2.

iia) If $h(s^*) \geq \alpha$ and $g(s^*) \geq g(\mathrm{s})$ then, we swap

$s, g(\mathrm{s})$ and $h(\mathrm{s})$ to $s^*, g(s^*)$ and $h(s^*)$, respectively.

$F(\mathrm{s})$ is given by $g(s^*)$.

iib) If $h(s^*) \geq \alpha$ and $g(s^*) < g(\mathrm{s})$ then,

$F(\mathrm{s})$ is given by $g(\mathrm{s})/\beta$, $\beta < 1$.

iii) If $h(s^*) < \alpha$ then, $F(\mathrm{s})$ is given by 0.

4.4. Numerical Examples

In this section, we present a numerical example to examine the computational efficiency of the proposed GA1, GA2, and GA3.

We set $N = 9, K_i = 5, i = 0, \cdots, 9, M = 20, L = 300$, and transition probability p_{ij}^k. Immediate rewards a_i^k, b_i^k are given in Tables 4.1 to 4.5. Transition probability p_{ij}^k is, what we call, one step transition probability. This type of transition probability is used for a queuing system.

In this example, the problem size is so small that we can find an optimal pure policy by using the enumeration method. We evaluate three types of GAs by the number of generations which are required to achieve an optimal policy or a near optimal policy, when α is 10, 20, 30, and 40. Moreover, GA1, GA2 and GA3 prepare a string which maximizes $h(s)$ by PIM at initial generation. PIM can maximize time average expected reward, whether $h(s)$ or $g(s)$. This operation is effective to search strings which satisfy a constraint.

Table 4.1 Reward a, b, and probability(1/1000) under action=1

j	a	b	0	1	2	3	4	5	6	7	8	9
i=0	8	8	165	835	0	0	0	0	0	0	0	0
i=1	14	31	276	503	221	0	0	0	0	0	0	0
i=2	13	20	0	404	28	569	0	0	0	0	0	0
i=3	2	16	0	0	179	579	243	0	0	0	0	0
i=4	2	21	0	0	0	256	128	617	0	0	0	0
i=5	10	41	0	0	0	0	762	200	38	0	0	0
i=6	19	37	0	0	0	0	0	168	358	474	0	0
i=7	17	2	0	0	0	0	0	0	336	347	317	0
i=8	3	20	0	0	0	0	0	0	0	354	146	500
i=9	11	38	0	0	0	0	0	0	0	0	504	496

Table 4.2 Reward a, b, and probability(1/1000) under action=2

j	a	b	0	1	2	3	4	5	6	7	8	9
i=0	7	10	162	838	0	0	0	0	0	0	0	0
i=1	4	5	639	14	347	0	0	0	0	0	0	0
i=2	18	22	0	209	442	349	0	0	0	0	0	0
i=3	3	22	0	0	256	279	465	0	0	0	0	0
i=4	15	36	0	0	0	399	45	556	0	0	0	0
i=5	5	32	0	0	0	0	161	0	839	0	0	0
i=6	16	23	0	0	0	0	0	14	183	803	0	0
i=7	2	44	0	0	0	0	0	0	159	483	358	0
i=8	0	7	0	0	0	0	0	0	0	89	278	633
i=9	12	6	0	0	0	0	0	0	0	0	917	83

Table 4.3 Reward a, b, and probability(1/1000) under action=3

j	a	b	0	1	2	3	4	5	6	7	8	9
i=0	9	17	497	503	0	0	0	0	0	0	0	0
i=1	7	4	252	290	457	0	0	0	0	0	0	0
i=2	19	33	0	237	437	326	0	0	0	0	0	0
i=3	19	21	0	0	10	367	622	0	0	0	0	0
i=4	11	48	0	0	0	365	170	465	0	0	0	0
i=5	5	20	0	0	0	0	650	317	33	0	0	0
i=6	19	17	0	0	0	0	0	95	715	190	0	0
i=7	19	41	0	0	0	0	0	0	332	328	341	0
i=8	2	24	0	0	0	0	0	0	0	295	210	495
i=9	4	39	0	0	0	0	0	0	0	0	709	291

Table 4.4 Reward a, b, and probability(1/1000) under action=4

j	a	b	0	1	2	3	4	5	6	7	8	9
i=0	3	39	446	554	0	0	0	0	0	0	0	0
i=1	8	38	436	221	344	0	0	0	0	0	0	0
i=2	19	48	0	620	315	65	0	0	0	0	0	0
i=3	0	5	0	0	196	297	507	0	0	0	0	0
i=4	10	48	0	0	0	127	253	620	0	0	0	0
i=5	5	25	0	0	0	0	240	625	135	0	0	0
i=6	11	46	0	0	0	0	0	347	189	463	0	0
i=7	3	35	0	0	0	0	0	0	301	610	89	0
i=8	4	44	0	0	0	0	0	0	0	109	341	551
i=9	15	35	0	0	0	0	0	0	0	0	673	327

Table 4.5 Reward a, b, and probability(1/1000) under action=5

j	a	b	0	1	2	3	4	5	6	7	8	9
i=0	6	28	549	451	0	0	0	0	0	0	0	0
i=1	6	15	30	626	343	0	0	0	0	0	0	0
i=2	6	40	0	559	235	206	0	0	0	0	0	0
i=3	13	14	0	0	264	595	142	0	0	0	0	0
i=4	16	12	0	0	0	258	379	363	0	0	0	0
i=5	11	26	0	0	0	0	409	181	409	0	0	0
i=6	18	26	0	0	0	0	0	316	377	307	0	0
i=7	11	11	0	0	0	0	0	0	156	269	575	0
i=8	18	15	0	0	0	0	0	0	0	215	355	430
i=9	15	29	0	0	0	0	0	0	0	0	603	397

The harder a constraint is, the more difficult the strings included in the initial population. This results in it taking many generations to search for strings which satisfy a constraint.

Figures 4.2 to 4.4 show the search process of GA1, GA2, and GA3, when α is 40. In these figures, the diamond-shaped point is plotted on the two kinds of average reward $(h(\mathrm{s}), g(\mathrm{s}))$ of the elite, when the generation achieves the side number of the diamond-shape.

Figure 4.2 shows the process of elite $h(\mathrm{s})$ on the X axis and $g(\mathrm{s})$ on the Y axis, when α is 40. GA1 took first movement at generation 66. First movement was going to the left side, in order to increase Y direction as much as possible. If new strings created by genetic operation satisfied a constraint, that is, $h(s) \geq \alpha$, new strings would succeed to the next generation. At first movement, GA1 took more generations than GA2 and GA3. From second movement to last generation, it took approximately 20 generations to the next movement. This would indicate that it is difficult to find strings which satisfy a constraint, even though a string which maximized $h(s)$ by PIM was prepared at initial generation. Second movement is earlier than first movement. It is seen that new strings which are interbred from old strings which satisfy a constraint have an element of satisfying a constraint. Accordingly, to increase the amount of strings which satisfy a constraint among the population means to accelerate the generations until attaining an optimal solution. As a result, when new strings tend to satisfy a constraint, the next movement will be faster. GA1 went zigzag in order to increase $g(s)$ to the Y direction. This seems to indicate that GA1 increased $g(s)$ no matter what the constraint.

Figure 4.3 shows the process of elite $h(\mathrm{s})$ on the X axis and $g(\mathrm{s})$ on

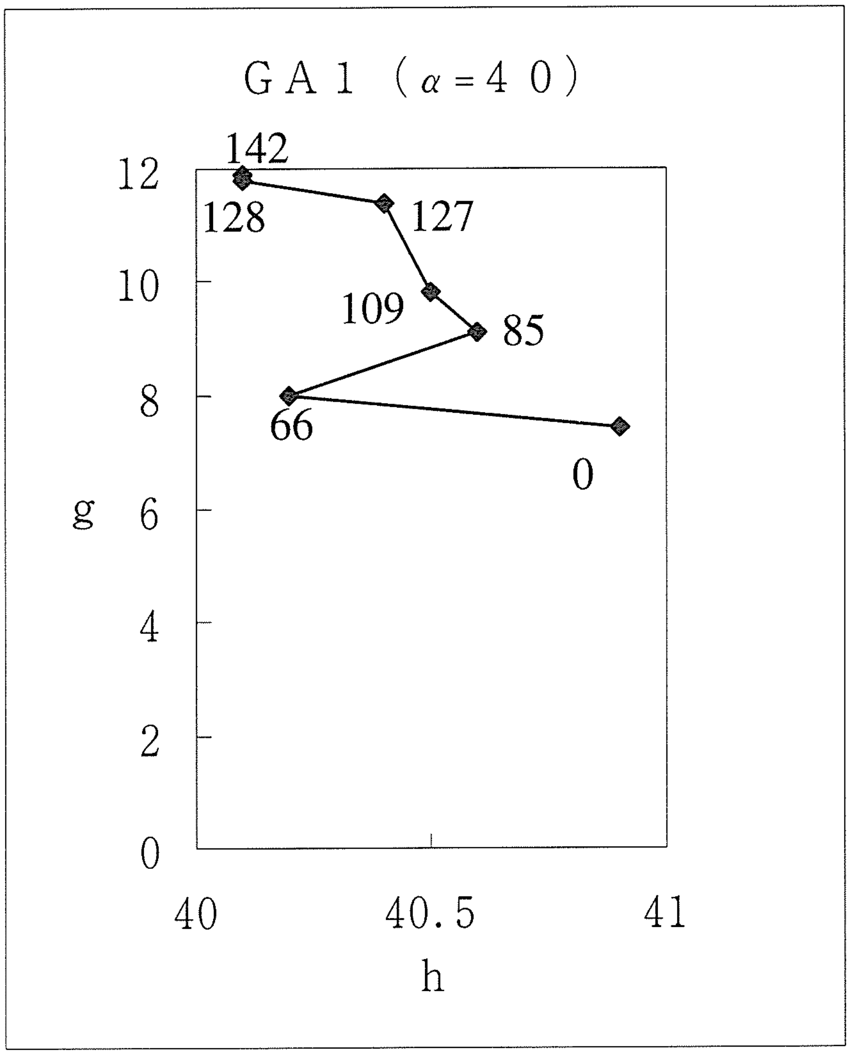


Figure 4.2 (h, g) in GA1 $(h \geq 40)$

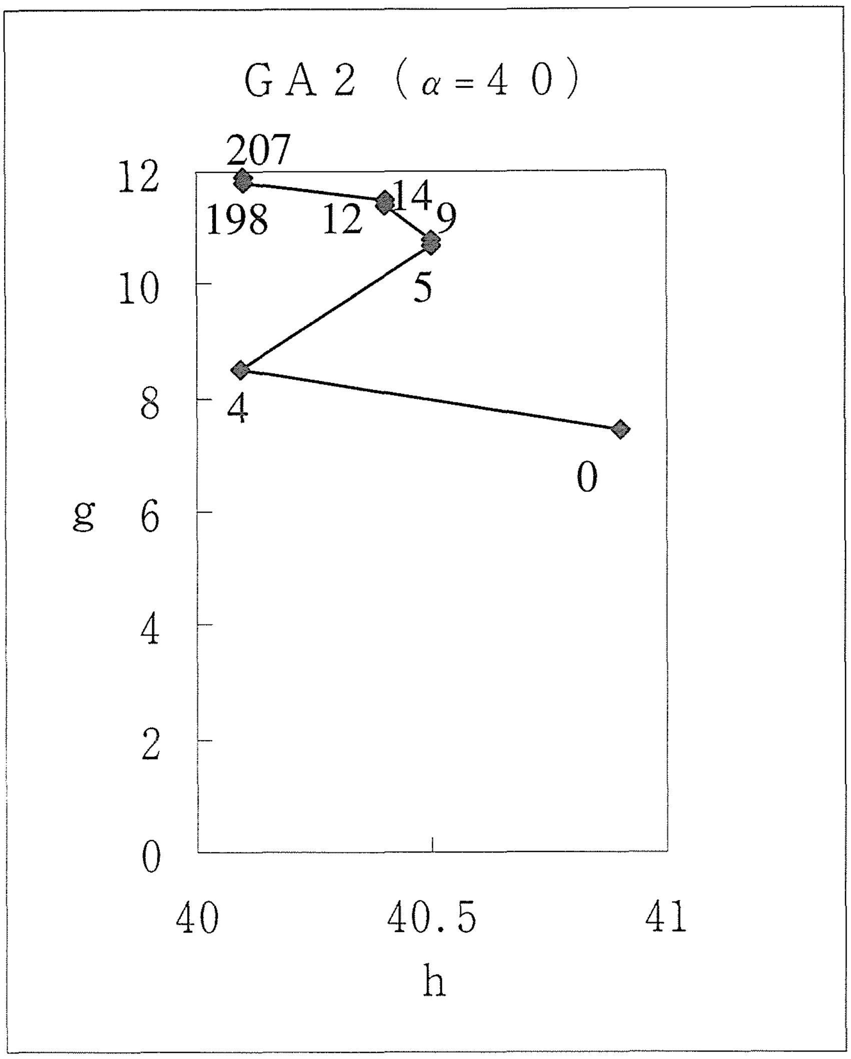


Figure 4.3 (h, g) in GA2 $(h \geq 40)$

the Y axis, when α is 40. GA2 took first movement at only generation 4. First movement was going to the left side, in order to increase Y direction as much as possible. According to Figure 4.4, second movement indicates a characteristic of GA2. GA2 progressed on the Y axis direction more than GA1 and GA3, since fitness of GA2 set up a slope of improvement by PIM, that is, $\frac{\Delta g}{\Delta h}$. PIM dose not always increase $g(s)$, even though it tends to increase $h(s)$. Fitness of GA2 attaches importance to increasing g(s) on the Y axis. If new strings created by genetic operation didn't satisfy a constraint, that is, $h(s) < \alpha$, PIM would try to satisfy a constraint. However, new strings which decrease g(s) on the Y axis have difficulty which do not surviving. Moreover, even new strings created by genetic operation satisfy a constraint survive easier than GA1. GA2 also went zigzag in order to increase $g(s)$ to the Y direction. This seems to indicate that GA2 advances even in the decreasing direction of $h(s)$. GA2 has characteristic which increase $g(s)$ no matter what the constraint.

Figure 4.4 shows the process of elite h(s) on the X axis and g(s) on the Y axis, when α is 40. GA3 achieved an optimal solution in the earliest generation. Moreover, the movement of GA3 is quicker than the others, and it took approximately 5 generations to the next movement. According to Figure 4.4, GA3 took first movement at only generation 3. However, it took last movement in generation 15. Due to attaching importance to satisfying a constraint, it is seen that the variety of strings

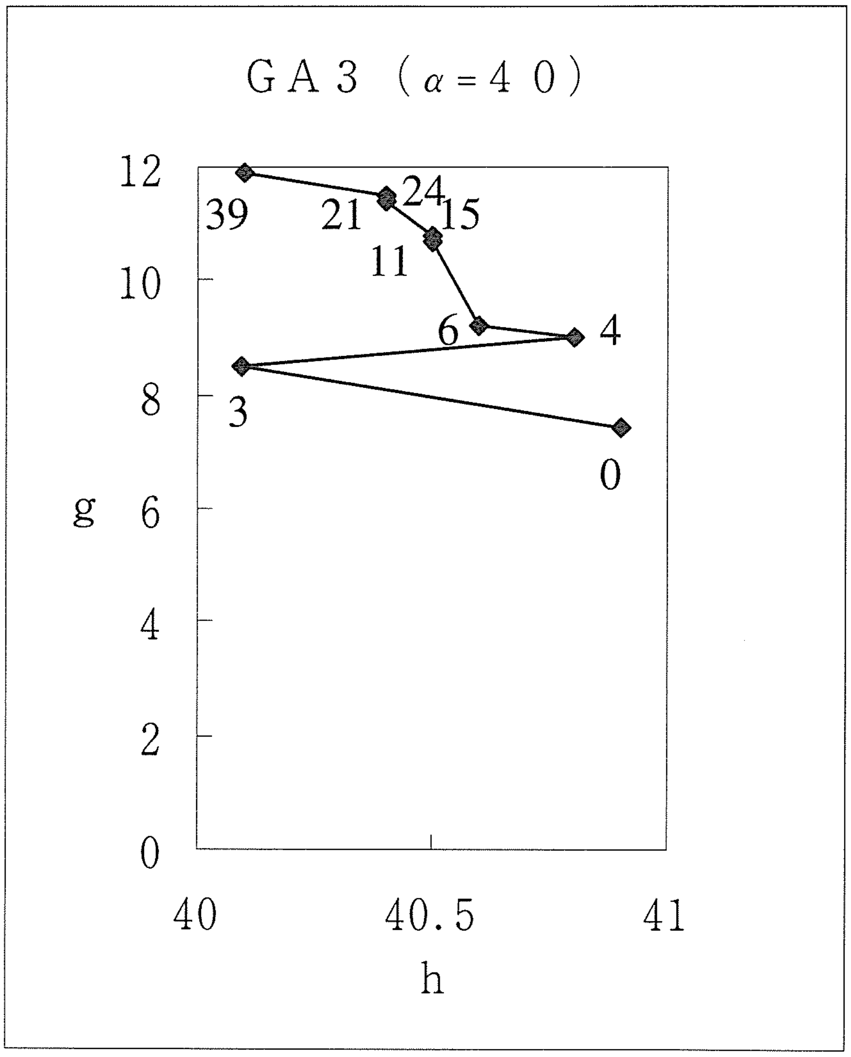


Figure 4.4 (h, g) in GA3 $(h \geq 40)$

in the population was lost. Thus, GA3 is effective when we solve the problems in which it is difficult to find the feasible solution.

Figures 4.5 to 4.8 show the generation process $g(\mathrm{s})$ and $h(\mathrm{s})$ of GA1, GA2, and GA3, when α is 20 and 40.

Figure 4.5 shows the process of elite $g(\mathrm{s})$ at each generation ,when α is 40. When α is 40, there are few feasible solutions which satisfy a constraint. According to Figure 4.5, GA3 attained an optimal solution the fastest. GA2 is better than GA3 until approximately generation 10. However, GA2 is worse than GA1 after approximately generation 120. It is seen that GA2 falls into the local optimal solution. Due to GA2 attaching importance to increasing of $g(\mathrm{s})$, it is difficult to escape from the local optimal solution. This figure indicates GA3 is a robust algorithm, even though there are few feasible solutions.

Figure 4.6 shows the same process, when α is 20. When α is 20, there are many feasible solutions, and this figure shows an opposite case of Figure 4.5. According to Figure 4.6, GA3 still attained an optimal solution the fastest. However, GA1 is better than GA3 until approximately generation 10. GA1 couldn't attain an optimal solution within 300 generations. This result does not include boundlessness generations. Therefore, GA1 might attain an optimal solution if it took more generation. Since GAs have the mutation which can escape the local optimal solution as genetic operation. This indicates GA2 and GA3 which com-

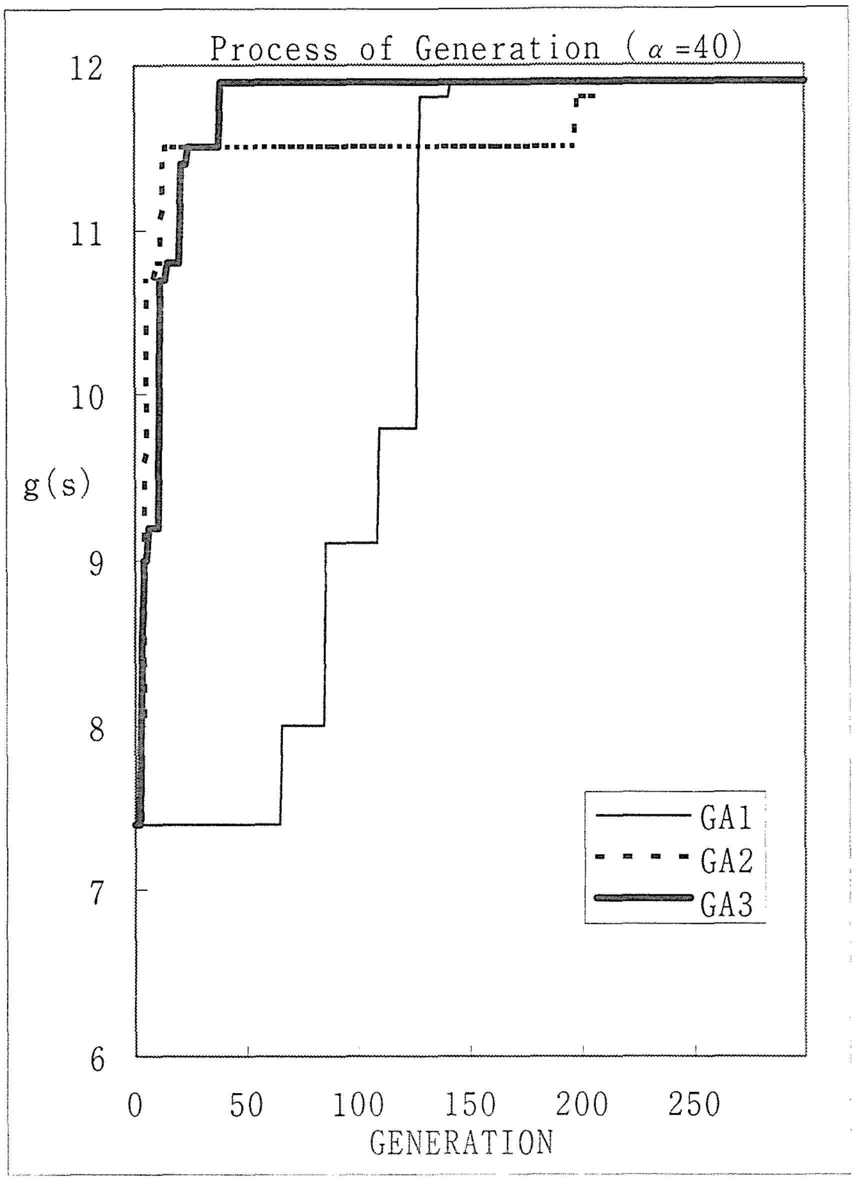


Figure 4.5 Progress of g(s) in GA1,GA2 and GA3 ($h \geq 40$)

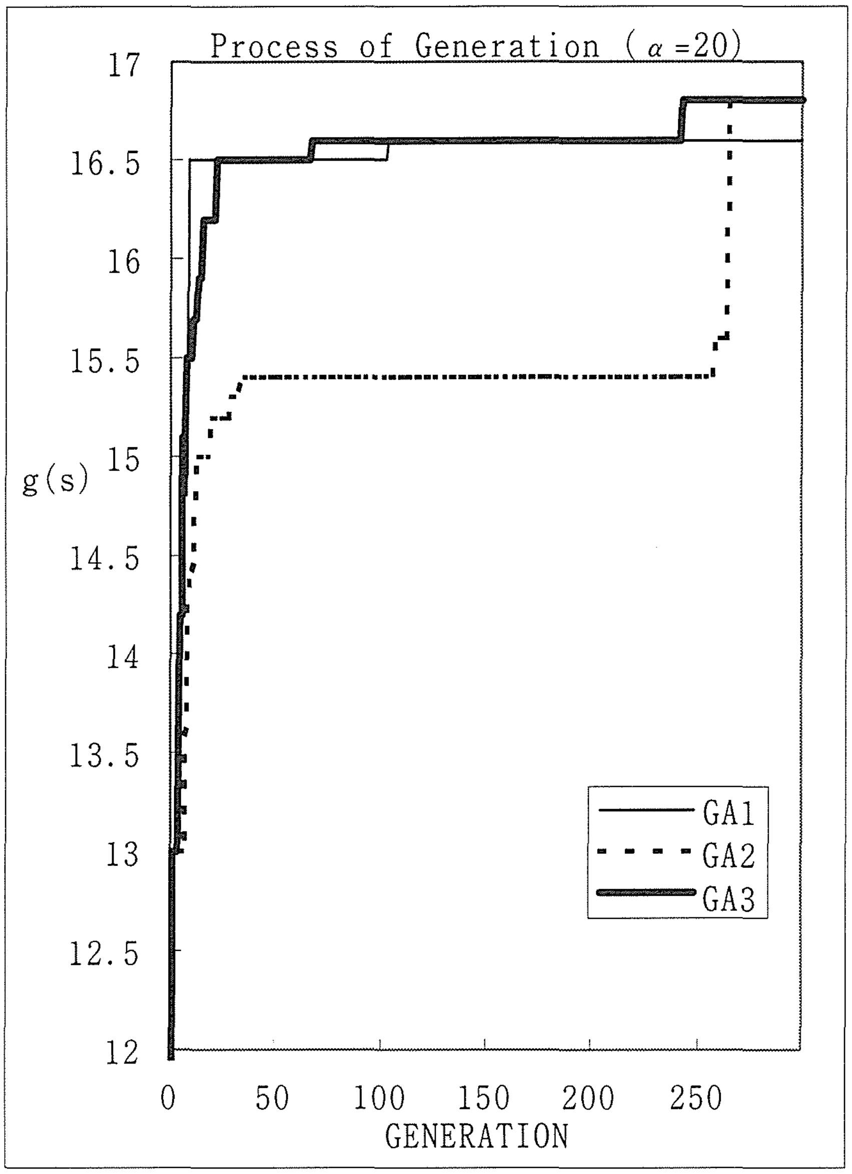


Figure 4.6 Progress of g(s) in GA1,GA2 and GA3 ($h \geq 20$)

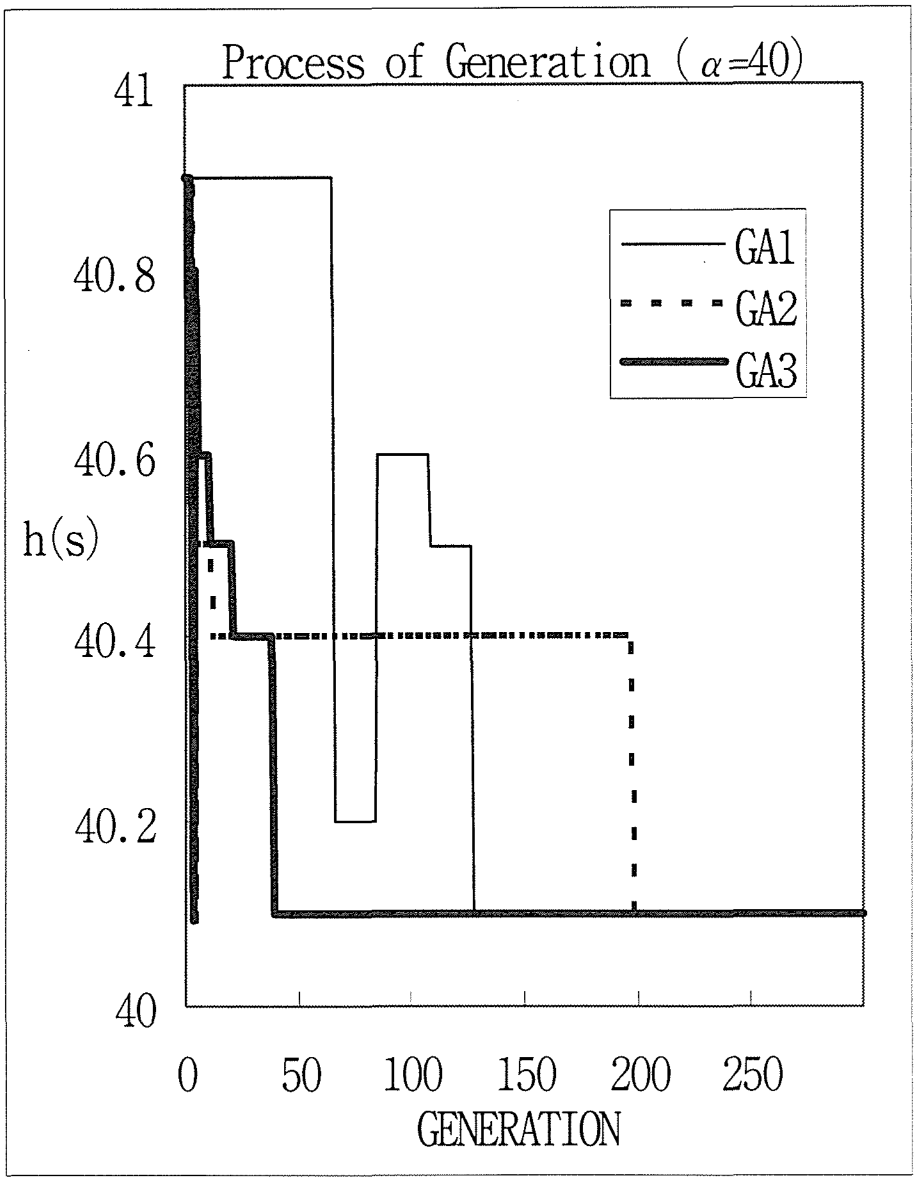


Figure 4.7 Progress of h(s) in GA1,GA2 and GA3 ($h \geq 40$)

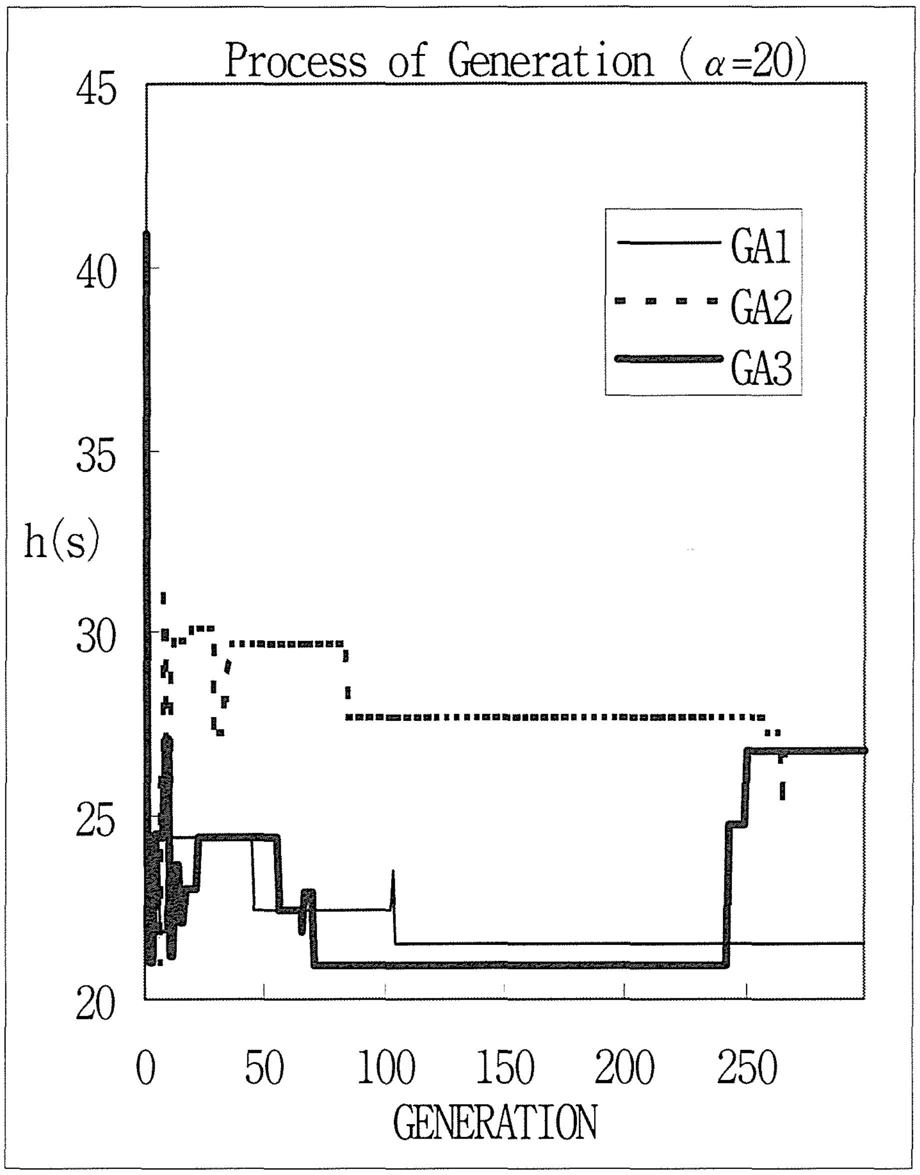


Figure 4.8 Progress of h(s) in GA1,GA2 and GA3 ($h \geq 20$)

bined with PIM are more effective than a simple GAs, i.e. GA1.

Figure 4.7 shows $h(\mathrm{s})$ in the same process as Figure 4.5. According to Figure 4.7, GA3 approached 40 the fastest. GA2 and GA3 repeated up and down until generation 10, and decreased step-wise. On the other hand, GA1 repeated up and down from generations 60 to 70. This indicates GA2 and GA3 searched a wide area by PIM until generation 10, and after that progressed in feasible solution space.

Figure 4.8 shows $h(\mathrm{s})$ in the same process as Figure 4.6. According to Figure 4.8, GA3 still approached 40 the fastest. As compared with Figure 4.7, GA1, GA2 and GA3 repeated up and down until generation 300. Due to there being many feasible solutions, they searched various points in wide feasible solution space in order to increase $g(\mathrm{s})$. Moreover, it is interesting that the GA2 searching area is different from GA3's. GA2 searched from $h(\mathrm{s}) = 27$ to $h(\mathrm{s}) = 30$, GA3 searched from $h(\mathrm{s}) = 21$ to $h(\mathrm{s}) = 27$. This indicates each GA approaches a problem in a different way.

The optimal solution and the average reward $g(\mathrm{s})$ (the generations) by each GAs in each problem is listed in Table 4.6. Table 4.6 shows enough results to understand that GAs are an effective search method. In this example, maximum $h(\mathrm{s})$ is 40.9 [$g(\mathrm{s}) = 7.4$] and maximum $g(\mathrm{s})$ is 16.8 [$h(\mathrm{s}) = 24.8$]. We confirm that Hybrid GAs (GA2,GA3) are more effective than a simple GA (GA1) because GA1 can not achieve an optimal pure policy until generation 200, when α is 20. However, GA2 and GA3 can find an optimal pure policy for whatever value α is. Also, GA3 is more effective than GA2, when α is a large value.

Table 4.6 The results of numerical example

α	10	20	30	40
Optimal Solution	16.8	16.8	16.6	11.9
GA1	16.8(202)	16.6(104)	16.6(185)	11.9(142)
GA2	16.8(202)	16.8(266)	16.6(117)	11.9(207)
GA3	16.8(202)	16.8(243)	16.6(100)	11.9(39)

4.5. Conclusion

In this paper, we proposed Hybrid GA which combined with PIM on MDP with constraints. We got successful results. When PIM is used once per improvement of individuals, it is possible to find the policy which satisfies the constraint by retrying. It is interesting that search processes are different by the setting of fitness, such as GA2 and GA3. We examined MDP with single constraint in this paper, and we are sure that GA can be used as a tool for MDP with multiple constraints. However, we think it is important to consider the way of setting the fitness in order to search effectively. It would be of benefit to continue the research of these problems.

Chapter 5.

A SINGLE RESERVOIR OPERATION OPTIMIZATION PROBLEM

5.1. Introduction

There are many decision making problems in the world. Scientific decision making is important for economic and government activity. Most decision making problems can be dealt with a MDP model. Hence, decision making problems in the real world have many constraints. Therefore, to solve the decision making problems in the real world, we must solve the MDP with multi constraints problems. As we described in the previous chapter, it is difficult to solve MDP with even one constraint. Moreover, there is not an application to get an optimal solution within a reasonable time.

A reservoir operation problem is also a decision making problem. However, if a reservoir operation optimization model is formulated as a MDP model, this model must be dealt with as a MDP with multi constraints. This is because it is required to estimate drought 'frequency', 'duration' and 'magnitude', in order to analyze reliability performances of a reservoir against droughts.

Since Moran's pioneering work [26] on 'Stochastic Reservoir Theory' in 1954, extensive research has been done to analyze reliability performance of reservoir systems [27], [28]. In particular, remarkable progress has been made in the estimation of relevant indices. These indices specify the probability that water is available from the reservoir (Reliability index) and the expected duration of that following a drought (Expected Duration index).

Hashimoto et al. [29] in 1982 proposed to use 'reliability', 'resiliency' and 'vulnerability' criteria for water resources system performance evaluation. Hashimoto's indices cover the concepts of drought 'frequency', 'duration' and 'magnitude' but fail to deal explicitly with the inflow distribution. In order to introduce these indices into a stochastic programming model, a consistent theoretical basis for formulating indices is required.

Tatano et al. discussed a reservoir operation rule in their paper [36]. The model was formulated in the form of a stochastic linear programming model which minimizes expected loss per period subject to two kinds of reliability constraints of drought frequency and expected drought duration. In that model, state variables were defined at maximum available amounts for release and occurrence of drought.

In this chapter, we try to solve a reservoir operation optimization problem, which is formulated to a MDP with multiple reservoir reliability constraints. The model determines an operation rule which minimizes expected welfare loss per period subject to some reliability constraints of 'expected drought duration' and 'drought frequencies'. We present a new approach to solve a reservoir operation optimization model applying hybrid GAs. In applying GAs, we represent each water release strategy by a string in GAs. The expected welfare loss per period of each strategy is described as the 'fitness'. At each stage to generate a string (i.e. water release strategy) in GAs, strings which are unsatisfied by either constraint are improved by using PIM. Therefore, we take into consideration the two constraints of expected drought duration and drought frequency. To create new strings, we apply genetic operators such as crossover and mutation. Some numerical examples are also presented to investigate the computational efficiency of our hybrid GA to solve the optimal water release problem with two constraints in reliability of a single reservoir. Our new algorithm can find an optimal solution of MDP with a constraint which has 3.24×10^{10} feasible solutions in a few minutes by 200mips Work Station .

5.2. Preliminaries

5.2.1. State Variables

Defining S_n, I_n and R_n as reservoir storage, inflow and release variables at the nth stage, respectively, where the stage parameter n implies the time duration from first stage to the nth stage, S_n, I_n and R_n take non-negative discrete values, i.e.,

$$I_n = 0, 1, 2, \cdots, \overline{I}, \tag{5.1}$$

$$S_n = 0, 1, 2, \cdots, v, \tag{5.2}$$

$$R_n = 0, 1, 2, \cdots, v + \overline{I}, \tag{5.3}$$

where the values of these variables are discrete based on a certain unit amount of water, v is the reservoir capacity and $\overline{I}$ is the upper bound of inflows.

In the present model, the state vector at stage n is defined as (X_n, δ_n). X_n is the maximum available amount of water for release at stage n and is defined as $S_{n+1} = S_n + I_n - R_n$. Therefore, the domain of X_n at each stage n is $\mathbf{X} = \{0, \Delta, 2\Delta, \cdots, v + \overline{I}\}$. δ_n is a dummy variable which specifies the occurrence of a drought. The definition of δ_n is

$$\delta_n = \begin{cases} 1 & \text{if} \quad R_{n-1} < D, \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

where D denotes demand level at normal stages n and is assumed to be a constant. Let $\Delta = \{0, 1\}$ be the domain of δ_n.

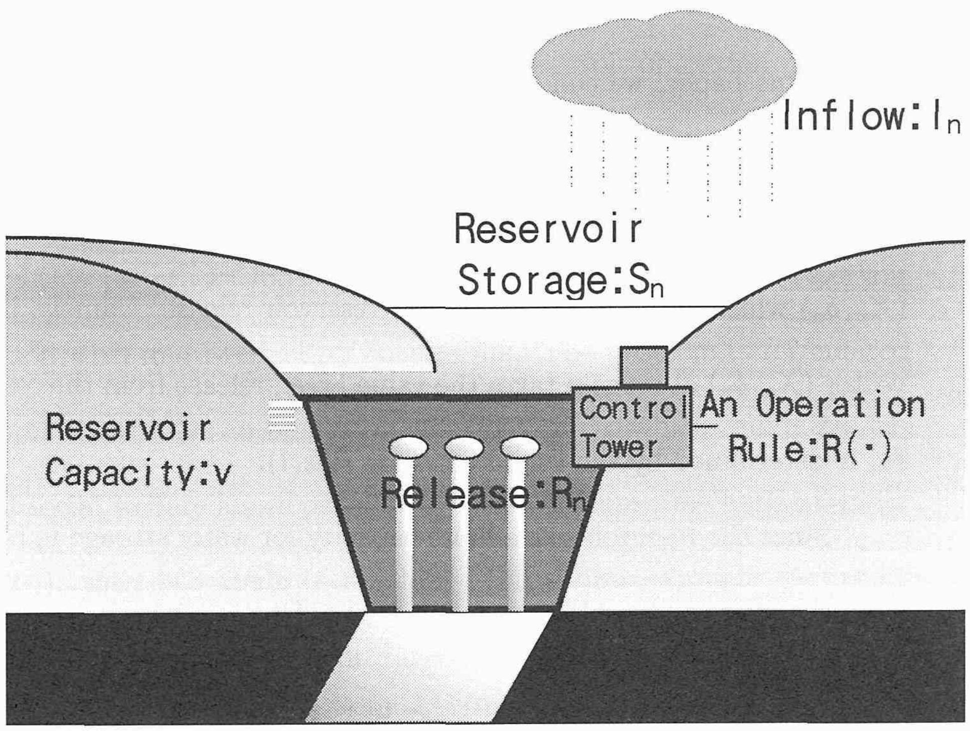


Figure 5.1 Process of $L(R(\cdot))$ and $FR(R(\cdot))$ on Problem 1

If a drought is in process at stage n, water demand D should exceed water supply R_{n-1} at stage $n-1$ and δ_n takes unity. If δ_n takes zero, reservoir release R_{n-1} exceeds water demand D at normal stages, i.e., a drought is not in process at stage n.

Figure 5.1 shows the model of a single reservoir operation optimization problem. According to Figure 5.1, an optimal operation rule is determined by the control tower while taking into account welfare loss per period and reliability constraints of 'expected drought duration' and 'drought frequencies'.

5.2.2. Single Reservoir Operation Rule

In this paper, we concentrate on finding a desirable operation rule from a class of steady, closed-loop operation rules. Reservoir operation rule $R(x,l)$ is defined as a function of values (x,l) of the state vector (X_n, δ_n) which specifies the amount of reservoir release. Namely, if state vector (X_n, δ_n) at stage n takes the value (x,l), release from the reservoir R_n is determined by the rule, i.e., $R_n = R(x,l)$.

Since the reservoir has a finite capacity for water storage v, releases from the reservoir should stay in the region $\Omega(x;v)$, where

$$\Omega(x;v) = \{R_n | \max(0, x-v) \leq R_n \leq x\}. \tag{5.5}$$

This means that the region $\Omega(x;v)$ consists of the one of the constraints in mathematical formulation to be solved in order to design desirable operation rules. In other words, mathematical formulation to be solved in this paper are formatted to find a satisfiable function $R^* : \mathbf{X} \times \Delta \to \mathbf{X}$ in terms of the objective, social expected loss minimization, which satisfies reliability constraints from a class of admissible functions $\mathcal{R}$, where

$$\mathcal{R} = \{R : \mathbf{X} \times \Delta \to \mathbf{X} | R(x,l) \in \Omega(x:v) \text{ for } \forall (x,l) \in \mathbf{X} \times \Delta\}. \tag{5.6}$$

For convenience of expression, an operation rule which is formulated as a function $R(x,l)$ is denoted by $R(\cdot)$. Since reservoir operation rule $R(\cdot)$ deterministically assigns a specific amount of releases to all the possible values of state vector $(x, l) \in \mathbf{X} \times \Delta$, the operation rule $R(\cdot)$ can be regarded as a pure strategy for a single reservoir operation policy.

5.3. Problem Definition

In this paper, we focus on a design problem of a single reservoir operation with multi-reliability constraints. The problem is formulated as a problem to find an operation rule $\tilde{R}(\cdot)$ in $\mathcal{R}$ which minimizes expected social welfare losses per period within infinite time horizon under $EL(R(\cdot))$, with two kinds of reliability constraints, drought frequency $FR(R(\cdot))$ and expected drought duration $ED(R(\cdot))$:

$$\min_{R(\cdot)\in\mathcal{R}} \; EL(R(\cdot)), \tag{5.7}$$

subject to

$$FR(R(\cdot)) \leq \overline{FR}, \tag{5.8}$$

$$ED(R(\cdot)) \leq \overline{ED}. \tag{5.9}$$

where $\overline{FR}$ and $\overline{ED}$ denote upper bounds of drought frequency and expected drought duration, respectively.

5.3.1. Transition Probabilities

Given an operation rule $R(x, l)$, one step transition probability $P^{R(x,l)}_{(x,l)\,(y,k)}$ from state (x, l) to (y, k) is formulated as

$$P^{R(x,l)}_{(x,l)\,(y,k)} = \Pr\{X_{n+1} = y, \delta_{n+1} = k | X_n = x, \delta_n = l\}. \tag{5.10}$$

Storage balance condition is defined as $S_{n+1} = S_n + I_n - R_n$. From the storage balance condition, the following relationship among maximum available amounts for release X_n, X_{n+1} , release R_n and inflow I_{n+1} holds,

$$I_{n+1} = X_{n+1} - X_n + R_n, (n = 1, 2, \cdots). \tag{5.11}$$

It is assumed that inflow I_n is an independent discrete random variable*, identical at each stage and having a PDF $\theta(\cdot)$.

Then, each value of one step state transition probability $P^{R(x,l)}_{(x,l)\ (y,k)}$ is calculated by

$$P^{R(x,l)}_{(x,l)\ (y,k)} = \theta(y-x+r) \cdot \{\chi(k=0)\chi(r \geq D) + \chi(k=1)\chi(r < D)\}. \tag{5.12}$$

where $\chi(\cdot)$ is a function which takes unity if $(\cdot)$ is true, and null otherwise.

It is easily found that state vector (X_N, δ_n) forms an ergodic Markov Chain given a specific operation rule $R(\cdot)$. Hence, the reservoir operation problem formulated in the above can be regarded as a special case of MDP problems with reliability constraints.

5.3.2. Reliability Indices for Designing Reservoir the Operation Rules

Based on the knowledge that state vector (X_N, δ_n) forms an ergodic Markov Chain, let us show the way to estimate reliability indices which

*Of course, it is easy to attend the model to the serially correlate inflow cases. In order to show the basic idea of applying hybrid GAs, the authors decided to show the model for the simplest case.

correspond to a specific operation rule $R(\cdot)$, $EL(R(\cdot))$, $FR(R(\cdot))$ and $ED(R(\cdot))$.

Welfare losses due to droughts are assumed to occur when release from the reservoir R_n is smaller than water demand D and to depend on deficits $D - R_n$. Let the social welfare losses in a single stage be denoted by function $L(r)$ of release from the reservoir at the stage where $L(r)$ takes a positive real number [†] if release is smaller than demand, i.e., $r < D$, and $L(r)$ takes zero otherwise.

Since state vector (X_N, δ_n) forms an ergodic Markov Chain, given a certain operation rule $R(\cdot)$, the expected value of social welfare losses per period, $\sum_{t=0}^{n} E[L(R(X_t, \Delta_t))]/(n+1)$ converges. Hence, we can denote the expected value of social welfare losses per period $EL(R(\cdot))$ as

$$EL(R(\cdot)) = \lim_{n\to\infty} \sum_{t=0}^{n} \frac{E[L(R(X_n, \Delta_n))]}{(n+1)}. \tag{5.13}$$

Then, $EL(R(\cdot))$ can be estimated by solving the following equations:

$$EL(R(\cdot)) + u_{(x,l)} = L(R(x,l)) + \sum_{(y,k)} P^{R(x,l)}_{(x,l)\ (y,k)} u_{(y,k)}, \tag{5.14}$$
$$\text{for } \forall (x,l) \in \mathbf{X} \times \Delta.$$

where $EL(R(\cdot))$ and $u_{(x,l)}$ are unknown variables for the equations.

In order to formulate $FR(\cdot)$ and $ED(\cdot)$, functions $L_{PF}(r,l)$ $L_{FR}(r,l)$ are formulated as

$$L_{PF}(r,l) = \begin{cases} 1 \text{ if } \ r < D, \\ 0 \text{ otherwise}, \end{cases} \tag{5.15}$$

[†] $L(r)$ can be measured by a monetary term.

$$L_{FR}(r,l) = \begin{cases} 1 \text{ if } \ r < D \ \text{ and } \ l = 0, \\ 0 \text{ otherwise.} \end{cases} \tag{5.16}$$

In the same discussion for $EL(R(\cdot))$, given an operation rule $R(\cdot)$, average expected values of these functions converge. Let $PF(R(\cdot))$ and $FR(R(\cdot))$ be the converged average expected values of these functions. Then, $PF(R(\cdot))$ and $FR(R(\cdot))$ can solve the following equations:

$$PF(R(\cdot)) + v_{(x,l)} = L_{PF}(R(x,l),l) + \sum_{(y,k)} P^{R(x,l)}_{(x,l)\ (y,k)} v_{(y,k)}, \tag{5.17}$$
$$\text{for } \forall (x,l) \in \mathbf{X} \times \Delta,$$

$$FR(R(\cdot)) + w_{(x,l)} = L_{FR}(R(x,l),l) + \sum_{(y,k)} P^{R(x,l)}_{(x,l)\ (y,k)} w_{(y,k)}, \tag{5.18}$$
$$\text{for } \forall (x,l) \in \mathbf{X} \times \Delta,$$

where $PF(R(\cdot))$, $v_{(x,l)}$, $FR(R(\cdot))$ and $w_{(x,l)}$ are unknown variables for the above equations. Given a certain operation rule $R(\cdot)$, drought frequency is estimated by solving Eq.(5.19). It has been proved by Tatano, et.al. [36] that $ED(R(\cdot))$ is estimated by

$$ED(R(\cdot)) = \frac{PF(R(\cdot))}{FR(R(\cdot))}. \tag{5.19}$$

Now, it is clearer to estimate values of functions which consist in our programming problem.

5.4. Genetic Algorithms

The optimization technique using GAs also falls into the category of Meta heuristics [21]. Meta heuristics have been remarkably applied for many problems, for which it has been difficult to solve the optimal solution up to now.

5.4.1. Procedure of GAs

GAs is a search algorithm based on the mechanics of natural selection and natural genetics [23]. In natural terminology, we say that chromosomes are composed of genes, which may take on some number of values called alleles. In genetics, the position of a gene (its locus) is identified separately from the gene's function. In artificial genetic search, we say that strings are composed of features or detectors, which take on different values. Features may be located at different positions on the string. On the computer, a new set of artificial strings is created by using bits and pieces of the fittest of the old in every generation. We encode strings and chromosomes to reservoir operation rule $R(\cdot)$ and reservoir release $R(x, l)$, respectively. Strings show as follows:

$$\overbrace{R(1,0), R(2,0), \dots, R(N,0)}^{l=0}, \overbrace{R(1,1), R(2,1), \dots, R(N,1)}^{l=1}$$

$$\underbrace{1, \quad 2, \dots, \quad 6, \quad 0, \quad 1, \dots, \quad 5}_{\text{operation rule } R(x,l)=\text{gene of } R^t_m(\cdot)}$$

Here, we design each chromosome $R(x, l)$ for $\forall(x, l)$ satisfying Eq.(5.5).

Each reservoir release is expressed by a decimal number. The GAs works by holding a set of strings (population) of encoded solutions to a search problem, and then tries to select and breed new solutions derived from the existing population. The members of the population are called parents and breed a new population. Typically, parents inter-breed by exchanging alleles to create a new chromosome using some form of "crossover" and "mutation". Those operations create the new strings i.e. reservoir operation rules. The operation rules which have higher fitness, i.e. loss per a period, exchange the lower fitness operation rules by selection. The process of GAs is as follows:

Step 1

Generation $t = 0$. Create initial population $G(0)$ of population size M at random.

$$G(0) = \{R_1^0(\cdot), R_2^0(\cdot), \cdots, R_M^0(\cdot)\}$$

Step 2

For each given operation rule $R_m^t(\cdot) \in G(t) = \{R_1^t(\cdot), \cdots, R_M^t(\cdot)\}$ estimate $EL(R_m^t(\cdot))$, $FR(R_m^t(\cdot))$ and $ED(R_m^t(\cdot))$ by Eqs. (5.14),(5.19) and (5.19), respectively, and set fitness to each operation rule referring to Table 5.1

If either $FR(R_m^t(\cdot))$ or $ED(R_m^t(\cdot))$ do not satisfy conditions, a local search is implemented and $R_m^t(\cdot)$ may be replaced by $\hat{R}_m^t(\cdot)$.

Step 3

Except for elitist operation rule, we apply genetic operations to every operation rule at t th generation so as to create a new population at the $t+1$ th generation, $G(t+1)$. The elitist (best)

Table 5.1 Setting of Fitness

$\overline{FR}$	$\overline{ED}$	Local Search	Fitness
satisfy	satisfy	–	$L(R(\cdot))$
not	satisfy	PIM(Eq.(5.20))	$L(\hat{R}^t_m(\cdot))$ or $L(R^t_m(\cdot))$ with Penalty α
satisfy	not	PIM(Eq.(5.20))	$L(\hat{R}^t_m(\cdot))$ or $L(R^t_m(\cdot))$ with Penalty β
not	not	–	0

operation is determined as the operation rule which has maximum fitness among those at the generation. Genetic operations consist of 'crossover' and 'mutation'. Figure 5.2 illustrates the genetic operations.

Step 4

Stop or $t = t + 1$ and go to Step 2.

$$
\begin{array}{lcl}
 & \text{cut} & \textbf{CROSSOVER} \\
\text{string A} & \underline{13233}|\underline{3334513233333345} & \longmapsto \text{string A' } \underline{13233}3334512222323345 \\
\text{string B} & 12233|333451222323345 & \longmapsto \text{string B' } 12233\underline{3334513233333345} \\
 & \text{change} & \textbf{MUTATION} \\
\text{string C} & 1223333345 1\underline{3}23333345 & \longmapsto \text{string C' } 12233333451\underline{2}23333345
\end{array}
$$

Figure 5.2 Genetic Operations

The important parts of this algorithm are how to set the fitness, how to survive the superior gene, and how to create the new gene. Selection plays an important role to induce the direction of a successful search. The genetic operator plays an important role to find new search space as a driving force. When this procedure works effectively, GAs gives full play to power.

Generally speaking, GAs is not suitable to local search. This is because it is difficult to change only tiny part of a gene, when the gene is large. However, GAs is powerful in global search by multiple point search. Thus, we propose Hybrid GAs combined PIM. We use PIM for local search, and GAs for global search. Therefore, we apply Hybrid GAs to a single reservoir operation model with multiple reliability constraints.

5.4.2. Policy Improvement Method for Local Search

In the primitive GAs applications to constraint optimization problems, algorithms may be described as genes which do not satisfy constraints that cannot survive more than one generation. This kind of description of algorithms often causes inefficient search for a better feasible solution because these algorithms may allow too small a number of genes to create the next generation.

To avoid this problem, local search is used in this paper. We focus on operation rule $R^t_m(\cdot)$ which does not satisfy one of the constraints. PIM is used to find a feasible operation rule $\hat{R}^t_m(\cdot)$ based on the infeasible one $R^t_m(\cdot)$. The local search (PIM) procedure is the following:

<u>Step 2-1</u>

Estimate $\hat{R}^t_m(\cdot)$ based on $R^t_m(\cdot)$ by

$$\hat{R}^t_m(x,l) = \begin{cases} \arg\min_{(x,l)\in\Omega(v)} \{L_{FR}(R^t_m(x,l),l) + \sum_{(y,k)} P^{R^t_m(x,l)}_{(x,l)(y,k)} w(y,k)\} \\ (FR(R^t_m(\cdot)) \geq \overline{FR}) \\ \arg\min_{(x,l)\in\Omega(v)} \{L_{PF}(R^t_m(x,l),l) + \sum_{(y,k)} P^{R^t_m(x,l)}_{(x,l)(y,l)} v(y,k)\} \\ (ED(R^t_m(\cdot)) \geq \overline{ED}) \end{cases} \tag{5.20}$$

where $w(y,k), v(y,k)$ are solutions of equations specifying the operation rule as $R^t_m(\cdot)$.

<u>Step 2-2</u>

If $\hat{R}^t_m(\cdot)$ is feasible, let $\hat{R}^t_m(\cdot)$ replace $R^t_m(\cdot)$. Otherwise, keep $\hat{R}^t_m(\cdot)$.

5.5. Numerical Examples

In this section, we apply the proposed model to a reservoir operation design problem, in order to examine our approach. It is assumed that the water demand D is $3m^3/s$, and the reservoir capacity v is $4m^3$. Inflows are assumed to obey a log-normal distribution of which the parameters have an average of $\mu = 6.25m^3/s$, and a standard deviation $\sigma = 9.65m^3/s$. The upper bound of drought frequency $\overline{FR}$ is 0.025, which is equivalent to a frequency of once in a year. The upper bound of average sojourn time of drought $\overline{ED}$ is 1.8 stage, which is equivalent to ten days. The social welfare loss function is specified as follows:

$$L(r) = \begin{cases} ae^{b(D-r)} & \text{if } r < D, \\ 0 & \text{otherwise.} \end{cases} \tag{5.21}$$

Here, social welfare loss at that stage is assumed to be a non-decreasing convex function of the deficit of $(D - R_n)$. Parameters are assumed to be $a = 0.1158$ and $b = 0.3225$.

There is a tradeoff between two constraints, FR and ED. Further, a denominator of ED is FR, and FR is restricted severely. Here, we define three types of problems, Problem 1, Problem 2 and Problem 3. Problem 1 is to solve an optimal operation rule which minimizes expected welfare loss per period, Eq.(5.7), subject to the reliability constraint of 'expected drought duration', Eq.(5.8). Problem 2 is to solve an optimal operation rule which minimizes expected welfare loss per period, Eq.(5.7), subject to the reliability constraint of 'drought frequencies', Eq.(5.9). Problem 3 is to solve an optimal operation rule which minimizes expected welfare loss per period, Eq.(5.7), subject to multiple reliability constraints of 'expected drought duration', Eq.(5.8) and 'drought frequencies', Eq.(5.9).

Figure 5.3 shows the process of expected loss per function $EL(\tilde{R}(\cdot))$, and the drought frequency $FR(\tilde{R}(\cdot))$ in Problem 1, when the elite reservoir operation rule $\tilde{R}(\cdot)$ is taken at each generation. Problem 1 is the same as GAs in the previous chapter, that is, MDP with a constraint. According to Figure 5.3, the expected loss per function $EL(\tilde{R}(\cdot))$ increased step-wise, on the other hand the drought frequency $FR(\tilde{R}(\cdot))$ repeated up and down until generation 50. This indicates that GAs still searched for an optimal solution in Problem 1.

Figure 5.4 shows the process of expected loss per function $EL(\tilde{R}(\cdot))$ and the average sojourn time of drought $ED(\tilde{R}(\cdot))$ in Problem 2, when

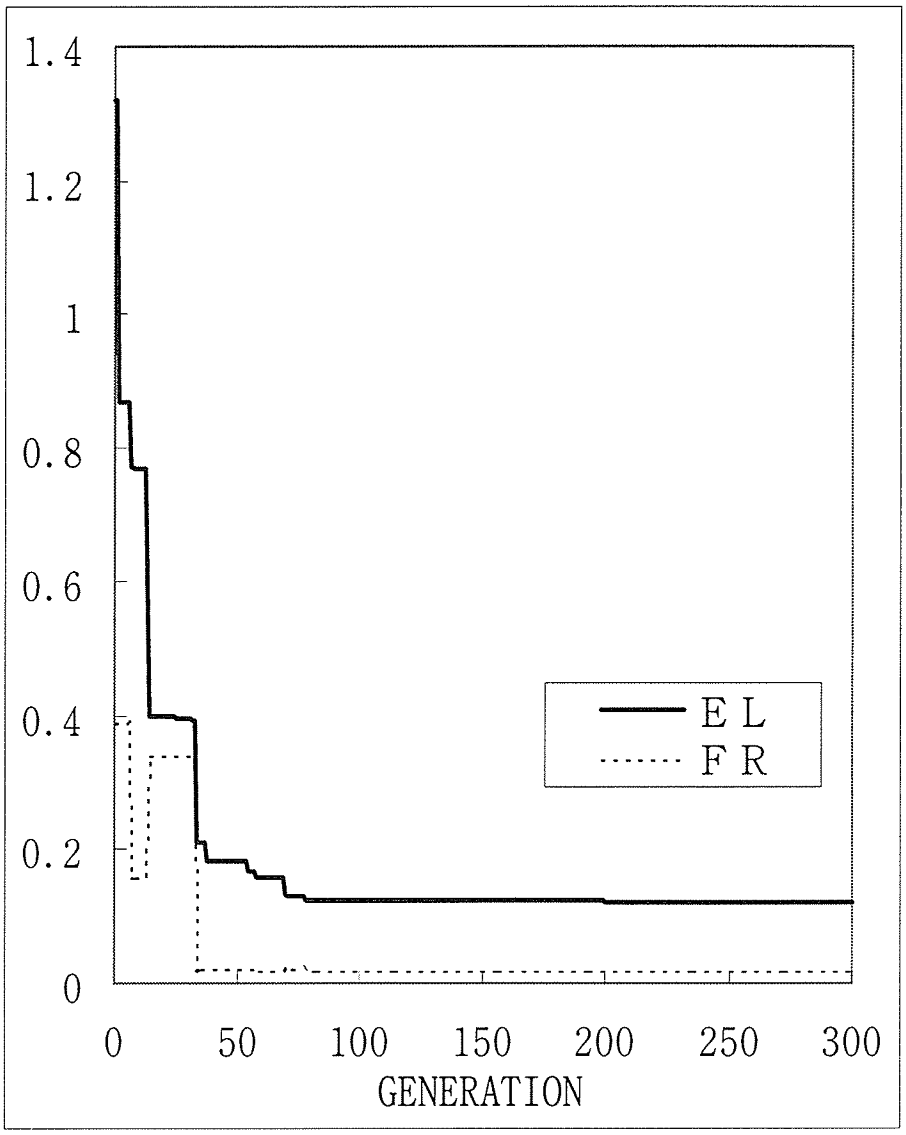


Figure 5.3 Process of $L(R(\cdot))$ and $FR(R(\cdot))$ on Problem 1

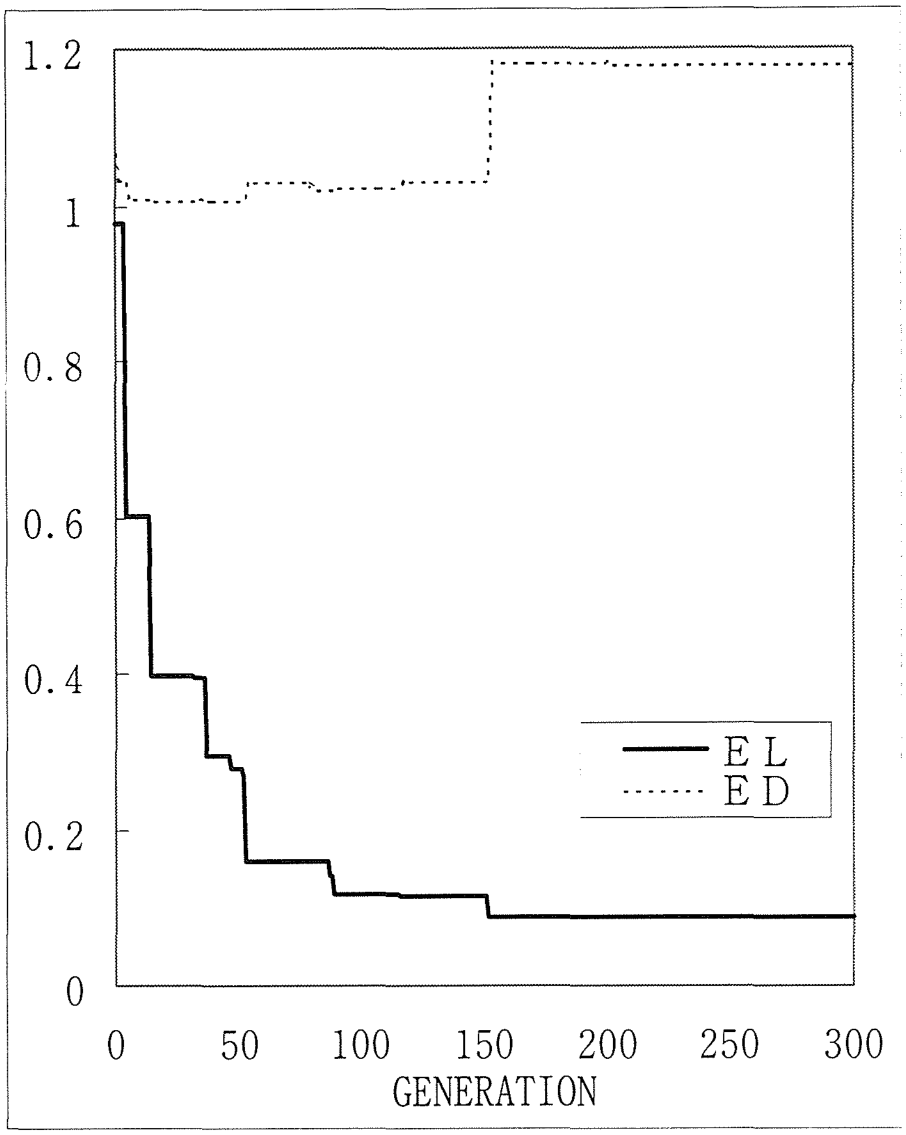


Figure 5.4 Process of $L(R(\cdot))$ and $ED(R(\cdot))$ on Problem 2

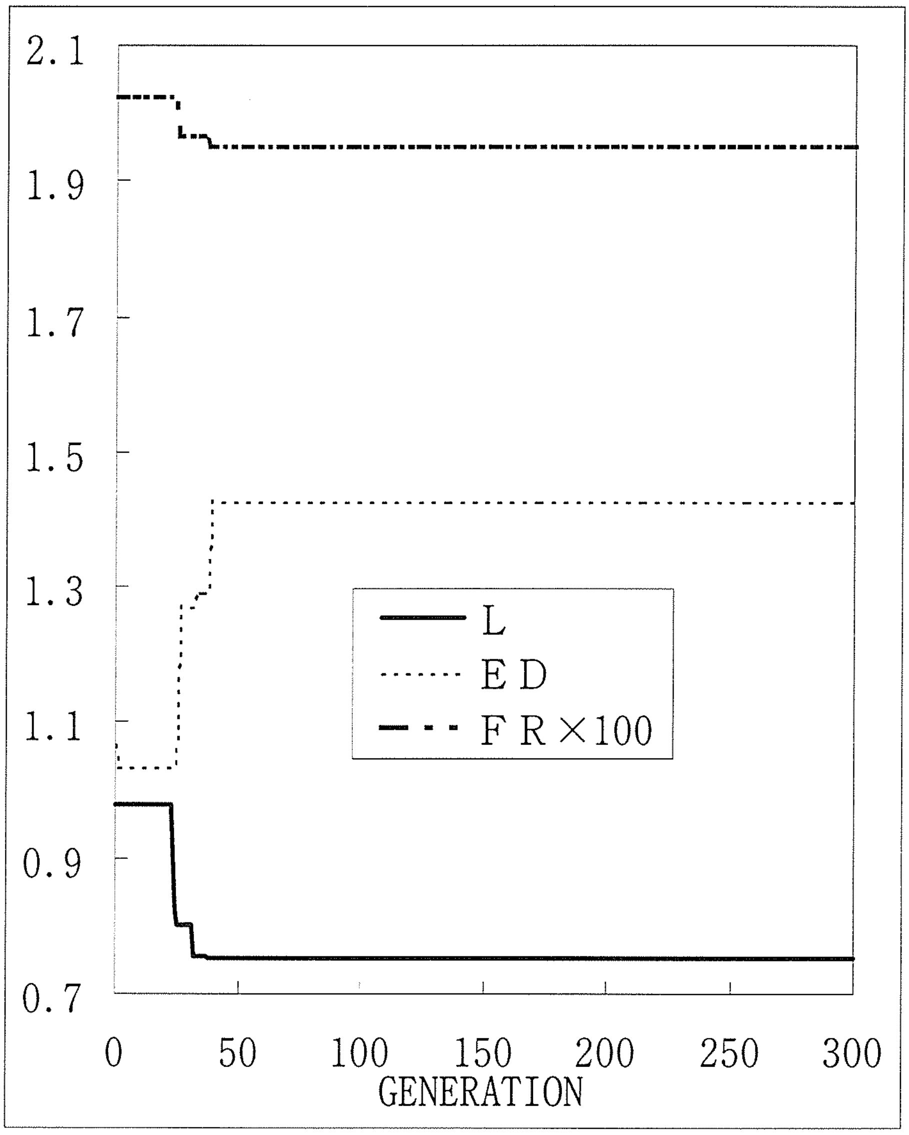


Figure 5.5 Process of $L(R(\cdot))$, $FR(R(\cdot))$ and $FR(R(\cdot))$ on Problem 3

the elite reservoir operation rule $\tilde{R}(\cdot)$ is taken at each generation. Problem 2 also has only one constraint, as does Problem 1. However, this constraint includes two functions, FR and PF. Constraint functions ED divides PF by FR. In this constraint, we are confused whether it is better to minimize ED by minimizing FR or maximizing PF. According to Figure 5.4, L was decreasing, while ED was increasing. This indicates GAs with the advantage of multi-point search is available even in this problem.

Figure 5.5 shows the process of expected loss per function $EL(\tilde{R}(\cdot))$, drought frequency $FR(\tilde{R}(\cdot))$ and the average sojourn time of drought $ED(\tilde{R}(\cdot))$ in Problem 3, when the elite reservoir operation rule $\tilde{R}(\cdot)$ is taken at each generation. Problem 3 has multiple constraints, which have tradeoffs. According to Figure 5.5, L was decreasing while FR and ED were controlled well. Therefore, GAs finds that changing FR is more effective than changing PF. This indicates that GAs can progress the objective function, while controlling multiple constraints which have tradeoffs.

Figure 5.6 is an optimal reservoir operation rule in Problem 1. This rule can be understood as a real operation rule. In safe state S, it releases water from the reservoir capacity, and the expected loss is kept as low as possible. In the drought state F, the operations save some of the water as storage in the reservoir.

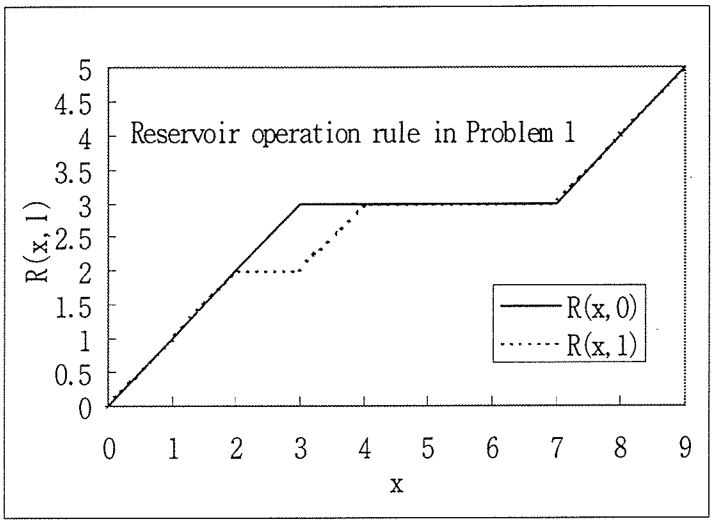


Figure 5.6 An optimized operation rule of Problem 1

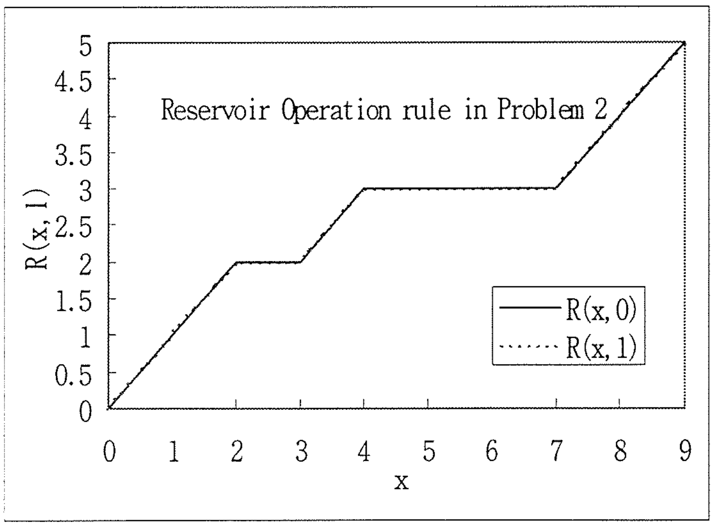


Figure 5.7 An optimized operation rule of Problem 2

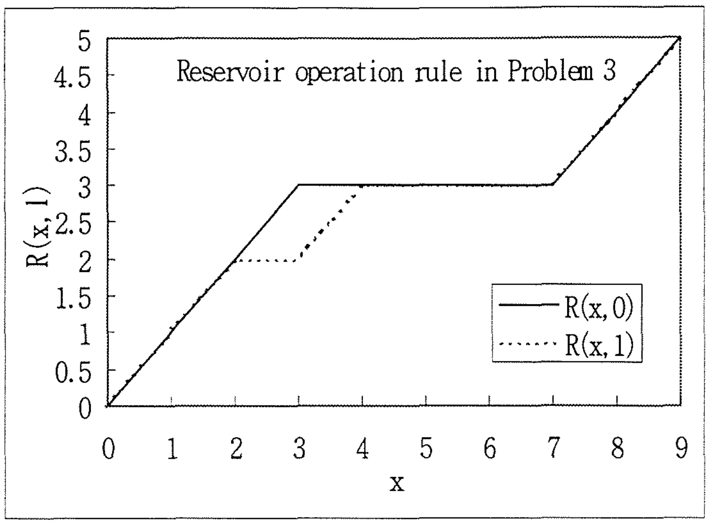


Figure 5.8 An optimized operation rule of Problem 3

Figure 5.7 is an optimal reservoir operation rule in Problem 2. According to Figure 5.7, an optimal reservoir operation rule in the safe state is the same as in the drought state F. This indicates Problem 2 must release water even in the safe state in order to satisfy the reliability constraint of 'drought frequencies'.

Figure 5.8 is an optimal reservoir operation rule in Problem 3. This rule is the same as Problem 1. This indicates FR has much effect. In order to satisfy both of the constraints, FR and ED, it should release an amount of water until the reservoir capacity is in safe state S. This operation keeps the expected loss as low as possible. By minimizing FR, L is decreasing, however, ED is increasing. Therefore, it is important to find the most suitable FR in Problem 3.

Table 5.2 shows the results for different combinations of constraint change.

Table 5.2 Results of three problems

	EL	FR	ED	GENERATION
Problem 1	0.1205	0.0196	–	391
Problem 2	0.0867	–	1.1788	286
Problem 3	0.1205	0.0196	1.6650	286

5.6. Conclusion

In this paper, we proposed hybrid GA which combined with PIM in an operation optimization model with multiple reliability constraints. We have gotten successful results. Here, PIM is used once per improvement of an operation rule. However, it is possible to find an operation rule which satisfies the constraint by retrying. We are sure that hybrid GAs can be used as a tool for a MDP model with multiple constraints. However, we think it is important to consider the way of setting fitness for effective search. It would be of benefit to continue the research of this problem.

Chapter 6.

CONCLUSION

In this thesis, various combinatorial optimization problems are solved by Genetic Algorithms. The modeling methods are different for each problem's characteristics, however, all of the Genetic Algorithms proposed also fall into a category of a Hybrid GAs in the sense. Genetic Algorithms are easy to model and combine with other optimization methods. However, it must be considered how to combine various optimization methods into Genetic Algorithms. There are many combinatorial optimization problems, in the real world. In this thesis, we have dealt with rectangular packing problems and natural numbering problems.

Rectangular packing problems are difficult to translate from a gene to a feasible solution which satisfies constraints, because a geometrical factor is included. Then, in a two-dimensional rectangular packing problem, we proposed that Genetic Algorithms be applied as parameter tuning of greedy algorithm. Moreover, we formulated a slab design problem as a two-dimensional bin-packing problem. Genetic Algorithms are applied as assorting information of heuristics. In both problems, Genetic Algorithms is give good results.

Natural number combinatorial optimization problems are formulated with as a Markov decision process model with constraints. Moreover, a reservoir operation problem is dealt with a Markov decision process model with constraints. Natural number combinatorial optimization problems are difficult to satisfy with constraints. However, transfer from a gene to a feasible solution is easy, due to a phenotype. That is, a solution is equal to a gene in natural number combinatorial optimization problems. In natural number combinatorial optimization problems, Genetic Algorithms also give good results. However, it is unknown whether phenotypes satisfy all constraints. Thus, it is important to change an infeasible solution to a feasible solution.

All of the Genetic Algorithms proposed in this thesis, have given good results. We are sure that Genetic Algorithms are a suitable tool for solving combinatorial problems.

ACKNOWLEDGEMENT

The author would like to thank Prof.H. Kawai of Tottori University for supervising this dissertation. His continuing guidance and encouragement on the dissertation have greatly helped the completion of the work.

The author is also grateful to Prof.K. Sugata, Prof.H. Kita, of Tottori University and Prof.K. Kobayashi of Kyoto University for their generous support of the work.

This dissertation could not have been completed without the guidance and encouragement of the above professors.

Furthermore, the author wishes to express his thanks to Prof.H. Tatano and Dr.K. Fukuyama for his helpful advice. His excellent assistance greatly helped to get the reservoir operation optimization problem into shape.

Prof.H. Tokuyama of Shizuoka University, Dr.T. Makino, Dr.N. Ueno, Mr.Y. Nakagawa and Mr.S. Sotojima of Sumitomo Metal Industries, Ltd. have contributed knowledge and support with regard to the slab design problem.

The author also wishes to express his thanks to Mr.S. Wever for his help with technical writing in English.

The author's thanks are also go to assistant Prof.J. Koyanagi and all of the members of Prof.H. Kawai's laboratory at Tottori University.

Bibliography

[1] Kantorovitch, L. V., “Mathematical methods of organising and planning production”, Management Science 6, pp. 366-422, (1960).

[2] Gilmore, P. C. and Gomory, R. E., “Multistage cutting stock problems of two and more dimensions”, Operations Research 13, pp. 94-120, (1961).

[3] Kathryn A. Dowsland and William B. Dowsland, “Packing problems ”, European Journal of Operational Research 56, pp. 2-16, (1992).

[4] H. Tokuyama and N. Ueno,“The cutting stock problem for large sections in the iron and steel industries”, European Journal of Operational Research 22, pp. 280-292, (1985).

[5] P. K. Agrawal, “Minimising trim loss in cutting rectangular blanks of a single size from a rectangular sheet using orthogonal guillotine cuts” European Journal of Operational Research 64, pp. 410-422, (1993).

[6] Ahmed El-Vouri, Neil Popplewell, S. Balakrishnan, Attahiru S. Alfa, "A search-based heuristic for the two-dimensional bin-packing problem", INFOR, Vol.32 No.4, (1994).

[7] Paull, A. and Walter, J., "The trim problem: an application of linear programming to the manufacture of newsprint paper", Proceedings of Annual Econometric Meeting, Montreal, Sept. 10th-13th, (1954).

[8] Baker, B.S. and Coffman, E.G., "A tight asymptotic bound for next-fit-decreasing bin-packing", SIAM Journal on Algebraic and Discrete Methods 2, pp. 147-152, (1981).

[9] David S. Johnson, "Fast Algorithms for Bin Packing", Journal of Computer and System Science, pp. 272-314, (1974).

[10] Coffman, E.G., Garey, M.R and Johnson, D.S., "Approximation algorithms for bin packing -an updated survey", Algorithm Design for Computer Systems Design, pp. 49-106, (1984).

[11] Wang. P.Y. , "Two algorithms for constrained two-dimensional cutting stock problems", Operations Research 31, pp. 46-49, (1983).

[12] Bellman, R. , A Markovian Decision Process, Princeton Univ. Press, Princeton, New Jersey, (1957).

[13] Howard, R.A., Dynamic Programming and Markov Processes, M.I.T. Press, Cambridge, (1960).

[14] Hordijk, A. and Kallenberg, L.C.M., "Constrained Undiscounted Stochchastic Dynamic Programming", Mathematics of Operations Research, Vol. 22, pp. 276-289, (1984).

[15] Beutler, F.J. and Ross, K.W., "Time-Average Optimal Constrained Semi-Markov Decision Processes", Advances in Applied Probability, Vol. 18, pp. 341-359, (1986).

[16] T. Kawakami and Y.Kakazu, "A GA-based hierarchical tuning of the 3-D packing strategy in a multiagent environment", 1994 Japan U.S.A Synposium on Flexible Automation, A Pacific Rim Conference , pp. 1319-1326, (1994).

[17] Coffman, E.G., Jr., and Shor, P.W., "Average-case analysis of cutting and packing in two dimensions", European Journal of Operational Research 44, pp. 134-145, (1990).

[18] Y. Masumoto, K. Yosikawa and N. Kato, " Realistic evaluation of two-dimensional bin-packing algorithm " (in Japanese), Proc. of Operational Research in Japan, pp. 42-43, (1986).

[19] H. Tokuyama, N. Ueno, M. Kawabata, Y. Kitano and H. Saito, "The two-dimensional assortment problem in the production of heavy steel plates", Asian-Pasific Operations Research: APORS '88, pp. 289-304, (1988).

[20] B. Watson, K. Matsuda, K. Nose, M. Konishi, Y. Tomita, S. Sakai and S. Sasaki, “Solution of the rectangular cutting stock problem by genetic algorithm”, 1992 Pacific Conference on Manufacturing Proceedings, pp 523-530, (1992).

[21] Toshihide Ibaraki, “New extend of modern heuristics – Genetic Algorithm, Simulated Annealing, Tabu Search,Neural Net is effective ? – Operations Research Society of Japan 30th symposium (in Japanese)”, pp. 1–10, (1993).

[22] Hiroaki Kitano, Genetic Algorithm (in Japanese), Sangyotosho, (1993).

[23] David. E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Wesley Publishing Company Inc, (1989).

[24] Beutler, F.J. and Ross, K.W.,“Optimal Policies for Controlled Markov Chains with a Constraint”, J. Math. Anal. Appl., Vol. 112, pp. 236–252, (1985).

[25] Toshio Kitagawa, Markov Process (In Japanese), Kyoritsushupan, (1967).

[26] Moran, P.A.P., “A Probability Theory of Dams and Storage System ”, Aust. Jour. Applied Science, Vol. 5, pp. 116-124, (1954).

[27] Lloyd, E. H., "Reservoirs with Serially Correlated Inflows", Technometrics, Vol. 5, No. 1, pp. 85-93, (1963).

[28] Prabhu, N. U., "Time-dependent Results in Storage Theory", Jour. Applied Probability, Vol. 1, pp. 1-46, (1964).

[29] Hashimoto T., Stedinger, J. R. and Loucks D. P., "Reliability, resiliency, vulnerability criteria for water resource systems performance", evaluation, Water Resour. Res., Vol. 18, No. 1, pp. 14-20, (1982).

[30] Yakowitz, S., "Dynamic Programming Applications in Water Resources", Water Resour. Res., Vol. 18, No. 4, pp. 673-696, (1982).

[31] Yeh, W., "Reservoir management and operations models : a state-of-the-art review", Water Resour. Res., Vol. 21, No. 12, pp. 1797-1818, (1985).

[32] Askew, A.J., "Optimum reservoir operating policies and the imposition of a reliability constraint", Water Resour. Res., 10(1), pp. 51-56, (1974).

[33] Askew, A.J., "Chance-constrained dynamic programing and the optimization of water resources system", Water Resour. Res., 10(6), pp. 1099-1106, (1974).

[34] Rossman, L., "Reliability-constrained dynamic programming and randomized release rules in reservoir management", Water Resour. Res., 13(2), pp. 247-255, (1977).

[35] Sniedovich, M., "Reliability-constrained reservoir control problems, 1, Methodological issues", Water Resour. Res., 15(6), pp. 1574-1582, (1979).

[36] Tatano, H., Okada, K., Yoshikawa, K. and Kawai, H, "A frequency and duration contrained model for the optimization of a single reservoir operation", in Hipel, K.W. and Fang, L. (eds.) Stochastic and Statistical Methods in Hydrology and Environmental Engineering, Kluwer Academic Publishers, Netherlands, Vol. 4, pp. 375-388, (1994).

END