A Study on Evolutionary Optimization Method

Naoyuki KUBOTA

A Study on Evolutionary Optimization Method

洋　1203444

Naoyuki KUBOTA

To my parents, my sister, and Emí

Acknowledgments

I wish to express my deepest thanks to my chief advisor Prof. Toshio Fukuda of Department of Micro System Engineering, Nagoya University, who gave me a great guidance, numerous useful advices, and helpful suggestions throughout this dissertation. I wish to express my sincere thanks to Prof. H.Ota of Department of Mechanical Engineering and Prof. Y.Hayakawa of Department of Electric Mechanical Engineering for their valuable advices and comments to revise this dissertation.

I gratefully appreciate Assistant Prof. F.Arai, Mr. H.Matsuura and Dr. H.Ishihara of Department of Micro System Engineering, and Dr. K.Shimojima of National Industrial Research Institute of Nagoya for their fruitful discussions. I also appreciate Prof. K.Kosuge of Tohoku University for his helpful advice.

I acknowledge Prof. N.Baba and Prof. T.Taniguchi of Osaka Kyoiku University for their heartfelt guidances and advices during my undergraduate of Osaka Kyoiku University. I also acknowledge Prof. T.Date of Hokkaido University for his heartfelt guidance and advice during my master course of graduate school of Hokkaido University.

I wish to thank Dr. D.B.Fogel of Natural Selection Inc. for his useful advices and comments, and I also wish to thank Dr. J.Sturke of University of Stuttared for his fruitful comments and discussion.

I thank Dr. N.Mitsumoto, Mr. K.Sekiyma, Mr. A.Cai, Mr. H.Ishigami, Mr.T.Arakawa, Mr. K.Morishima and Mr. F.Kobayashi for their meaningful discussion. I also thank Dr. H.Shioya, Mr. T.Kojima and Mr. T.Kawasaki of Hokkaido University Mr. T.Ito of Osaka University, Mr. Y.Hasegawa of Mitsubishi Heavy Industry Ltd. for their useful discussion. I also thank all secretaries of Fukuda laboratory for their supports in my miscellaneous work. Furthermore, I thank my colleagues and all members of the Fukuda Laboratory for my meaningful life in the university.

Finally I owe my lasting gratitude to my parents, my sister and my uncle who helped me to continue my undergraduate and graduate studies.

Contents

Chapter 1 Introduction

The essential of *Engineering* lies in optimization and invention, while the essential of *Science* is to discover the law of *Nature*. The optimization in the engineering is to produce something more suitable, more precise, and speedier. In the growing complex human society, especially, industry and commerce, the concept of optimization is very important for product design, resource allocation, plant location, manufacturing scheduling, and so on. With the progress of computing power, the optimization has been recently done by using computational technology, though the previous optimization had been done by human knowledge of experts. In fact, the computer contributes greatly to the current human society, and the computer power keeps increasing while the size and cost of the computer keep decreasing. As a result, we can solve various complex problems quickly and effectively and therefore, we require the powerful and quick solution methods. This chapter briefly presents optimization methods and the related fields such as operations research, artificial intelligence, evolutionary computation, soft computing, and artificial life.

1.1 Background on Optimization

Optimization plays a very important role in various fields and today its application can be seen in business, education, government, industry, and so on. Therefore the origin of the optimization can be retraced the history of all the fields such as engineering, science, economics and sociology. From the historical point of view, the development of the computer can be the latest revolution for the optimization. The computer provides us quick computing, advanced information processing, and numerous data storing [1].

Before the development of the computer, a systematic approach can be regarded as a methodological revolution. The system analysis by logical and mathematical methods provides us effective solutions and insights for large complex problems. For example, operations research is well-known as one of the systematic approach [3,4]. The operations research was first successfully used for the analysis of military operations at World War II. The operations research is, in general, the use of quantitative methods to analyze and predict the behavior of systems which are influenced by human decisions. In fact, the operations research applies optimization methods such as linear programming and dynamic programming for decision

making.

When retracing the history of the optimization furthermore, we can reach the fact that the optimization was done by human knowledge of experts, and the human knowledge for the optimization was inherited from ancestors to offspring. This kind of human knowledge is often called *heuristics*. In addition, there have been a lot of historical facts concerning the optimization. Thus, we can see a lot of facts concerning the optimization in the human history.

Table 1.1 Optimization and related fields from the historical points of view

	Fuzzy System / Neural Network	Computer / Artificial Intelligent	Evolutionary Computation	Others
1940	McCulloch-Pitts Neuron Model Hebbian Learning	Programmable Computer		Operations Research Cybernetics
1950	Perceptron	Turing Test Lisp	Machine Evolution	Metropolis Algorithm Dynamic Programming Monte Calro Method
1960	Fuzzy Sets	General Problem Solver A* Search	Evolutionary Programming Evolution Strategy	Random Search Branch-And-Bound Technique
1970	Back-Propagation Algorithm Fuzzy Controller	NP-Completeness Expert System	Genetic Algorithm Classifier System	
1980	Hopfield Network Fuzzy Modeling Boltzmann Machine	Belief Network	Bucket Brigade Algorithm	Artificial Life Immune System Tabu Search
1990	Neuro-Fuzzy	Qualitative Probabilistic Network	Genetic Programming	Q-Learning

On the other hand, from the viewpoint of related fields, the optimization has been developed together with operations research, evolutionary computation, artificial intelligence, neural network, fuzzy logic, and so on (Table 1.1). In addition to the dynamic programming as mentioned before, random methods such as metropolis algorithm and Monte Calro method were proposed in 1950s. Furthermore, A* search and branch-and-bound technique were proposed to search for solutions effectively. After that, evolutionary computation such as evolutionary programming, evolution strategy, and genetic algorithm, has been applied to various optimization problems. In recent years, reinforcement learning such as temporal difference learning and Q-learning has well discussed. Furthermore, these kinds of optimization methods are applied to neural network, fuzzy system, and so on. In this way, the optimization has been developed through the interaction with these related fields. The following sections present the optimization and these related fields in brief.

1.2 Optimization, System Engineering and Intelligent Manufacturing System

In this section, we consider what the role of the optimization is in the engineering. The systematic approach in the previous section is explicitly discussed in the system engineering, though the systematic approach is utilized widely in various fields. A system is generally defined as an assemblage of functionally interacting components, which is satisfied the following conditions: (1) having two or more components, (2) accomplishing an object as a whole, and (3) each component achieves each function. In fact, the effectiveness of all the components regarded as a system may be greater than the sum of the separated effectiveness of each components.

The system engineering has been gradually developed on the basis of operations research, information theory, cybernetics, computer science, and so on. *Information theory* provides us the concepts such as reliability and redundancy based on entropy [5]. In addition to entropy, Markov chain is very useful for prediction. *Cybernetics*, which was proposed by N.Weiner, is a theoretical study of communication and control processes in biological, mechanical, and electronic systems and the concept of feedback is very important [6]. A current computer is regarded as Turing machine based on the concept of automaton which produces some output in response to an input according to its internal state. And V.Noman proposed a stored program computer based on the Turing machine [1].

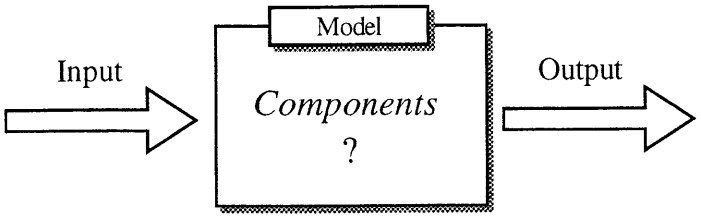


Figure 1.1 System synthesis: Design of components

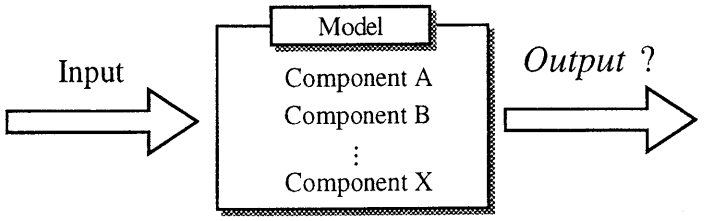


Figure 1.2 System analysis: Examination by quantitative/qualitative methods

The system engineering has two aspects of (1) *system synthesis* and (2) *system analysis*. The system synthesis combines separate components to form a coherent system (Figure 1.1). The system analysis examines methodically by using both quantitative methods and qualitative methods (Figure 1.2). In general, the system synthesis is far more difficult than the system analysis since the system synthesis requires system design, simulation, optimization and evaluation. In fact, when a problem is given, the procedure to develop a system for the problem is as follows:

Step 1. Definition and formulation
Step 2. Development of alternative solutions
Step 3. Modeling of alternative solutions
Step 4. Determination of the evaluation function
Step 5. Optimization and evaluation

In step 1, we define and formulize the problem to clarify the structure. Second, we develop some alternative solutions for solving the problem. Next, we build models of alternative solutions from the results of the cause-effect analysis of the problem. Next, we determine the evaluation function and evaluation criterion for the problem [7]. Finally, we optimize each model and compare the evaluation result. As a result, we can obtain a desired system. Thus, the optimization plays the important roles, and the decision making selecting the best alternative depends on the optimization results. In general, an optimization problem is defined as follows,

$$\text{minimize(maximize)} \quad f(x)$$

$$\text{subject to} \quad \begin{cases} g_i(x) \le 0 & (i = 1,\text{L}\ ,l) \\ h_j(x) = 0 & (j = 1,\text{L}\ ,m) \end{cases} \tag{1.1}$$

where x is a set of decision variables, $f(x)$ is called objective function, which includes effectiveness and cost functions, and $g_i(x)$, $h_j(x)$ are called constraints. In a minimization problem, the optimization is to find the decision variables x^* which satisfies the following equation,

$$f(x^*) \le f(x) \tag{1.2}$$

The objective of the optimization is, in general, to obtain the best values which maximize the effectiveness and which minimize the cost at the same time. However, the optimization problem usually has some trade-off between the effectiveness function and the cost function. It is therefore very difficult to optimize the objective function. In general, most of problems to be dealt with in engineering, science and sociology, can result in this kind of optimization problems. For examples, a current manufacturing system has many complex optimization problems such as product design, resource allocation, plant location, manufacturing scheduling since the current manufacturing system has a lot of constraints such as space capacity, machining ability and time restriction [2]. Most of these optimization problems belong to ill-defined and/or ill-structured problems. To deal with these kind of complex problems effectively, intelligent manufacturing system has been discussed recently [2].

The intelligent manufacturing system comprises a new concept for coping with a large number of products and manufacturing processes, and its effectiveness lies in the processing capability corresponding to high variety. Therefore, the intelligent manufacturing system requires the powerful and quick optimization methods to create an ideal manufacturing environment. In fact, the intelligent manufacturing system has various complex problems such as resource allocation problems with prediction, relocation problems of machining centers, path planning for automated guided vehicles and dynamic manufacturing scheduling problems.

1.3 Optimization Methods

Various types of optimization methods have been developed so far and the optimization methods can be divided into two categories of exact and approximate methods. The exact method can obtain optimal solutions using local information peculiar to the optimization problem, but the exact method needs much computation cost. On the other hand, the approximate method can obtain optimal or quasi-optimal solutions with less computation cost. Thus, each optimization method has a trade-off between the performance of solution and the computation cost. There are four criteria to evaluate the optimization methods, i.e., completeness, time complexity, space complexity and optimality [3].

On the other hand, from the mathematical point of view, the optimization methods can be divided into three main categories: (1) *numerical methods* [8] such as simplex methods, bisection line search and steepest decent method, (2) *enumeration methods* [3,4,9] such as dynamic programming and branch-and-bound algorithm, and (3) *random methods* [10~12] such as Monte Carlo method, simulated annealing and genetic algorithm.

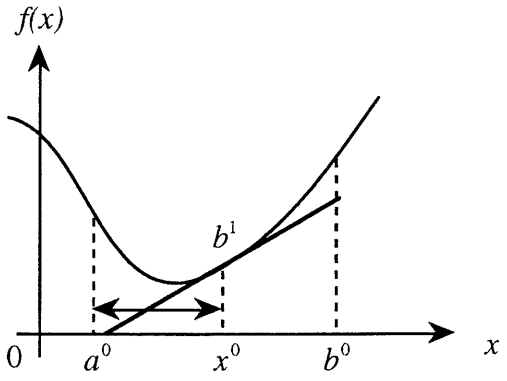


Figure 1.3 Bisection line search

1.3.1 Optimization by Numerical Methods

The numerical methods are mainly known as linear programming and non-linear programming. The optimization methods by non-linear programming have been studied heavily so far. These methods are based on iterative hill-climbing by using a descent direction. In this

section, we focus on a minimization problem of an objective function. One of the simplest techniques is a bisection line search (Figure 1.3) [3]. The basic idea is to search for a point x satisfying $df(x)/dx = 0$. First, we initialize two starting points, a^0 and b^0. Next, find the midpoint $x^0 = (a^0 + b^0)/2$, and evaluate the derivative. If the derivative at a point x^0 is positive, then the minimum is between a^0 and x^0 and the next right point $b^1 = x^0$. Otherwise, $a^1 = x^0$. This process is repeated until $df(x)/dx = 0$. The bisection line search can converge to a global minimum when the objective function is convex.

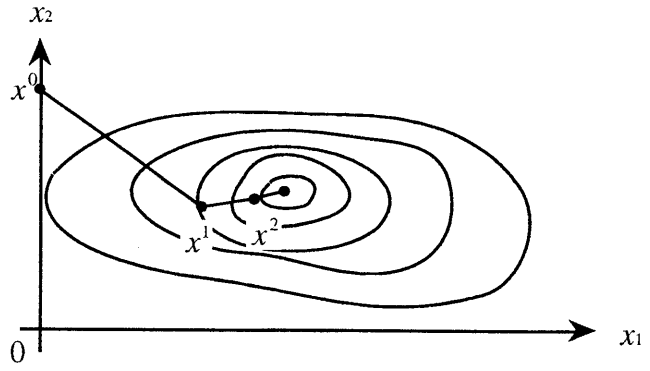


Figure 1.4 Steepest descent method

When we consider an objective function with more than one variables, the most used method is a steepest descent method (Figure 1.4). The gradient of the objective function always points the ascent direction, and therefore the opposite direction is a descent one for minimizing the objective function. Consequently, the next candidate solution is calculated by the following equation,

$$x^{i+1} = x^i - \alpha \cdot \frac{df(x^i)}{dx} \tag{1.3}$$

where α is a step size which is non-negative. The optimal value of α can be found by solving the following equation,

$$\text{minimize} \quad f(x^{i+1}) = f(\alpha) \tag{1.4}$$

that is, by solving the equation $df(\alpha)/d\alpha = 0$, we can obtain the optimal α. These process is repeated until $df(x^{i+1})/dx = 0$. However, the steepest descent method has problems of trapping to local minima easily and requiring many iterations [7].

In addition, when we can use the second derivatives of the objective function, we can use a Newton's method. First, the objective function $f(x)$ is approximated according to Taylor series expansion up to the second-order,

$$f(x) \cong f(x^i) + \frac{df(x^i)}{dx}(x - x^i) + \frac{1}{2}(x - x^i)^T H(x^i)(x - x^i) \tag{1.5}$$

where $H(x^i)$ is the Hessian matrix which consists of the second partial derivatives of $f(x)$. The point minimizing the objective function $f(x)$ is equal to the following point x^{i+1} by differentiating eq.(1.5) with respect to x and setting it to zero.

$$\frac{df(x^i)}{dx} + H(x^i)(x - x^i) = 0 \tag{1.6}$$

Next, by inserting x^{i+1} into x, we obtain the minimization point x^{i+1} of the approximated objective function.

$$x^{i+1} = x^i - H(x^i)^{-1}\frac{df(x^i)}{dx} \tag{1.7}$$

If the $H(x^i)^{-1}$ exists, we can obtain a unique solution. Though the Newton's method can obtain a minimization point by only one iteration, the Newton's method has problems of the difficulty of calculating a Hessian matrix and its inverse matrix, and numerical problems due to round-off errors. To improve Newton's method, quasi-Newton's methods and others have been proposed.

1.3.2 Optimization by Enumeration Methods

The enumeration methods can find the best solution by counting out one by one, but the enumeration methods requires much computation time. To reduce computation cost, dynamic programming and branch-and-bound algorithm have been proposed. The dynamic programming is often used for solving a certain problem requiring sequential decision that made be at various stage based on the *principle of optimality* by R.Bellman, that is,

> an optimal policy has the property that whatever the current state and decision are, the remaining decisions must constitute on optimal policy with regard to the state resulting from the current decision [3].

On the other hand, the branch-and-bound method basically divides the optimization problem into some subproblems and solves the subproblems [3]. The procedure dividing subproblems is called a *branching* operation. Next, the subproblem, which has no possibility of containing optimal solutions, is excluded from possible subproblems. This procedure is called a *bounding* operation. In the branch-and-bound method, the branching operation is very important since the number of search times depends on the branching operation. In general, there are best-first search, breadth-first search and depth-first search in a sequence of subproblems that differ from one another. However, these methods require much computation cost as the problem size is large, though these methods can obtain optima within finite search times.

1.3.3 Optimization by Random Methods

The random method is basically a hill-climbing search which moves in the direction of increasing an evaluation value in case of a maximization problem. The random methods can find global optima within infinite time without trapping to local optima in solving non-convex optimization problems [7]. However, the global convergence is not important because the enumeration method in a finite search space can find global optima in finite time. The importance is that the random methods can obtain quasi-optima with less computation cost.

In addition, the random methods have two approaches of local search and global search. The local search selects a next searching point out of the neighborhood of the current point, while the global search selects a next searching point out of all solution space. In general, the computation time for the search depends on how to design the ratio of the global search to the

local search. The local search can fast attain local optima, while the global search can slowly attain global optima. On the other hand, random methods that the search ratio is well designed, are simulated annealing [11] and genetic algorithm [12]. These methods, which are called *modern heuristics*, are optimization methods simulating physical phenomenon and biological evolution, respectively.

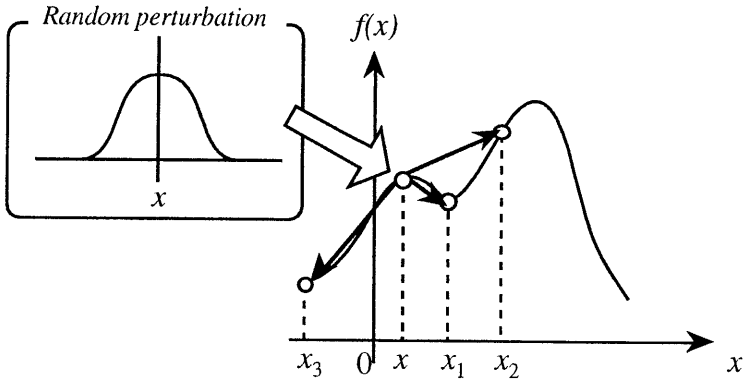


Figure 1.5 Simulated annealing algorithm

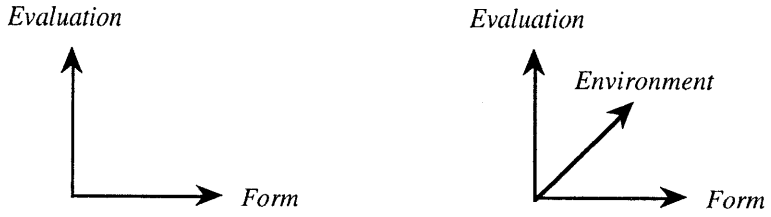
The simulated annealing is a hill-climbing method which can move in the downhill direction. The simulated annealing generates a next point, x_{i+1}, with random perturbation and the next point is evaluated. And then, the current point, x_i, is replaced with the next point according to the Metropolis or equivalent criterion. Metropolis algorithm uses the probability, $\max(1, e^{\Delta f/T})$, where Δf is the difference between $f(x_{i+1})$ and $f(x_i)$. If $\Delta f > 0$ then the current point is replaced with the next one. Otherwise, the current point is replaced with the next one with the probability $e^{\Delta f/T}$. Another frequently used criterion is the Boltzmann probability distribution, $1/(1+e^{\Delta f/T})$. Consequently, while it is easy to move in the downhill direction when the temperature, T, is high, the search point move only in the hill-climbing direction when the temperature is low. In addition, the search ratio depends on the cooling schedule of the temperature.

On the other hand, the genetic algorithm simulates the process of evolution and genetic operation. The next section presents the genetic algorithm and evolutionary computation. Thus, nature often provides us effective ideas for engineering.

1.4 Optimization and Evolution

1.4.1 Evolution on Engineering

Living things in nature evolve and adapt to their external environments. If it is possible to simulate evolution on a computer, then we can realize an adaptive system. First of all, we consider the terms concerning *Evolution* and *Adaptation* on Engineering.



(a) Evaluation of change (b) Evaluation of adaptation

Figure 1.6 Evaluations of change and adaptation

- *Change.* The process or result of giving a different form or appearance. Here the form includes figure, pattern and shape of cells, function, organization, culture and environments. Change can be evaluated under an evaluation criterion as evolution or degeneration.

$$Evaluation = f_{criterion}(form) \tag{1.8}$$

The evaluation result determines evolution or degeneration (Figure 1.6.a).

- *Evolution*: a progression from a simple form to a more complex one.
- *Degeneration*: a change from a complex form to a simpler one.

Here we can not judge whether or not the evolving one adapts to its environment from the evaluation result. Whether or not one adapts to its environment, depends on the environmental condition.

- *Adaptation.* Adjustment in structure or function to a changing environment

(Figure 1.6.b). Adaptation can be evaluated under an evaluation criterion and environmental factors.

$$Evaluation = f_{criterion}(form \circ environment) \tag{1.9}$$

where '$\circ$' is an operator. The adaptation is not evaluated without environmental factors since the adaptation is evaluated as the result of the interaction with the environment .

Here the important point is that an evolved one is not necessarily the adapted one (Figure 1.7), though evolution is often regarded as a subset of the adaptation in engineering. For example, human being can be said to adapt to the current environment on earth, but human being lost the ability of generating vitamin in the human body through the process of evolution. The lost ability can be said to be degeneration. Thus, the evolution and degeneration occur in nature as the results of the adaptation. Another example is a numerical control (NC) machine in the intelligent manufacturing system. The NC machine adapted to a machining environment may not adapt it to another one. On the other hand, the machine can be said to be evolved, if the processing speed of a computer in the NC machine is higher than other NC machines. Consequently, the evolution and adaptation can not be evaluated under the same evaluation criterion at the same time.

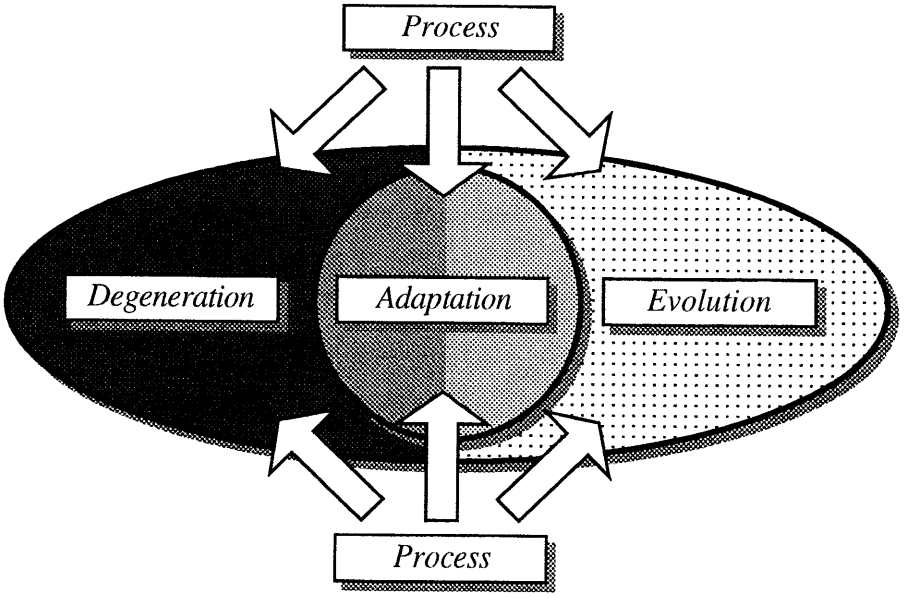


Figure 1.7 Evolution and adaptation

Next, we consider the process of evolution and adaptation.

- *Process*. A series of actions, changes, or functions bringing about a result. The process includes inheritance and learning.

  - *Inheritance*. The process of transmission of genetic information from parents to offspring. As transmitting errors, *mutation* changes genetic information.
  - *Leaning*. The act, process, or experience of gaining knowledge or skill. Learning includes *learning by discovery, learning by show* from parents, and *reinforced learning* through the interaction with the environment.

1.4.2 Evolutionary Computation

Evolutionary computation is a field of simulating evolution on a computer [13~17]. As mentioned before, an optimization problem is defined as eq.(1.1). Consequently, the evolutionary optimization method can be defined as an optimization method by using eq.(1.8) as an evaluation criterion, since the optimization problem, in general, has neither input nor output from the environment. However, if the objective of the optimization problem is to model a controller and to acquire rules and skill through a computer simulation, the evolutionary optimization method requires eq.(1.9).

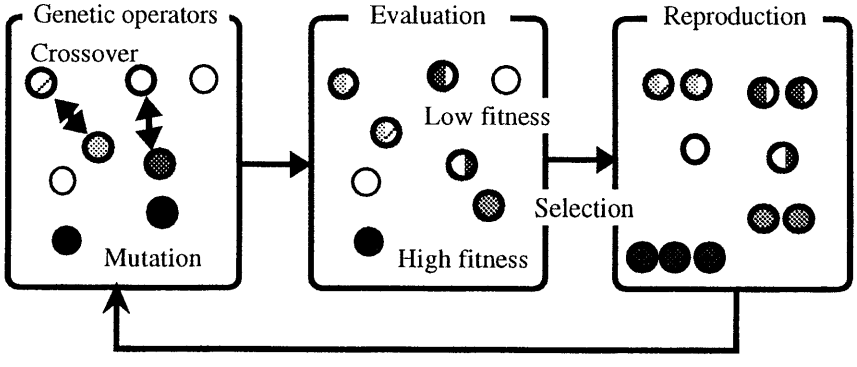


Figure 1.8 Evolution of population in genetic algorithm

From the historical point of view, the evolutionary optimization methods are divided into three main categories: genetic algorithm by J.Holland [14], evolutionary programming by L.Fogel [13], and evolution strategy by H.Schwefel and I.Rechenberg [15, 16]. These algorithms are fundamentally iterative generation and alternation processes operating on a population of candidate solutions. In addition, these evolutionary optimization methods have been applied successfully to various optimization problems so far. Figure 1.8 shows the evolution of a population in the genetic algorithm. The genetic algorithm generates new candidate solutions by using genetic operators based on the current candidate solutions, while the random search randomly generates new candidate solutions. Next, the genetic algorithm selects next candidate solutions according to fitness value. In this way, the fitter candidate solutions survive and generate better candidate solutions.

1.5 Artificial Intelligence and Optimization

With the progress of computation ability, various methods concerning artificial intelligence have been successfully proposed. The study of intelligence can be retraced to very old philosophy. From over 2000 years, reasoning theory and learning theory have been discussed by Plato, Socrates and Aristotle [8]. The first work of artificial intelligence is a model of artificial neurons which was proposed by McCulloch and Pitts. On the other hand, Shannon and Turing were writing chess program for a computer, and Samuel wrote a computer program that eventually learned to play checkers. At that time, a high level computer language, Lisp, was developed. The objective of early artificial intelligence is problem-solving. One of the attempts was general problem solver (GPS) by Newell and Simon. The main methods of GPS embody the heuristics of means-ends analysis. After that, a lot of knowledge representation method, search strategies and reasoning methods have been proposed for problem-solving.

The ultimate goals of the artificial intelligence are to understand the mechanism of brain and to realize human intelligence on a machine. The goals are scientific and engineering approaches, respectively. In fact, the problem of artificial intelligence is to describe and build an intelligent agent, which perceives its environment, make a decision, and performs action (Figure 1.9). Here the environment includes deterministic, dynamic and continuous elements. The researches concerning the artificial intelligence can be divided into three fields;

1. knowledge acquisition
2. recognition of external environment
3. learning and decision making

However, these fields have mutual dependence. Here knowledge means data or procedures that are given as truth for an objective and also includes the law of physics and mathematics. The knowledge acquisition includes how to represent knowledge and how to store knowledge. The recognition of external environment is to know or identify from the past experience or knowledge on the basis of the input and output from the environment. Learning is the process to acquire knowledge or skill based on the information acquired from external environment, and especially, the learning is to acquire inference rules as knowledge [18]. One of knowledge-based inference systems in the artificial intelligence is an expert system. The expert system is generally defined as a program that uses available information (knowledge), heuristics, and inference to propose solutions to a special problem.

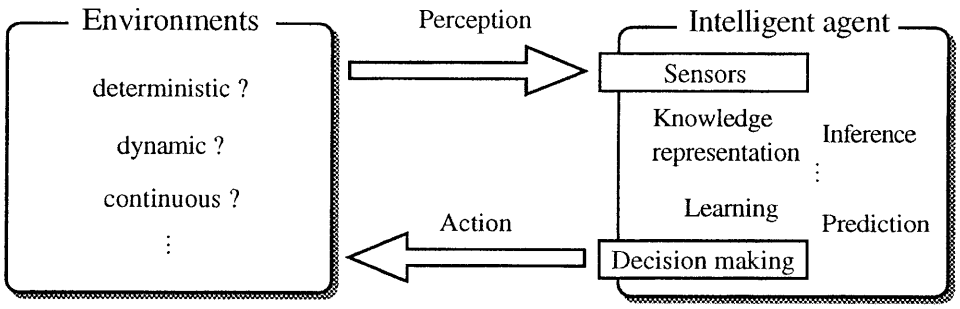


Figure 1.9 Intelligent agent interacting with various environments

As a next stage, the artificial intelligence requires an ability for problem-solving as the expert system. Here a problem is, in general, solved by using the acquired knowledge, but it is known that the solution by the acquired knowledge requires much time. Therefore, the solution methods by heuristics, have been proposed for reducing computation time. Here the *heuristics* is defined as useful knowledge acquired by experience in most case. The optimization is required for obtaining optimal knowledge and skill. In fact, the optimization methods have been applied for acquiring the knowledge such as inference rules and feature extraction.

1.6 Soft Computing and Optimization

Soft computing, which was proposed by L.Zadeh, is a new concept for information processing and its objective is to realize a new approach for analyzing and creating flexible information processing of human such as sensing, understanding, learning, recognizing, and thinking. In fact, L.Zadeh described,

> Soft computing is an emerging approach to computing which parallels the remarkable ability of the human mind to reason and learn in an environment of uncertainty and imprecision [22].

As methodological approaches of the soft computing, there are neural network [19~22], fuzzy systems [19,22,23], evolutionary computation, machine learning [18], conventional artificial intelligence and so on.

1.6.1 Neural Network

The human brain processes information super-quickly and super-accurately as a network. McCulloch and Pitts proposed that a suitably defined network could learn in 1943, and Hebb demonstrated a simple updating rule for modifying the connection strength between neurons of network in 1949 (Hebbian learning). In addition, Rosenblatt proposed perceptron which was a pattern classification system recognizing abstract and geometric patterns in the late 1950s. After that, the rediscovery of back-propagation algorithm by Rumelhart, popularized artificial neural network, though the back-propagation algorithm was developed by Werbos [8].

Artificial neural network simulating the biological brain can be trained to recognize patterns and to identify incomplete patterns. These training and learning features make neural networks suitable for applications in pattern classification, signal processing, control, and so on [19]. The basic attributes of neural network are the architecture and the functional properties; *neurodynamics*. Neural network is composed of many interconnected neurons with input, output, synaptic strength, and activation. The neural networks can be divided into two types; feed-forward and recurrent networks. A feed-forward network has input layer, hidden layer, output layer and unidirectional links between neurons (Figure 1.10), while the recurrent network has feedback links between neurons (Figure 1.11).

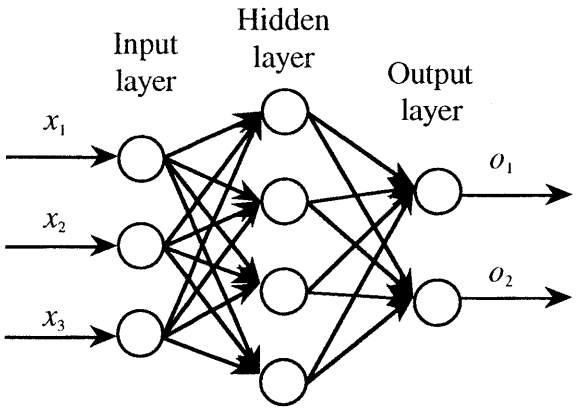


Figure 1.10 A feed-forward neural network

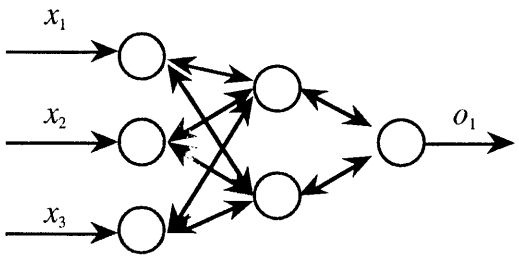


Figure 1.11 A recurrent neural network

The learning algorithm is, in general, determined by the teacher type from the environment. The learning algorithms for adjusting weights of synaptic strength are classified into supervised learning with target responses, unsupervised learning without target responses, reinforced learning only with the response of success or failure. In general, a multi-layer neural network is trained by a back propagation algorithm based on the error function between the output response and the target response. However, the back propagation algorithm, which is known as a gradient method, often misleads to local minima. In addition, the learning ability of the neural network depends on the structure of the neural network and initial weights of the synaptic strength. Therefore, the optimization of the structure and the synaptic strength is very important for obtaining the desired target response.

Other artificial neural networks are Hopfield network and Boltzmann Machine. Hopfield network is regarded as an autoassociative fully connected network which has symmetrically weighted links. Boltzmann Machine is based on the simulated annealing according to Metropolis dynamics.

1.6.2 Fuzzy System

The conventional expert system in a process controller has a problem of how to represent human decision making. Fuzzy theory and fuzzy logic, which was proposed by L.Zadeh, provide us the linguistic representation such as *'slow'* and *'fast'*(Figure 1.12). Fuzziness is often confused with probability. A statement is probabilistic if it expresses a likelihood or degree of certainty or if it is the outcome of clearly defined but randomly occurring events. On the other hand, the fuzzy expresses a degree of truth, which is represented as a grade of a membership function. The fuzzy logic is a powerful tool for nonstatistic and ill-defined structure.

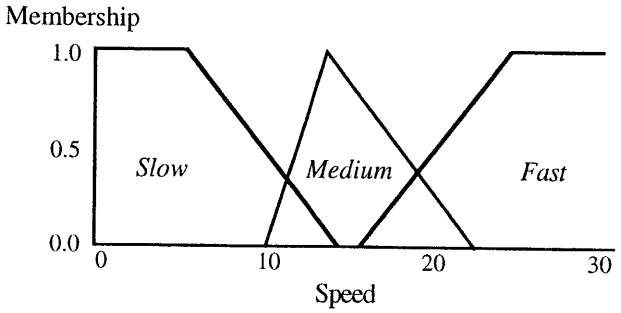


Figure 1.12 Membership function for speed

Fuzzy inference system is based on the concept of fuzzy set theory, fuzzy if-then rule, and fuzzy reasoning. The fuzzy reasoning derives conclusions from a set of fuzzy if-then rules. Fuzzy inference system implements mapping from its input space to output space by a number of fuzzy if-then rules. The widely used fuzzy inference systems are Mamdani fuzzy models and Takagi-Sugeno fuzzy models, which are used as fuzzy controllers [22]. The feature of the fuzzy controller is the locality of control and the interpolation among local control laws. In the fuzzy controller, the state space of the system is divided into some regions as membership functions which are antecedent, and the output (consequence) for the system control is designed as singletons or membership functions. Next, the fuzzy rules are interpolated as a global controller.

However, fuzzy theory and fuzzy logic have no tuning methods for fuzzy rules and human experts have generated and tuned the membership functions and fuzzy rules so far. Therefore, we require optimization and learning methods to obtain optimal fuzzy rules. The learning of

fuzzy systems can be basically classified into three categories; (1) structure learning which optimizes the combination of fuzzy rules, (2) antecedent part learning which optimizes the shapes of input membership function, (3) consequence part learning which optimizes the output of fuzzy if-then rules frequently defined as singletons. Of course, the combined optimization can be considered.

1.6.3 Emerging Synthesis on Soft Computing

Each technique plays the peculiar role in the soft computing and related fields (Table 1.2). However, there are not complete techniques for realizing all functions, and therefore, we can integrate and fuse some techniques to overcome the disadvantages of each techniques. One characteristics of neural networks are to recognize patterns and to classify input, and to adapt themselves to dynamic environments by learning, but the neural network is a black box. In addition, fuzzy systems can cope with human knowledge and perform inference, but fuzzy systems does not fundamentally have learning processes. Nuero-fuzzy computing has developed for overcoming each disadvantage [28~33]. In general, the neural network part is used for its learning and classifying, while the fuzzy logic part is used for inference and crisp output. Figure 1.13 shows the emerging synthesis of artificial neural network, fuzzy logic and genetic algorithm.

Table 1.2 Features of soft computing and related fields

Knowledge Representation	· Natural Language, Communication Language, Programming Language · If-Then Rule, Fuzzy If-Then Rule
Inference	· Production System, Fuzzy Inference System · Artificial Neural Network, Classifier System, Belief Network
Learning	· Back-Propagation Learning, Reinforcement Learning · Temporal Difference Method, Q-Learning, Bucket Brigade Algorithm
Search	· A* Search, Heuristics, Branch-And-Bound Method, Dynamic Programming · Line Search, Steepest Descent Method · Hill-Climbing Search, Genetic Algorithm, Simulated Annealing

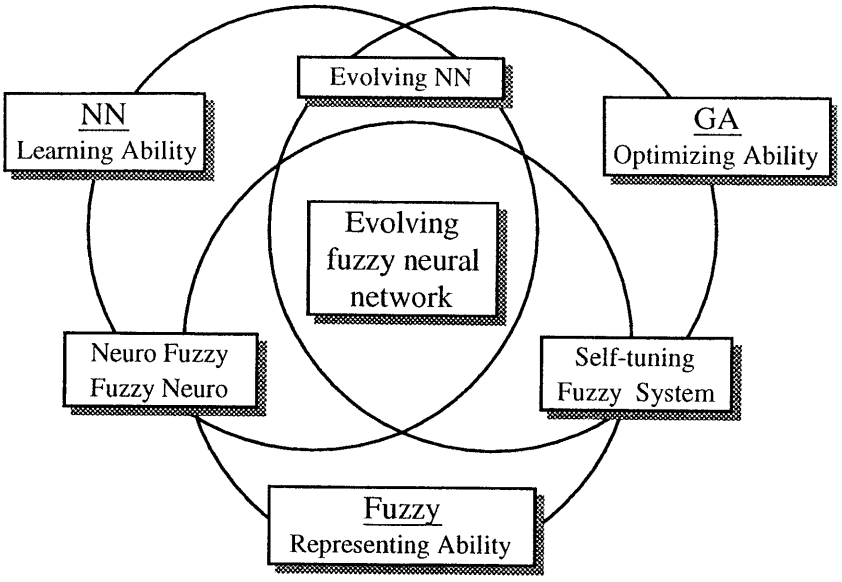


Figure 1.13 Emerging synthesis of neural network (NN), fuzzy and genetic algorithm (GA)

On the other hand, the evolutionary computation as mentioned before, plays the role of the optimization. The hybrid algorithms with evolutionary computation on the soft computing are known as evolving artificial neural network [25,26], self-tuning fuzzy system [27], evolving fuzzy neural network [34,35]. These hybrid algorithms can make up for each disadvantage by the advantages of other methods. In this way, the evolutionary computation provides the artificial neural network and fuzzy system with optimizing ability.

1.7 Artificial Life and Molecular Computing

Living things on earth can be divided into four levels from the structural point of view: (1) the molecular level, (2) the cellular level, (3) the organism level, and (4) population level. A living thing at any level is a complex adaptive system emerging from the interaction of a large size of elements from the below level. *Artificial life* means 'life made by humans rather than by nature.' The artificial life has three types of approaches: (1) wetware system from the molecular level, (2) software system from the cellular level and (3) hardware level from the organism level [36~39]. In the wetware system, there are a lot of attempts to artificial evolutionary process such as the molecular evolution in the tube. Cellular automaton, is known as one of software system, is placed in the grid on the cell space. All cells change each state according to the previous state of the cell and its close neighbors. The hardware system

is often described as an ecological system formulated by J.Holland [14,39]. The Holland's ecological system uses genetic algorithm and allow a large range of ecological interaction. The objective of the ecological system is to study how simple interactions among simple agents lead to emergent high-level phenomena.

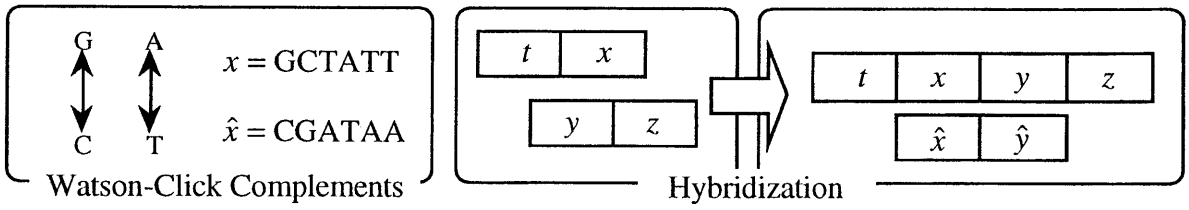


Figure 1.14 Representation by using DNA molecules

On the other hand, there are researches to solve a hard combinatorial optimization problem by building a DNA based computer. The DNA based computer encodes binary strings as DNA molecules composed of {A,T,G,C} (Figure 1.14). The combination of strings '*tx*' and '*yz*' is preformed by the hybridization based on the Watson-Click complements. By using biological operations such as polymerase chain reaction, the DNA molecules can be extracted and amplified for symbolic operations. These research are called *molecular computing* [40,41]. In fact, L.Adleman applied the DNA computer to a directed Hamiltonian path problem and R.Lipton applied the DNA computer to a satisfaction problem [40,41]. Though the DNA computers performs an operation very slowly (1 hour per operation, on the average), the DNA computer has a capability of huge parallel processing (10^{12}). These results indicates a probability that a DNA computer can realize far more computation than Turing computer. Therefor, the molecular computing is recently focused on as a computer science of the next generation. However, there are some problems such as the difficulty of the implementation, the difficulty to deal with errors in large scale systems.

1.8 Motivation and Goals

The optimization plays a very important role in the engineering, as presented in the above sections. The engineering requires the powerful and quick optimization method for complex problems to obtain the high performance. The quick solution of the problem enables the

reduction of the computation cost. And the powerful solution enables the improvement of the performance. Figure 1.15 shows the optimization techniques from the various points of view. The evolutionary computation and random methods are, in general, global and problem-independent searches since they does not use the features of the optimization problems, while the heuristics and numerical methods uses the features peculiar to the optimization problems.

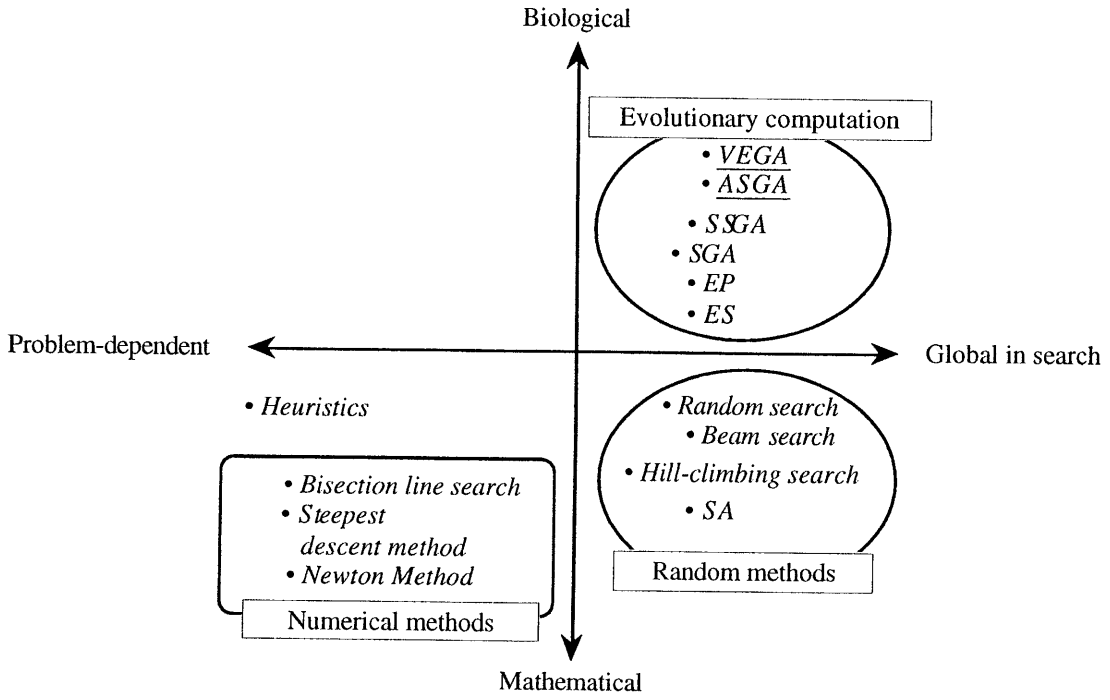
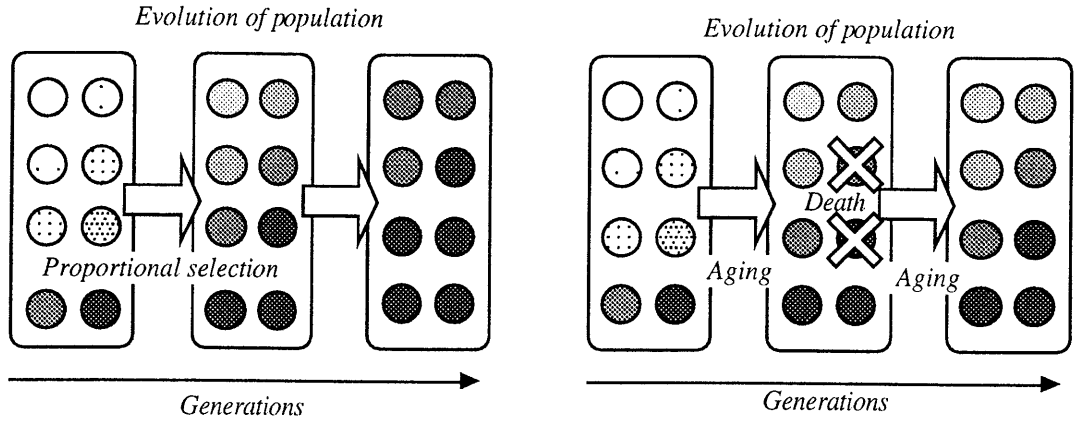


Figure 1.15 Optimization techniques from various points of view.
(*SGA*; standard genetic algorithm, *SSGA*; steady-state genetic algorithm, *EP*; evolutionary programming, *ES*; Evolution strategy, *ASGA*; age-structured genetic algorithm, *VEGA*; virus-evolutionary genetic algorithm, SA; simulated annealing)

On the other hand, nature often gives us effective ideas. In fact, human being have often simulated natural phenomena to create new technology, though the created technologies are sometimes different from natural phenomena. Evolutionary optimization methods simulate natural evolution from the viewpoint of neo-Darwinism, and work on a population of candidate solutions. The operations for the optimization are very simple and require neither the differential information nor the continuity concerning the objective function of a complex optimization

problem, but the evolutionary optimization methods can solve the complex optimization problem effectively. However, the evolutionary optimization methods have some problems in search. We therefore propose biologically inspired evolutionary optimization methods to improve the searching ability.



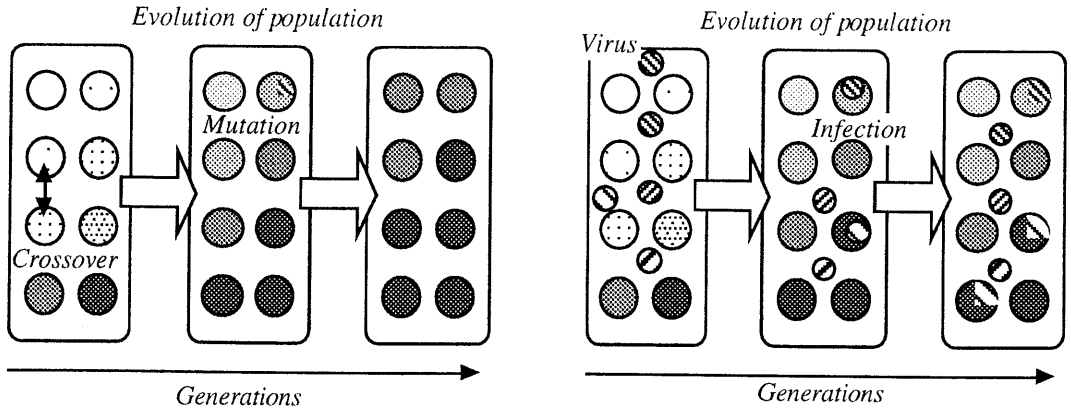
(a) Evolution by SGA (b) Evolution by ASGA

Figure 1.16 Difference of evolution of populations between SGA and ASGA

First, we propose an age-structured genetic algorithm (ASGA) as a simple extension from a genetic algorithm. The standard genetic algorithm (SGA) has a problem of premature local convergence which occurs when a population of individuals lacks genetic diversity, that is, the most of individuals are same one (Figure 1.16(a)). The ASGA maintains genetic diversity of a population by removing aged individuals from the population (Figure 1.16(b)). Furthermore, the ASGA is applied to a knapsack problem.

Second, we propose two evolutionary optimization methods based on virus theory of evolution; virus evolutionary genetic algorithm (VEGA) and virus evolutionary algorithm (VEA), into which virus infection operators are introduced. The SGA has no directionality in search and therefore the SGA is weak at local search, since the SGA uses crossover operators randomly recombining some individuals (Figure 1.17(a)). To improve this weakness, the VEGA uses local search by virus infection operators, while the VEGA basically uses genetic operators as global search (Figure 1.17(b)). Here virus individuals have local genetic information.

Furthermore, the proposed methods are applied to conventional and traditional optimization problems: (1) a knapsack problem, (2) a traveling salesman problem, and (3) a function optimization problem. The simulation results indicate that virus infection operators can effectively solve the optimization problems with the directionality in search.



(a) Evolution by SGA (b) Evolution by VEGA

Figure 1.17 Difference of evolution of populations between SGA and VEGA

Finally, the proposed evolutionary optimization methods are applied to engineering optimization problems: (1) trajectory planning problems of redundant manipulators as the application to robotics, (2) a pallet allocation problem in self-organizing manufacturing system as the application to intelligent manufacturing system, and (3) a self-tuning fuzzy controller for a cart-pole problem as the hybrid algorithm in soft computing. These simulation results show that the proposed methods can be successfully applied to complex engineering optimization problems.

1.9 Organization of this Dissertation

This dissertation is organized as follows. Figure 1.18 shows the outline of this dissertation. Chapter 2 presents evolutionary optimization methods in detail. To put it concretely, the chapter presents natural evolution, artificial evolution, genetic algorithm, evolutionary

algorithm, and advanced evolutionary optimization algorithms and their problems.

Chapter 3 proposes an age-structured genetic algorithm based on the population with age structure and its application to a knapsack problem.

Chapter 4 proposes two evolutionary optimization methods based on virus theory of evolution; virus evolutionary genetic algorithm and virus evolutionary algorithm. Furthermore, the chapter presents their applications to conventional and traditional optimization problems and some numerical simulation concerning virus infection.

Chapter 5 presents three application examples to engineering optimization problems. The first section proposes a hierarchical trajectory planning method for redundant manipulators and presents the application to some trajectory planning problems. The second section proposes a self-organizing manufacturing system and presents the application to a pallet allocation problem. The third section proposes a self-tuning method for a fuzzy controller and the application to a cart-pole problem.

Chapter 6 presents the conclusions and the future works.

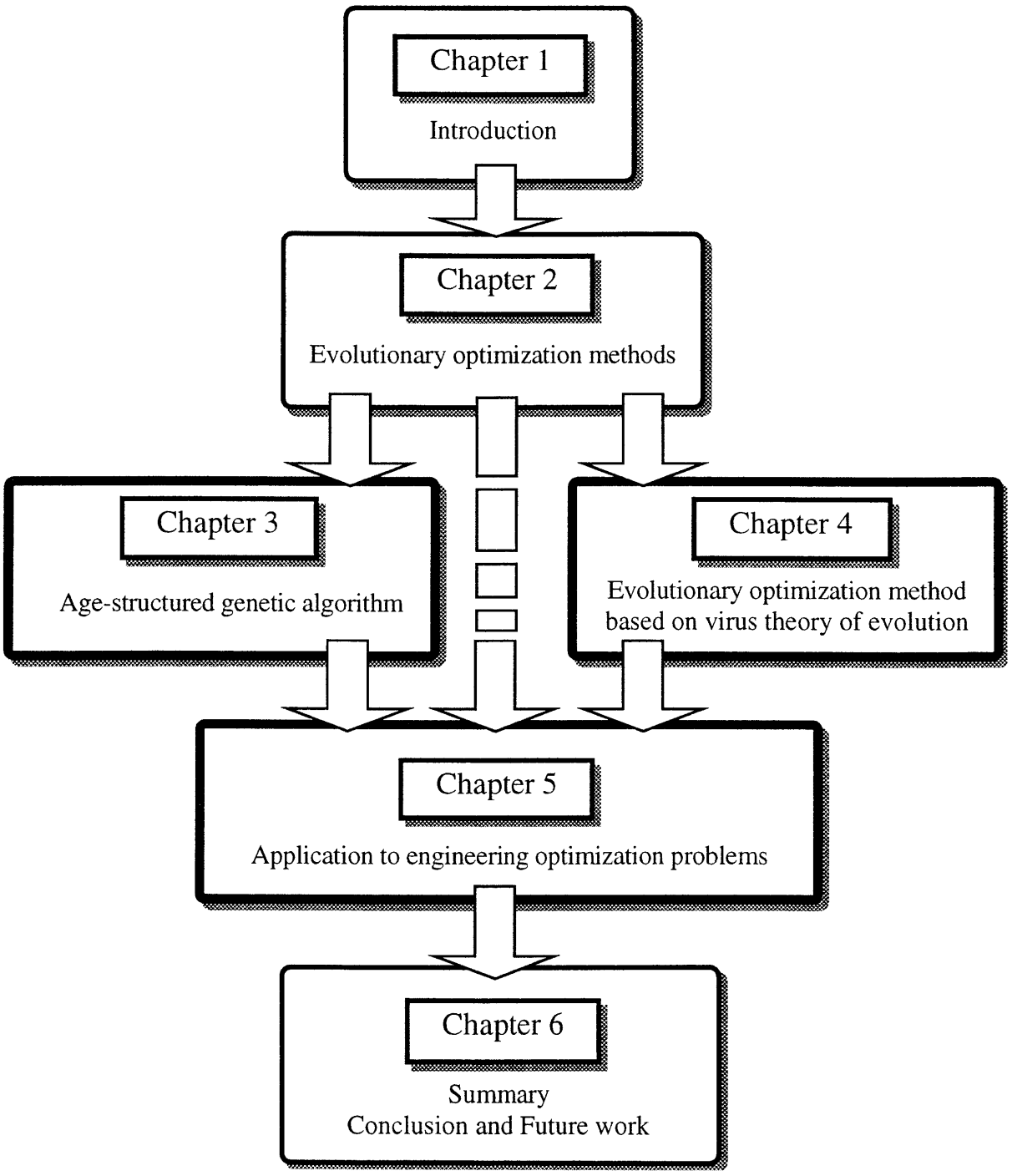


Figure 1.18 Outline of this dissertation

Chapter 2 Evolutionary Optimization Methods

Nature often gives us good ideas. Evolutionary computation is a field of simulating evolution on a computer. In the evolutionary computation, stochastic optimization methods simulate natural selection and work on a set of candidate solutions with operators based on evolution and genetics. Though evolutionary optimization method has very simple architecture, but is very powerful for optimization problems.

This chapter presents natural evolution and evolutionary optimization methods. Second, a simple genetic algorithm which is most basic and standard, and some extended genetic algorithms, are presented. Finally, other evolutionary algorithms: evolutionary programming and evolution strategy are presented.

2.1 Optimization Methods Based on Evolution and Genetics

Evolutionary optimization method is constructed based on the Darwinian theory of evolution. First, this section presents natural evolution of Darwinian theory and natural genetics. Second, this section presents artificial creatures evolving in an artificial world by simulating natural evolution.

2.1.1 Natural Evolution

What is *natural evolution*? The fact that evolution occurs in nature is certain. However, it is difficult to explain the process of evolution in nature. One of the most important evolutionary theory in biology is Darwinian theory of evolution [42]. The Darwinian theory is based on natural selection. Before Darwin, Lamarck argued that species changed over time into new species. Lamarck's conception of evolution has two mechanisms. One is "internal forces" to produce offspring slightly different from parents and this causes visible transformations over generations. The other is the inheritance of acquired characteristics. The inheritance is conventionally called "Lamarckian inheritance." With a progress of molecular biology, various theories of evolution have been proposed so far. There are other evolutionary theories [42~46] such as neutral theory of molecular evolution, Imanishi's theory of evolution, serial symbiosis

theory, and virus theory of evolution. First of all, the evolutionary process of the Darwin theory is presented.

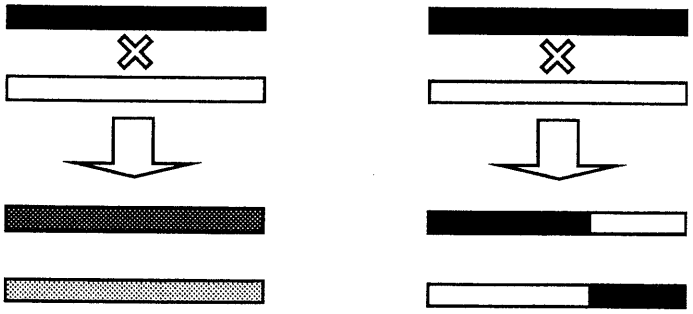
Darwin proposed the evolutionary theory in the book "On the Origin of Species" [44]. Evolution means change of living things (creatures) as the result of natural selection through generations. Darwin called evolution "descent with modification". Evolutionary modifications in creatures depend on their environmental changes and random genetic innovations. Evolution is mainly controlled under natural selection. Natural selection means the process that the creatures adapted to their environment, tend to survive and reproduce more offspring to the next generation. Natural selection requires the following conditions in nature,

1. Reproduction to form a new generation.
2. Heredity of features from parents.
3. Variation in characteristics of the members of the population.
4. Variation in the "fitness" of creatures associated with their various characteristics.

Fitness of a creature is defined as the relative contribution to the next generation, that is, when the fitness is higher, the creature can reproduce more offspring. However, Darwinian original theory is lack of heredity theory. In order to explain heredity theory, Neo-Darwinism, which was proposed as synthesis of Darwinian theory and Mendel's "atomistic" theory of heredity. Before Mendel, most theories of evolution were based on "blending" theory. In a blending mechanism, the "genes" are not preserved. The genes inherited from its parents are physically lost, as the two parental sets are blended together. On the other hand, in Mendelism, it is possible for the phenotypes of the parents to be blended in the offspring, but the genes do not blend (Figure 2.1). Therefore, the combinatorial changes of genes cause the changes of phenotype. Next, we present the mechanism of natural selection from viewpoints of molecular biology and population genetics [42,45,47].

In nature each individual has a certain number of chromosomes which are structured of DNA (deoxyribo nucleic acid). DNA carries genetic information used to build a new body and causes the inheritance of characteristics from parent to offspring. DNA consists of four types of bases, G (guanine), C (cytosine), T (thymine) and A (adenine). Polypeptide chain is made up of two chains of nucleotides twisted in a double helical structure and joined by hydrogen bonds between the complementary bases (Figure 2.2). Gene is a hereditary unit composed of DNA. Every gene is located at a particular place on a chromosome called genetic locus. A chromosome determines an individual characteristics based on the genetic composition of the

individual called a genotype. Therefore, a genotype determines its phenotype which expresses characteristics of the individual.



(a) Blending heredity (b) Mendelian heredity

Figure 2.1 The difference of Blending heredity and Mendelian heredity

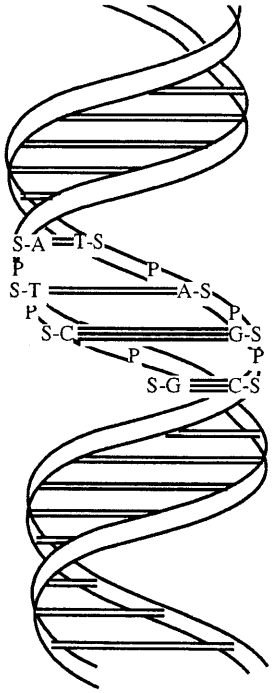


Figure 2.2 Structure of DNA: double helix model

The genotype of offspring is, in general, different from the parents because of a recombination between parents with different genotypes. Recombination is the rearrangement of genes that occurs when reproductive cells are formed. Figure 2.3 illustrates an example of recombination between two chromosomes. First, a pair of chromosome lines up, and second, a strands of the pair of chromosomes breaks at the same point, and last, the strands join together and recombine. A recombination generates offspring which have a combination of characteristics different from that of their parents.

When an individual reproduces, its genes are physically replicated. However, accidental errors naturally occur at a low rate during replication. The change by these errors is called mutation. There are, furthermore, some types of mutations such as inversion and translocation. The inversion reverses a sequence of genes on a chromosome. In the inversion, a strand of a chromosome breaks off and links in the reverse direction (Figure 2.4.a). In the translocation , a strand of a chromosome is broken off and rejoined to the chromosome at the different locus (Figure 2.4.b). The majority of mutations are, in general, harmful to individuals, but the mutation may increase an individual's fitness with a very small proportion. These successfully mutated individuals spread over the population by natural selection. Thus, genetic variations in species arise by these mutations.

To summarize simply, evolution is controlled under natural selection in Darwinism. However, species needs new genotype to adapt its external environmental change. Species can adapt itself to changing environment by recombination and mutation.

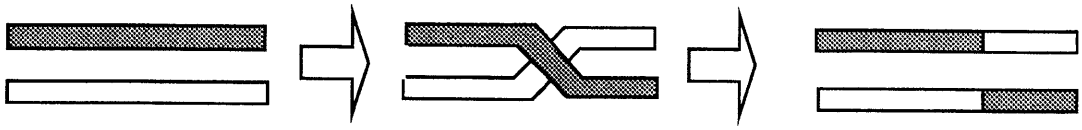
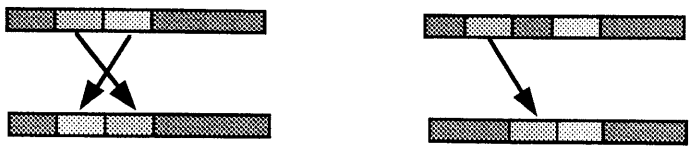


Figure 2.3 Recombination of chromosome



(a) Inversion (b) Translocation

Figure 2.4 Mutation (inversion and translocation)

2.1.2 Evolution in Artificial World

Living things evolve and adapt to their external environment, as mentioned in the previous section. If it is possible to simulate evolution on a computer, then we can realize an adaptive system like living things in nature. This section consider how to realize the evolution of artificial creatures in artificial world on a computer. As mentioned before, living things have some chromosomes which determine their characteristics by the genetic composition, and natural selection eliminates living things according to their adaptiveness or fitness to their environment. Darwinian theory of evolution provides a framework for realizing artificial evolution on a computer, and genetics provides a framework for dealing with symbolic operation on a computer. To realize an artificial creature and world on a computer, we assume the following condition:

1. An artificial creature has certain genetic information which is inherited from ancestor to offspring.

2. An artificial world has a population of the creatures, and selects the creatures adapted to their environment from the population into the next generation.

3. A fitter individual to its environment can reproduce more offspring.

Figure 2.5 shows the process of above artificial evolution. First of all, it is required to define environments in the artificial world to simulate the evolution of artificial creatures. If an optimization problem is regarded as an environment, then the environment can give each artificial creature (individual) a certain fitness. Each individual has a certain strategy or information to survive in the environment. In addition, individuals generate new individuals with recombination. As the result of recombination, individuals may create more suitable strategy or information generation by generation. Since the process of selection and recombination is repeated on a computer, more suitable individuals are created through generations. At last, we can obtain the most suitable individual, that is, a solution to the problem. This kind of method simulating evolution in nature, is called *evolutionary computation* [12~17]. There are three categories in evolutionary computation. J.Holland proposed *Genetic Algorithm* [14]. L.Fogel proposed *Evolutionary Programming* [13]. I.Rechenberg and H.Schwefel proposed *Evolution Strategy* [15,16]. These algorithms have historically independent origins, but the difference among them has become less distinct recently. Next, we consider the process of an evolutionary optimization method by using the genetic algorithm.

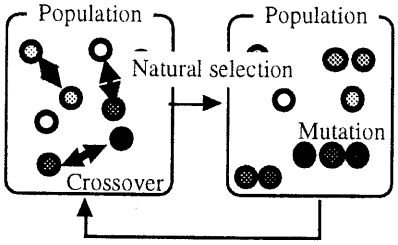


Figure 2.5 Evolution in artificial world

The initial genetic algorithm was devised as an adaptation process, and later, the genetic algorithms were used as optimization methods [12]. When we solve an optimization problem, the goal is often to obtain variables to maximize or minimize a given objective function. In the genetic algorithms, we must encode a decision variable into a finite length string on a chromosome as *genotype*, because genetic algorithms use genetic symbolic processes. The genetic algorithm works on a set of strings which called *population* with copying strings (i.e. *reproduction*) and exchanging of partial strings (i.e. *recombination*). *Individual* has one or some chromosomes of strings. In general, a string of an individual stands for a candidate solution by binary notation for the objective function. *Phenotype* of an individual represents a candidate solution and we can obtain phenotype with decoding a string of an individual. *Fitness* of an individual represents the value calculated an objective function. In the genetic algorithms, the operations for string are called genetic operators and the selection of candidate solutions to the next generation is performed by reproduction.

1. Recombination: this generates new individuals (offspring) by combining substrings on the string of parents.

2. Mutation: this replaces a gene on the string with the other one.

3. Reproduction: this creates the population of the next generation by selecting individuals according to their fitness values from the current population.

The selected fitter individuals generate new individuals with the recombination and the mutation. In this way, a population evolves toward optimal solutions. To summarize, the procedure for solving an optimization problem with a genetic algorithm is as follows,

Step 1. Determine a coding method from solution space into string space as genotype.
Step 2. Design fitness function.
Step 3. Design crossover operators and mutation operators suitable for the problem.
Step 4. Design selection scheme.
Step 5. Parameter tuning of crossover probability, mutation probability, selection pressure and so on.

2.2 Simple Genetic Algorithm

In this section, we present a simple genetic algorithm [12] and its application to a simple function optimization problem. Furthermore, we illustrate how a population evolves on a simple genetic algorithm by hand. A genetic algorithm mainly needs reproduction, recombination, and mutation. Though various genetic operators have been proposed so far, we first present a simple genetic algorithm because of the simplest and most standard.

A simple genetic algorithm is composed of roulette wheel selection (reproduction), one-point crossover (recombination), and simple mutation. Each operation is very simple and works on strings in a population only with simple bit operations. The procedure of the simple genetic algorithm is as follows:

```
begin
  Initialization
  repeat
    Roulette wheel selection
    Crossover
    Mutation
    Evaluation
  until Termination_condition = True
end.
```

With the genetic operators on a population by hand, we illustrate how to evolve on the simple genetic algorithm one generation. As a basic example, let us consider a function maximization problem in the following function:

$$f(x) = x^2 \tag{2.1}$$

where x is defined as unsigned integer.

First, we must represent each character in bit number (binary digit) to apply a simple genetic algorithm. We call this operation *coding* which is mapping a finite-length string space from a solution space. We can express the number from 0 to 31 with using 5 length bits in this case. The bit number 10110, for example, decodes to the number 22 as follows:

$$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 20 \tag{2.2}$$

In this way, we can decode the genotype into the phenotype.

We then illustrate the procedure of the simple genetic algorithm by hand. Initialization is to generate the initial population. Each individual's string in an initial population is generated with selecting character 0 or 1 at random. Assume the following population of four individuals at generation t:

No.	String
1:	10100
2:	01011
3:	11001
4:	01111

Next, we evaluate the fitness value of each individual against the environment. Calculating fitness function eq.(2.1), we can obtain the fitness value. The fitness value is defined as payoff or profit. In the simple genetic algorithm, the survival ability of an individual is dependent on the fitness value. The individual with a higher fitness value can reproduce more offspring under the selection. The reproduction generates a new population of the generation t+1 with copying strings of the generation t. The reproduction makes use of the selection of individuals according to their fitness values. In order to select offspring, the simple genetic algorithm uses the roulette wheel selection scheme as one of the most basic selection strategy. The roulette wheel selection selects an individual with the probability in proportion of its fitness value to the summation of all individual fitness value. The selection probability of an individual i, $p_{selection}$ is as following:

$$p_{selection} = \frac{fitness_i}{\sum_{j=1}^{n} fitness_j} \tag{2.3}$$

where n denotes a population size, which is 4 in this case.

A new population is generated with the roulette wheel selection four times. Table 2.1 shows a reproduction result of a new population in this case. Decoding a string into base 10 integer, we obtain the value of string and calculate fitness function. Calculating the selection probability of individual $p_{selection}$, we calculate the summation of all fitness values, which is a sum of 1371. We obtain each individual pselection according to the sum. The expected value is the number of individual to be selected into the next generation. In this case, based on each expected value, the roulette wheel selection selects the string with the highest fitness 625 twice, and the strings with the fitness 400 and 225 once, respectively. As the result of the selection, we obtain a new population of the next generation. The reproduction generates the population with higher fitness on average, not to generate individuals with new genotypes. Therefore, in order to generate new candidate solutions for solving the problem, genetic algorithms require recombination operators and mutation operators.

Table 2.1 Selection result by roulette wheel selection

Genotype	Phenotype	Fitness	*pselction*	Expected value	Selected
10100	20	400	0.33	1.16	1
01011	11	121	0.08	0.32	0
11001	25	625	0.43	1.72	2
01111	15	225	0.15	0.60	1

Second step is the crossover operator to generate new strings from two or more individuals. Mating between two individuals randomly selected, and next, choosing crossing site of them randomly, the crossover exchanges the partial substrings of strings cut by the crossing site between them. To divide population into couples mating at random shown in Table 2.2, we

choose each couple of crossing sites 2 and 4, respectively. The crossover is performed by the crossover probability $p_{crossover}$ = 1.0 here. Consequently, all individuals are recombined by the crossover. The third step is the mutation operator for reversing the bit of string at random according to mutation probability $p_{mutation}$ = 0.05 here. In Table 2.2, the mutation occurs at the one position, which bit '1' is replaced '0' in the locus 4 on the third string here. And then, we calculate the summation of all fitness value, which is 1818. The sum of all individuals becomes better than that of the previous generation by 1371 and the highest fitness value becomes better than the previous one by 49.

Table 2.2 One point crossover and mutation

Genotype	After crossover	After mutation	Value	Fitness
10|100	10001	10001	17	289
11|001	11100	11100	28	784
110|01	11011	11001	24	576
011|11	01101	01101	13	169

In this way, we illustrated three operations of the simple genetic algorithm by hand. As a result, we obtained a better solution after one generation, though we used only some probability, some bit operations and the roulette wheel selection. To summarize, only the simple symbolic operations without using characteristics of the problem enable the solution of the problem. All the population evolves toward a better solution set with increase of the average of fitness values generation by generation.

Why can genetic algorithms obtain a better solution? The reason is that the number of individual that the leftmost bit is '1', increases after one generation in this case. The leftmost bit is very significant, since the bit is the most influence to a fitness value. Because significant substrings increase in a population, all the population can evolve toward an optimal solution. In genetic algorithms, significant substrings increase in a population without considering themselves. With genetic operators, the significant substrings often generate new better significant substrings. This is called *building block hypothesis*. The evolution of a population,

not an individual, is an essence of genetic algorithm. The crossover operator is the most essential symbolic operation for generating better candidate solutions to search the solution space of the problem effectively. Further discussions of genetic algorithm in detail are presented in the following section 2.3.5.

2.3 Genetic Algorithm Architecture

Genetic algorithms are often called '*no prior knowledge required*' optimization techniques, since genetic algorithms can obtain optimal or quasi-optimal solutions with only simple symbolic operations without considering characteristics of objective functions for optimization problems. The simple genetic algorithm which was presented in the previous section, is shown to be powerful for solving an optimization problem, though its procedure is very simple. Simple genetic algorithm has the simplest architecture in genetic algorithms, but genetic algorithms should be extended suitable for an optimization problem to be solved. A number of genetic operators have been proposed in order to apply various optimization problems. We then present the fundamental characteristics of genetic algorithm again as follows:

1. Genetic algorithms work with strings coded the candidate solution.
2. Genetic algorithms work on a population of strings.
3. Genetic algorithms need only fitness value, not derivative information.

The reason why these genetic operators work effectively on a population, lies in the concept of building block hypothesis. But genetic algorithms can not solve all of optimization problems. These problems are called *GA-hard problems*, one of which is a minimal deceptive problem. In the problem, a crossover operator can not work well on a population. Furthermore, there is a problem of premature local convergence, which arises from the architecture of genetic algorithms.

In this section, we present coding methods from the solution space of an optimization problem into a string space, other genetic operators for applying various optimization problems, building block hypothesis, and some problems concerning genetic algorithms.

2.3.1 Coding

When we try to solve an optimization problem with genetic algorithms, first of all, we must design how to encode the solution space for the optimization problem into a string space. The coding means the mapping from the solution space into a finite-length string space. We require a good coding design in order to solve the optimization problem effectively. Various coding design methods have been proposed so far [48,77,79]. The most important condition in coding is to cover all of the solution space with the string space without redundancy, though living things in nature have some chromosomes which include a lot of redundant genes. Consequently, the phenotype of the string space should be equal to the solution space of the problem in order to solve the problem easily. In addition, the string space should be generated as a set of feasible candidate solutions in order not to perform the useless search.

Genetic operators, however, directly work on the genotype, not phenotype in genetic algorithms. Therefore, the performance of the genetic search depends on symbolic operations for the genotype. Genetic algorithms work best when substrings have consistent benefit throughout the string space. This is based on the concept that a coding method is deeply concerned in a crossover operator. The neighborhood of a candidate solution in the solution space should be similar to the neighborhood of the string correspondent to the candidate solution in the string space (Figure 2.6). If new offspring generated by a crossover were not similar to their parents, the population would proceed toward a different direction of the evolution, and the genetic search would result in failure. To the contrary, if genetic algorithms generate good offspring, the genetic search results in success. When we take into account the relation between coding methods and genetic operators, we offer two principles [12] of the coding rule as follows:

1. We should select the smallest characters for representing a solution space.
2. We should design a coding method for generating consistent strings by genetic operators.

The former rule means the coding design should be a non-redundant representation. We then consider the latter rule. As mentioned before, the building block hypothesis is very important. Genetic algorithms perform best when significant substrings increase in a population without the destroy of them by genetic operators. Though the genetic search is dependent on genetic operators, the genetic information to be inherited from parents to offspring is dependent on the coding design. Therefore, we should design a coding method so that genetic operators can generate meaningful offspring. The details of the principles will be presented in the following subsection.

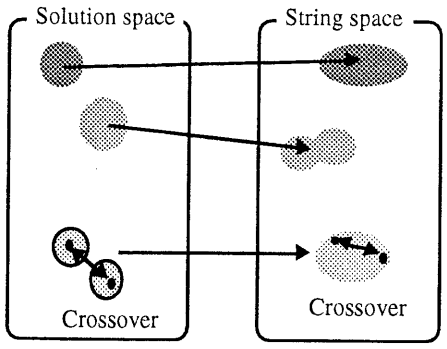


Figure 2.6 Coding solution space into string space

We then present some examples of coding techniques. First, we consider a binary coding in case of a function optimization problem. The coding is to design genotype corresponding to phenotype. When we assume the phenotype is integer, the integer x is represented by using the following equation:

$$x = \sum_{i=0}^{l-1} g_i \times 2^{i-1} \tag{2.4}$$

where l is the string length and g_i is the i-th gene. We can represent genotype very simply with the binary coding and it is easy to decode the genotype into the phenotype. As one of other bit representations, there is Gray coding. The Gray coding represents a adjacent integer as the binary string that the Hamming distance is 1 from a code to the next. There are some reports concerning comparisons between the binary coding and the Gray coding (Table 2.3). As the result of comparisons, the Gray coding is superior to the binary coding, because the Gray codes have the smaller perturbations by many single mutations than the binary coding. This shows that the performance of genetic algorithms is dependent on the similarity of strings in the string space.

Table 2.3 Representation of binary coding and Gray coding

Integer	Binary code	Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Next, we consider a permutation problem. In general, a permutation is not permitted overlapping of the same character on one string. In case of the permutation of the integer numbers from 1 to 5, the permutation is defined as follows:

1 2 3 4 5

In the permutation coding, we must design effective recombination operators. For example, we illustrate a one-point crossover operator used in the simple genetic algorithm (Table 2.4).

Table 2.4 One-point crossover in permutation problem

String	String after crossover
1 2 3 | 4 5	1 2 3 4 3
2 1 5 | 4 3	2 1 5 4 5

As the result of crossover, the characters "3" and "5" are overlapped in each string, respectively. These generated strings are out of the solution space, that is, infeasible candidate solutions. Consequently, genetic algorithms require good coding design and genetic operators in order not to generate meaningless strings.

2.3.2 Fitness Function

The goal of an optimization problem is often to find feasible solutions to maximize/minimize a given objective function. Most of genetic algorithms find feasible solutions to maximize a designed fitness function. The genetic algorithms basically require no more information than fitness value for an optimization problem. However, we can be free to introduce some ideas concerning the optimization problem into the fitness function. The reproduction operator based on natural selection is to reproduce more individuals with higher fitness values in order to obtain optimal solutions to maximize the fitness function.

We calculate the fitness value against its environment to generate a population of the next generation. Decoding the genotype into the phenotype, genetic algorithms calculate fitness function and obtain fitness value (Figure 2.7). A roulette wheel selection used in a simple genetic algorithm is to select an individual with probability in proportion of its fitness value to the summation of all individual fitness values. To the contrary, in the case of a minimization problem, we must transform the minimization problem into the maximization problem to apply a roulette wheel selection. We can transform a fitness function using the following equation:

$$fitness_i' = fitness_{max} - fitness_i + C \tag{2.5}$$

where $fitness_i$ denotes a fitness value of an individual i, $fitness_i'$ denotes a new fitness value, $fitness_{max}$ is the maximal fitness value in a population, and C denotes the offset for the range of fitness value. In this way, we can transform the fitness function suitable for genetic algorithms.

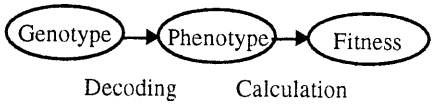


Figure 2.7 Calculation from genotype into fitness value

In the genetic algorithms, the convergence of a population is controlled under selection pressure [49~51]. The selection pressure is dependent on fitness values. If the selection pressure is high, the convergence speed is also high. If the selection pressure is too low, the selection randomly selects an individual without relation to its fitness value. *Fitness scaling* is used as a scheme for increasing or reducing the difference between fitness values. There are three fundamental schemes of fitness scaling: linear scaling, sigma truncation and exponential scaling. The procedure of linear scaling is as follows:

$$fitness_i' = A \times fitness_i + C \tag{2.6}$$

where A denotes a coefficient for increasing or reducing the difference. The procedure of sigma truncation is as follows:

$$fitness_i' = fitness_i - (fitness_{mean} - B \times \sigma) \tag{2.7}$$

where $fitness_{mean}$ denotes a mean average, B denotes constant, and σ denotes standard deviation. The procedure of exponential scaling is as follows:

$$fitness_i' = fitness_i^k \tag{2.8}$$

where k denotes a coefficient of the exponent. To control fitness values is actually the same to control selective pressure. With the control of fitness values, genetic algorithms can perform adequate selection to search the solution space.

2.3.3 Selection

The procedure of genetic algorithms starts with reproduction, recombination, and mutation. Selection simulates the process of natural selection, and genetic algorithms needs selection operations to make a population evolve toward the better direction of optimal solutions. The main process of a selection is to reproduce a next population with selecting an individual with the selection probability proportional to its fitness value. In this kind of stochastic selection, an individual with a higher fitness can reproduce more offspring. Because the selection is carried out by stochastic techniques, a number of the same individual may be selected by chance. As a result, some types of strings sometimes occupy a population. This is called

'*genetic drift*' [14] or '*random drift*' which often occurs in the case where the population size is relatively small in genetic algorithms. Various selection schemes have been proposed in order to prevent a population from genetic drift.

Selection schemes are classified into two main categories of proportional selection scheme and competition selection scheme. *Proportional selection scheme* is based on the fitness value of an individual against the total fitness value of all the population. On the other hand, *competition selection scheme* is based on the comparison with fitness values of some other individuals. We present some of the selection scheme as follows:

1. Roulette wheel selection scheme
2. Elitist selection scheme
3. Tournament selection scheme
4. Ranking selection scheme
5. Expected value selection scheme

Roulette wheel selection scheme [12], which presented in the previous section, is called Monte Carlo method in another name. In short, this scheme selects an individual with probability in proportion to its fitness values. This scheme is most basic and genetic algorithms use a hybrid scheme of this scheme and some others. The weak point of this scheme is that there is probability not to select an individual with a high fitness, or to select the individual many times, since this scheme uses only selection probability without any other ideas.

Elitist selection scheme [12] which is one of the competition selection scheme, preserves the fittest individual through all generation t, that is, the fittest individual is certainly selected to prior to others into the next generation. Elitist possesses rate is defined as percentage to the population size. This scheme prevents an individual with a higher fitness value from happening to be eliminated from the population. However, this causes the lack of genetic diversity in a population, because this scheme selects only the fittest individual. The most basic elitism concept was proposed by [14] as follows:

> Let $a^*(t)$ be the best individual generated up to time t. If, after generating $A(t+1)$ in the usual fashion $a^*(t)$ is not in $A(t+1)$, then include $a^*(t)$ to $A(t+1)$ as the $(N+1)$-th member.

In general, to preserve the fittest individual is very important for optimization methods to solve optimization problems, for we often want to obtain the best solution of the already

obtained solution candidates.

Tournament selection scheme [12,13,51] which is also one of the competition selection scheme, in general, selects an individual with the highest fitness value between m randomly selected individuals where m is the number of competing members. A genetic algorithm reproduces a new population with repeating this scheme n times where n is the population size.

Ranking selection scheme [12,50] is a scheme based on the rank sorted according to individual fitness values. This scheme selects individuals by the number of the its reproduction into the next generation based on the ranking table which is predefined.

Expected value selection scheme [12] is a scheme based on the expected value of individual fitness. First, we calculate the expected value from an individual selected the probability based on a roulette wheel selection scheme. Second, if an individual is selected, its expected value is decreased by 0.5. This scheme relatively prevents an individual from being selected more than twice in the population.

2.3.4 Genetic Operators

Genotype of an individual is changed by genetic operators such as genetic combination, transcription, inversion, and duplication, while evolution is controlled under natural selection in nature. Genetic algorithms require genetic operators for the purpose of evolution of all the population. The recombination generates new individuals with some crossover operators. The mutation generates new individuals with some perturbation operators. Each operator plays the role of the specialized functions, respectively. In this subsection, we present the crossover operators and mutation operators.

2.3.4.1 Crossover

Crossover operator generates new individuals as solution candidates in genetic algorithms [12,52~54]. Genetic algorithms can search the solution space by mainly using crossover operators. Without crossover, the genetic algorithm would be a random search algorithm. In addition, if the crossover did not work on a population, the genetic search would be like a random search or result in failure. The crossover operator replaces some or all of the population

with individuals of new genotype. Fundamentally accepted, the crossover operator exchanges each substring between two individuals. There have been various crossover operators peculiar to optimization problems [12,79,80]. We present some of crossover operators as follows:

1. One-point crossover
2. Multi-points crossover
3. Uniform crossover
4. Cycle crossover
5. Partially matched crossover; PMX

One-point crossover, as mentioned in the previous section, is one of the basic crossover operators. With choosing a break-point as a crossing site, the crossover recombines substrings between two individuals.

Multi-point crossover is a crossover operator with some break-points on a string. This operator recombines some substrings which cut by some break-points between individuals (Figure 2.8).

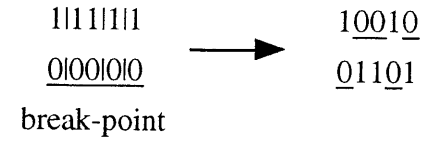


Figure 2.8 Multi-point crossover

Uniform crossover recombines two strings according to a string which called a mask pattern. First, we generate randomly a mask pattern including "0" or "1". We generate offspring with the exchange of characters between individuals at the locus of "1" of the mask pattern (Figure 2.9).

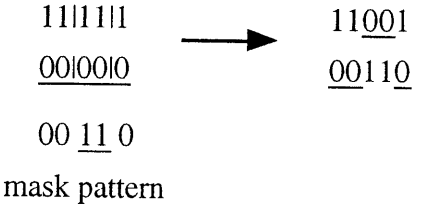


Figure 2.9 Uniform crossover

We often require crossover operators to satisfy constraints of permutation problems. These crossover operators prevent a character of an individual from overlapping on a string. For example, a travelling salesman problem is one of the combinatorial optimization problems and we present the traveling salesman problem in the following chapter 4. In order to solve the permutation problem we use crossover operators such as cycle crossover and partially matched crossover (PMX).

The cycle crossover [12], first, chooses a starting point, not a crossing site. Second, the cycle crossover makes a closed round of substrings between individuals. For example, we consider two parents as follows:

P_1 : 31245
P_2 : 24351

We illustrate the procedure of this crossover as follows. We start with choosing a starting point of the string and next choose the locus 2 of the P_1 in this case. A closed round of substring begins from this starting point. We copy the characters 1 and 4 in the locus 2 of the P_1, P_2 to the same locus of offspring, respectively.

O_1 : *1***
O_2 : *4***

Next, we copy the character of the locus existed the character 4 in the locus 2 of P_2 .

O_1 : *1*4*
O_2 : *4***

We repeat this process until we return to the first chosen character of P_1. Consequently, we can obtain a closed round of substring in a sense.

O_1 : *1*45
O_2 : *4*51

Finally, we fill characters from the former parents to the latter offspring. As a result, we obtain the complete offspring as follows:

O1 : 21345
O2 : 34251

The partially matched crossover; PMX [12] is a crossover operator consisted of two process. We illustrate the procedure of this crossover. First, we start with choosing randomly two break-points as crossing sites. In this case, we also consider two parents as follows:

P1 : 31|24|5
P2 : 24|35|1
break-points

We choose substrings between two break-points in both the parents and exchange substrings between the parents without overlapping of character.

O1 : **35**
O2 : **24**

The character 2 in the locus 3 of P1 is exchanged the character 3 in the same locus of P2, that is, we exchange the positions of the characters 2 and 3 in the P1, and the characters 3 and 2 in the P2. Furthermore, the characters on the substring rounded by two break-points are also operated by this exchange, respectively. Consequently, we can obtain the partially matching substrings as follows:

O1 : 2*354
O2 : 3524*

At last, the remained character is copied from P1 to O1, and from P2 to O2. As a result, we can obtain the complete offspring and as follows:

O1 : 21354
O2 : 35241

In this way, genetic algorithms can generate new individuals of feasible candidate solutions satisfying the constraint of permutation problems. There have been other crossover operators satisfying the constraint of the permutation problems [78,80]. As mentioned before, there is interdependent relation between coding and crossover. When we design the crossover operator,

the most important is to design crossover operators which can generate meaningful individuals, not meaningless individuals which are not feasible candidate solutions. Further, we must take into account the inheritance of genetic information from parents to offspring. After crossover operators, the offspring should inherit adequately genetic information from the parents. To summarize briefly, crossover operator is the most important for genetic algorithms to search the solution space effectively. And genetic search is dependent on the performance of crossover operators.

2.3.4.2 Mutation

Mutation occurs as the error in replicating in nature. In genetic algorithms, the mutation operator [12,55~57] replaces a randomly selected character on the string with the other one. The mutation is performed independent of individual fitness values. A standard mutation is one-point changing per individual. Mutation operator has several types in nature such as inversion, translocation and duplication. We present some of the mutation operators as follows:

1. Inversion
2. Translocation
3. Duplication

Inversion partially changes a character sequence from one direction to the opposite. The procedure of the inversion is as follows. First, we choose two points randomly and cut string at the points. Next, we link the substring in the reverse direction into the remained substring. For example, we assume a string included five characters. We choose two point of the locus 2 and 4.

1|234|5

With linking the substring, inversion results in the following state.

14325

Translocation (shift) changes the character sequence with moving to the different position. In translocation, we choose a substring randomly the same as the procedure of the inversion. We translocate the substring to a randomly chosen locus. For example, we assume the following

string, and the substring is chosen the segment from the locus 1 to 2. The translocating point is chosen as the locus 4.

|12|345

We insert the substring from the locus 4 to 5, and the remained substring of the string shift to the leftmost point of the string.

34512

Duplication overlaps the same substring on a string. In the duplication, we choose a substring randomly in the same way of the inversion. We overwrite the substring to a randomly chosen locus. For example, We assume the following string, and the substring is chosen the segment from the locus 1 to 2. The overlapping point is chosen as the locus 4.

|12|345

We overwrite the substring over the locus 4 and 5 and the remained substring is not operated as follows.

12312

However, this generated individual does not satisfy the constraint for permutation problems. Therefore, duplication is not suitable for permutation problems.

We then consider the role of the mutation operator. While the crossover operator works well on a population as the genetic search, the mutation operator second works on a population. However, the mutation operator is also an important one for genetic algorithms, since the mutation operator can reintroduce the characters eliminated from a population through the long generations. We assume that the behavior of a genetic algorithm is defined as Markov chain [13,58~62]. The Markov chain of the genetic algorithm composed of a selection scheme and a crossover operator is possible to transit to an absorbing state. The absorbing state means that a set of states can not escape from the own irreducible set. Figure 2.10 shows an example of an absorbing state. In the example, state transitions from the state A to B, from B to A, and from A to C are possible, but state transitions from C to any other states are impossible. The state C is an absorbing state in this example.

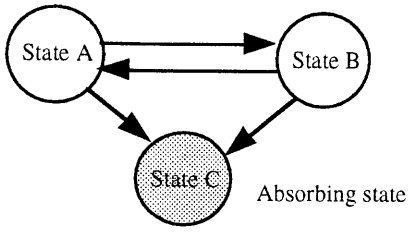


Figure 2.10 Absorbing state in Markov chain

Figure 2.11 shows an example of a population of an absorbing state in a genetic algorithm. The leftmost bit of all of the strings in the population are the character '1' in Figure 2.11. The individual that the leftmost bit is '0' can not be generated from the population by using the selection scheme and the crossover operator. However, the mutation operator can generate the individual that the leftmost bit is '0'. This shows that genetic algorithms require reintroducing operators like the mutation operator. Therefore, the mutation operator plays an important role for the genetic search.

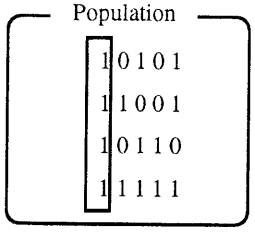


Figure 2.11 A population of absorbing state in genetic algorithms.

Thus thee mutation operator also influences a population. In conclusion, the mutation operator helps genetic algorithms to search a global solution space, since the mutation operator randomly changes the strings of individuals. If mutation probability is high, the mutation operator often happens to break an important substring on an individual easily. Therefore, genetic algorithms should, in general, use a low mutation probability. Genetic algorithms, in fact, use very low mutation probability such as 0.001 per character. There are many researches about the control of a mutation probability. For example, one of them is the dynamic control of a mutation

probability, in which the mutation probability is high at early generation, and decreases generation by generation. The dynamic control makes us enable the genetic search without considering the genetic bias of an initial population.

2.3.5 Building Block Hypothesis

The previous subsection presented genetic operators for various optimization problems. Why can genetic algorithm improve fitness value of a population throughout generations? As mentioned before, the reason is that significant substrings increase in a population without considering the genetic information of substrings. The significant substring is a segment for increasing fitness values. The substring is called a *schema* [12]. As schemata increase in a population generation by generation, all the population evolves toward the direction of optimal solutions. In fact, the crossover operator between parents with higher fitness often results in the generation of offspring with the higher fitness, that is, the schemata often generate new better schemata with genetic operators. This is called *building block hypothesis* [12]. We presented the role of genetic operators in the previous section, but we don't consider the performance of the genetic algorithms. We often treat schemata to analyze the behavior of the genetic algorithms.

We analyze how a schema increases in a population. First, we must represent string as the schema. Here we consider a problem with strings composed of characters {0, 1} over a binary string of length l. A schema H is defined as a string composed of three characters {0, 1,*}. Asterisk or star '*' refers to a 'don't care' or 'wild mark' which maybe matches 0 or 1. For example, we consider a schema H = 1*0*. This schema stands for four strings:

1000, 1001, 1100, 1101

Next, to analyze such a schema, we introduce schema order and defining length. The *order* of the schema H, which is denoted by $o(H)$, is the number of fixed position, that is, the order is equal to the number of 0 or 1 on the schema. The order of the schema H = 1*0* is 2, because the first and third characters are fixed. The *defining length* of the schema H, which is denoted by δ(H), is the distance between the first and last fixed positions. For example, the defining length of the schema H = 1*0* is 2 with the calculation of $\delta(H)$ = 3 - 1 = 2, because the first fixed position is 1 and the last fixed position is 3. In the case where the number of fixed position is 1, that is, $o(H)$=1, the defining length is 0, $\delta(H)$ =0, because the first and fixed

positions are the same.

As we defined schema order and defining length, we analyze how a schema increases in a population. As mentioned before, genetic algorithms are composed of reproduction, recombination, and mutation. An iteration performs these three genetic operations. We use a simple genetic algorithm as an example. We assume that the number of strings including schema H at generation t is denoted by m(H,t), where the population size is n and the string length is l. Then, we analyze how a schema increases in a population through genetic operators.

First, we consider reproduction. As the reproduction is roulette wheel selection scheme, we can obtain $m(H,t+1)$ after the selection scheme with calculating from $m(H,t)$. Here we assume that $m(H,t)$ is the expected value of strings. The schema increment equation of $m(H,t+1)$ after the selection scheme is as follows:

$$m(H,t+1) = m(H,t) \cdot \frac{f(H)}{\bar{f}} \tag{2.9}$$

$$\bar{f}(H) = \frac{1}{n}\sum_{i=1}^{n} f_i \tag{2.10}$$

where $\bar{f}(H)$ is the average fitness of the strings the schema H and f_i is the fitness value of the string i. This equation indicates that the schema increases in proportion of the average fitness of the schema against the average fitness of the population. In other words, the schema increases in a population when the average fitness of the schema is higher than the average fitness of the population. The reproduction can increase good schemata, but the reproduction can not generate new schemata. Since a genetic algorithm is a stochastic search method, the genetic algorithm requires recombination and mutation in order to generate new solution candidates.

Though crossover operators can generate new schemata, the crossover operator, however, has a possibility to break schemata at the same time. We illustrate how the crossover influences schemata.

$H_1 = * \ 1 \ 0 \ * \ * \ 1 \ *$

$H_2 = * \ * \ * \ 1 \ 0 \ * \ *$

We choose crossing point between the locus 3 and 4, for example. As the result of the crossover operator, the H_1 and H_2 becomes as follows.

$H_1 = * 1 0 | * * 1 *$

$H_2 = * * * | 1 0 * *$

When genetic algorithm performs crossover operator at the crossing point, the schema H_1 is broken up, but the H_2 survives without being broken up. From the result of this example, the schema will be broken up easily as the schema length is long. The probability P_s that the schema H survives after a crossover operator is as follows:

$$P_S = 1 - P_c \cdot \frac{\delta(H)}{l-1} \tag{2.11}$$

where P_c, l and $\delta(H)$ denote crossover probability, the string length and the defining length of a schema H, respectively. However, the survival probability is, in fact, higher than P_c, because the same schema may be generated by the crossover operator. Then, we consider the survival probability of a schema after mutation operator. The mutation operator changes a character with the mutation probability P_m. If a mutation occurs on the string without fixed characters of the schema, the schema can survive after the mutation operator. Therefore, the survival probability after a mutation operator is as follows:

$$P_S = (1 - P_m)^{O(H)} \tag{2.12}$$

where $O(H)$ is the order of the schema, that is the number of the fixed characters of the schema. As the value of the probability P_m is very low relatively, we may approximate the P_s of eq.(2.12) as follows:

$$P_S \approx 1 - P_m \cdot O(H) \tag{2.13}$$

Therefore, we can obtain the schema increment equation of $m(H,t+1)$ after selection, crossover operator, and mutation operator is as follows:

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \cdot \left[1 - P_c \cdot \frac{\delta(H)}{l-1} - P_m \cdot O(H)\right] \qquad (2.14)$$

From eq.(2.14), the number of a schema increases when the schema is a short length and low order. This conclusion is very important and is called *schema theorem* or *fundamental theorem of genetic algorithms* [12]. Since some schemata generate new better schemata with the crossover operator and the number of a schema increases in a population with the selection operation, this is called building block hypothesis. Building up bits of blocks called schemata, genetic algorithms can obtain optimal solutions. In addition, we should use low crossover probability and low mutation probability from the conclusion of the eq.(2.14). However, the schema theorem does not consider new individuals generated by crossover operator and mutation operator, while a schema increases in a population. Consequently, the schema theorem does not indicate whether or not genetic algorithm can obtain optimal solutions. Whether or not genetic algorithm can obtain optimal solutions with finite time is an open problem. In addition, genetic algorithms can not generate individuals of new strings if crossover probability is low. Therefore, the crossover probability should be relatively high in order that the genetic algorithms can search the solution space effectively.

2.3.6 Problems of Genetic Algorithm

We presented building block hypothesis in the previous subsection. In this subsection, we present problems in genetic algorithms as follows:

1. Premature local convergence
2. Minimal deceptive problem

The premature local convergence is a phenomenon resulted from stochastic bias by a proportional selection scheme. The proportional selection scheme selects individuals with higher fitness priory, and as a result the population lacks genetic diversity. On the other hand, the minimal deceptive problem is dependent on the solution space, in which the problem is that genetic operators are not effective for searching the solution space.

2.3.6.1 Premature Convergence

Selection operation simulates the process of natural selection and select an individual with the probability proportional to its fitness value. In this kind of stochastic selection scheme, an individual with a higher fitness value can reproduce more offspring. In general, selection operation should realize two different aims. The first one is to select individuals with high fitness values in order to select good candidate solutions. The other is to maintain genetic diversity in population in order to generate new offspring with new genotype. However, there is a trade-off between these aims. Genetic algorithms can control the trade-off with selection pressure. The convergence of a population is controlled under the selection pressure which is dependent on fitness values [12]. If the selection pressure is high, the convergence of a population proceeds fast. If the selection pressure is very low, the selection randomly selects an individual without relation to its fitness value. The proportional selection scheme often causes the genetic bias in a population, because the scheme is performed stochastically. As mentioned before, this is called 'genetic drift' or 'random drift'.

Premature local convergence which is one of genetic drifts, occurs when a population lacks genetic diversity in early generation. The phenomenon of premature local convergence is that a certain individual with a high fitness value occupies the population, though the individual is far from the optimal solution. As a result, premature local convergence may mislead the evolution of the population toward the different direction of the optimal solution. Genetic algorithms have no convergence guarantees in arbitrary problems, but there is a convergence theory proved by using a simulated annealing [68] and Markov chain analysis. We will present this theorem in the following section 2.4.2.1.

Genetic algorithms should maintain genetic diversity within a population in order to prevent a premature local convergence in solving an optimization problem. Because the standard proportional selection such as roulette wheel selection, is dependent on fitness values of individuals, the convergence of a population can be controlled with varying selection pressure. For example, the selection pressure is lower in the early generation, and higher in the late generation. By varying selection pressure, the global search and local search can be switched.

Other methods for preventing premature local convergence are fitness scaling, ranking selection scheme, and so on. The fitness scaling is used as a method for increasing or reducing the difference between fitness values. As mentioned before, there are fundamental methods of fitness scaling such as linear scaling, sigma truncation and exponential scaling. The ranking

selection scheme is based on a rank table sorted according to individual fitness. This selection scheme selects individuals by the number of the reproduction into the next generation based on the ranking table.

2.3.6.2 Minimal Deceptive Problem

In the schema theorem, the shorter length and the lower order a schema has, the more the schema effectively increases in a population. In addition, in the building block hypothesis, genetic algorithms can obtain optimal solutions with genetic operators for building up bits of blocks. There are, however, optimization problems impossible to generate good offspring from good parents with crossover operator. This kind of problem is called a *deceptive problem.* A deceptive problem has been defined based on the schema analysis by Goldberg, which stated that even the deceptive problems are often solved effectively by genetic algorithms.

We consider a simple example of a minimal deceptive problem. Figure 2.12 shows the case of two bit's combination problem. Table 2.5 shows the results of crossover operator between all individuals. We assume the fitness values of strings '00', '01', '10' and '11' are 5, 3, 2 and 6, respectively. 'S' in Table 2.5 means stable where the offspring generated after crossover are the same as the parents.

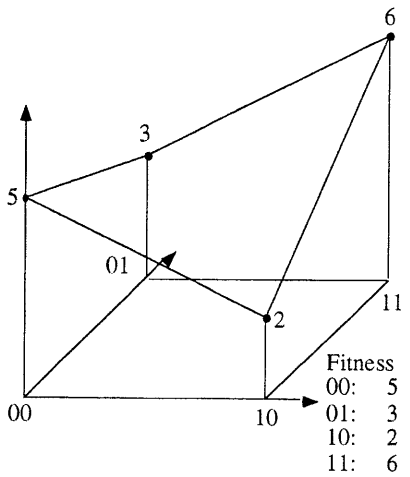


Figure 2.12 An example of minimal deceptive problem

Table 2.5 Result of all crossover

String	00	01	10	11
00	S	S	S	01,10
01	S	S	00,11	S
10	S	00,11	S	S
11	01,10	S	S	S

In this example, the single crossover generates only bad individuals: '01' and '10' from the good individuals: '00' and '11'. This shows that the crossover operator works ineffectively on a population.

In this way, the minimal deceptive problem can not adopt the schema theorem. In order to solve an optimization problem effectively with genetic algorithms, the solution space of the optimization problem should satisfy the schema theorem, that is, the string space including schemata which have a shorter length and a lower order, should be designed to point the way towards optimal solutions.

2.4 Advanced Operators in Genetic Algorithm

Genetic algorithms have been applied to various optimization problems by introducing advanced genetic and biological ideas and operators. This section presents advanced architectures and genetic operators based on genetics, biology and ecology. Furthermore, this section presents improving methods for genetic algorithms from the mathematical point of view.

2.4.1 Parallel Genetic Algorithm

Parallelism of genetic algorithm has been proposed by many researchers [63~68]. Parallelism can be mainly divided into two types. The first one is to divide a population into some subpopulations, and genetic operators in a subpopulation prevent local optima from widely

propagating to other subpopulations. The other is to realize the fast computation by parallel computers. We can realize both approaches simultaneously. In this subsection, we present two implementations of genetic algorithm on parallel architectures of a grid model and a subpopulation model.

2.4.1.1 Fine-Grained Parallel Genetic Algorithm

In a fine-grained parallel genetic algorithm [67], individuals in a population are placed on a planar grid. Genetic operations such as selection and crossover are restricted to the neighborhoods on that grid (Figure 2.13). Individuals of the next generation on a particular location are selected from the individual in the neighborhood of its location. An individual is replaced with the selected individual of its neighborhood. Crossover is performed between an individual mated from its neighborhood. The procedure of the fine-grained parallel genetic algorithm is as follows:

```
begin
  Initialization
  repeat
    Selection by replacement
    Crossover
    Mutation
    Evaluation
  until Termination_condition = True
end.
```

The main deference from a standard genetic algorithm is the selection operation by replacement. Fine-grained parallel genetic algorithm replaces each individual with an individual selected from its neighborhood on the grid, if the selected individual has the higher fitness value. Next, crossover operator is performed between individuals randomly selected mate from its neighborhood with a crossover probability. Mutation operator is the same operator as the standard genetic algorithm.

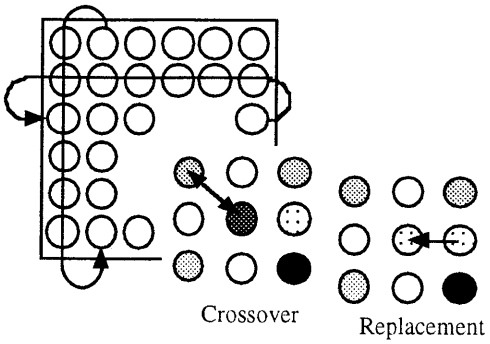


Figure 2.13 Fine-grained parallel genetic algorithm

The concept of the fine-grained parallel genetic algorithm is based on the ecological and technical reasons. The technical reason is easy to construct a parallel genetic algorithm because of the consideration of only neighborhood, not a global population. The ecological reason is no global selection in a population. In nature, natural selection is a local phenomenon, which is taken place in the local environment of an individual.

2.4.1.2 Parallel Genetic Algorithm with Subpopulations

Parallel genetic algorithm divides a population with some subpopulations (Figure 2.14). The genetic operations are carried out in the subpopulations independently, and each subpopulation try to locate good local optima [64~66]. New offspring are generated by genetic operators within each subpopulation. Therefore, each subpopulation evolves toward each different direction like a hill-climbing algorithm. After some generations, the locally best solutions found by a subpopulation are propagated to the neighboring subpopulations, which is called *migration.* Every K generations, the best individual of each subpopulation is sent to its neighbors as the migration. The migration frequency is very important. If the migration is often done, the subpopulations are the same as one population. If any subpopulations does not perform migration for a number of generations, individuals in the subpopulation will try only local hill-climbing. The division into some subpopulation prevents premature local convergence from occurring in a population. The genetic algorithms with subpopulations often are applied on parallel computers with multi-processor, since each subpopulation can be assigned to each processor easily.

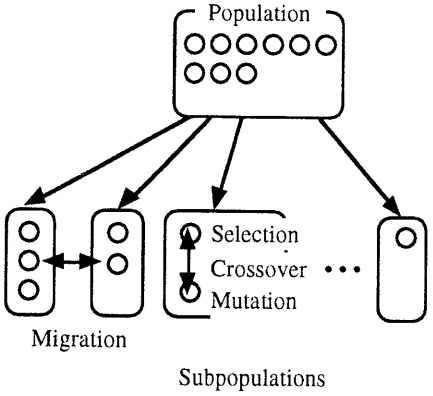


Figure 2.14 Parallel genetic algorithm with subpopulations

2.4.2 Hybrid Genetic Algorithm

In recently, various optimization methods have been proposed by many researchers [12,68~70]. Each method has both of effectiveness and weakness. Complete methods for solving any optimization problems have not been proposed so far. In general, there are a global search method and a local search method in optimization techniques. Local search methods search for optima based on the specialized information peculiar to an optimization problem. The more the specialized information is, the higher performance the local search methods can obtain. However, it is very difficult to apply the specialized local search methods to other optimization problems, since the local search methods lack the generality for any other optimization problems.

When we solve optimization problems, it is very important to obtain high performance. As mentioned before, genetic algorithms have a weakness in the local search. Genetic algorithms can obtain optimal or quasi-optimal solutions, but can not efficiently search the neighborhood of an candidate solutions. Since the genetic algorithm is a global search method, it is effective to hybridize genetic algorithm with other local search methods.

Hybrid genetic algorithm incorporates the available domain-knowledge of local search methods into genetic operators. There are traditional hill-climbing methods such as Quasi-Newton methods [8] and random optimization methods [7,10] in local search methods. In this subsection, we present a genetic algorithm with a simulated annealing algorithm and genetic algorithm with a hill-climbing method.

2.4.2.1 Parallel Recombinative Simulated Annealing

Genetic algorithm [12] and simulated annealing algorithm [11] are known as stochastic optimization methods simulated process in nature. Both algorithms can be applied to a wide variety of problems, and require little knowledge of optimization problems. Though both algorithms have possibility to prematurely converge to local optima, the simulated annealing algorithm possesses a formal proof of convergence by manipulating the cooling scheduling. Genetic algorithm is a parallel algorithm with a population referred to as implicit parallelism. In short, implicit parallelism refers to the fact that genetic algorithms implicitly process schemata although genetic algorithms explicitly process strings. On the other hand, the parallel algorithm of simulated annealing is an explicit parallelism.

Parallel recombinative simulated annealing [68] retains the asymptotic convergence properties of simulated annealing, and adding the population approach and recombinative power of genetic algorithm. Parallel recombinative simulated annealing closely follows simulated annealing with using population in parallel, mutation as a neighborhood operator, and crossover for recombining independent solutions. The procedure of parallel recombinative simulated annealing is shown as follows:

```
begin
  Initialization
  repeat
    repeat: Replacement under cooling schedule
      Pairing
      Crossover
      Mutation
      Keep parents with probability, 1/(1+e^(Δf/bT))
    end
  until Termination_condition = True
end.
```

First we start with a randomly generated initial population. In the main loop, we generate new offspring through neighborhood operator, and replace parents with offspring according to the Metropolis or equivalent criterion. Metropolis algorithm uses the probability $e^{\Delta f / T}$ where Δf is the difference between the fitness values of the parents and offspring. Another frequently

used criterion is the Boltzmann probability distribution, $1/(1+e^{\Delta f/T})$.

The proof of the convergence is shown by the following simulated annealing algorithm. We prove global convergence by reducing the Metropolis algorithm. First, concatenate of all strings of parallel recombinative simulated annealing. This string is considered as one string and called '*superstring*'. Consider that we optimize the superstring with crossover-and-mutation as a neighborhood operator. And fitness value of superstring is defined as sum of each string's fitness value. As such a special case, this algorithm inherits both of the convergence guarantees and behavior of simulated annealing, provided the neighborhood operator meets two conditions. The first condition requires that it can move from any state to an optimal solution in a finite number of transition. This requirement is satisfied by the mutation operator. The second condition requires that the transition probability at any temperature from state i to j is as same as the probability from state j to i. This requirement is satisfied by crossover-plus-mutation operator.

2.4.2.2 Genetic Algorithm with Hill Climbing

Simple genetic algorithm has no operators for improving performance by local changing on strings. To improve the weakness of local search, it is required that genetic algorithms are implemented genetic operators like hill-climbing methods [69,70].

Bit-climbing operator is the simplest method. Hill-climbing start with a string composed of bits. When the new string flipped bits is better than the fitness value of the old one, replace the old one with the new one. The bit-flipping continues until no improvement in trials or until the predefined trial times.

Genetic algorithms are often applied to numerical optimization problems. When applying to these problems, we use coding design with real number. In this case, we can hybridize genetic algorithm with hill-climbing operators such as *simplex crossover* [70]. Let n be the number of variables of the function to be optimized. A simplex is a geometrical figure consisting of $(n+1)$ vertices in an n-dimensional space. The simplex moves, contracts, and expands through geometric transformations, and finally surrounds the optimum. The elementally transformation is determined by the relative order between the values of the function. After each transformation, the worst point is replaced with a better one.

2.4.3 Steady-State Genetic Algorithm

There are currently two reproductive techniques in general use. The first one is a generational reproduction model, which is probably the most widely used, and the other is a steady-state reproduction model [71]. In short, the generational reproduction replaces all the population at once, while the steady-state reproduction replaces only a few individuals at once. According to schema theorem, the performance of genetic algorithm is highly dependent on its reproductive operation. The generational reproduction replaces all the population with a new population. This is done by selecting an individual from the current population according to fitness value, and adding the individual to the next population, repeatedly.

The steady-state reproduction replaces only a few individuals during a generation. And next, crossover and mutation is also performed against only a few individuals. These generated individuals are immediately available for generating and selecting next individuals, while the generational reproduction model can not immediately use the generated individuals. In general, an individual is replaced with one generated by genetic operators in each generation. Therefore, we must have methods for selecting an individual to be deleted. In this case, Syswerda presents three alternative deletion methods as follows:

1. Delete least fitness: deletion of the individual with the least fitness in the population.
2. Exponential ranking: The worst individual has some probability, p of being deleted. If it is not selected, then the next to the last also has p chance, and so on.
3. Reverse fitness: Each individual has probability of being deleted according to fitness value.

2.5 Evolutionary Algorithms

As mentioned before, the three main categories of evolutionary optimization methods are genetic algorithm, evolution strategy, and evolutionary programming. The procedure of these algorithms are very similar. The main iteration loops of all evolutionary optimization methods are basically the same, and it starts with the selection/reproduction or recombination/mutation. The evolutionary algorithms, however, differ from genetic algorithms in the specific representation, mutation, and selection. In this section, we present the evolution strategy and the evolutionary programming briefly.

2.5.1 Evolution Strategy

Evolution strategies [15,16,72] are applied to solve continuous parameter optimization problems, for an evolution strategy represents an individual by means of integer or real variables, neither binary nor string. Therefore, the evolution strategies directly operate with phenotype, not genotype. The main evolutionary operator for searching is the mutation operator using a normal random value with zero mean, not crossover operators. The reason is that crossover operator is included in the mutation operator in evolution strategies, while a crossover operator is generally utilized as genetic operator for the global search in the genetic algorithms. However, crossover operators rarely are used in the evolutionary strategies. The evolutionary strategies are divided into two types of $(\lambda+\mu)$-evolutionary strategy and (λ, μ)-evolutionary strategy. First, we present $(\lambda+\mu)$-evolutionary strategy.

Reproduction in the evolution strategies is performed by a deterministic selection, though genetic algorithm uses stochastic selection schemes such as roulette wheel selection. The $(\lambda+\mu)$-Evolution strategy reproduces the population of the next generation from the current population which includes both parents and children created by the mutation (Figure 2.15). The procedure of evolution strategy is as follows:

```
begin
  Initialization
  repeat
    Creation (μ) with Mutation
    Evaluation
    Selection (λ)
  until Termination_condition = True
end.
```

First, initialization randomly generates an initial population. In the iteration loop, μ individuals are selected as parents from the population of λ individuals. And with creation (μ) with mutation, the parents create μ children. As a result, the evolution strategies create an intermediate population of $\lambda+\mu$ individuals. Finally, the population of the next generation is reproduced by

selecting best λ individual from the intermediate population. The iteration continues until the Termination_condition is fulfilled. The notation $(\lambda+\mu)$-evolutionary strategy refers to an evolution strategy with the selection scheme which selects the μ individuals from λ individuals to create new μ children .

(λ, μ)-evolutionary strategy is different from $(\lambda+\mu)$-evolutionary strategy in a selection scheme. (λ, μ)-evolutionary strategy also generates m individuals by mutation operator, but replace all of λ individuals with μ generated individuals $(\lambda \geq \mu)$.

From the viewpoint of stochastic search, we can state the correspondence between evolutionary strategies and other stochastic search methods.

- (1, 1)-evolutionary strategy: random search
- (1+1)-evolutionary strategy: hill-climbing (iterative) search
- $(\lambda+\lambda)$-evolutionary strategy : beam search

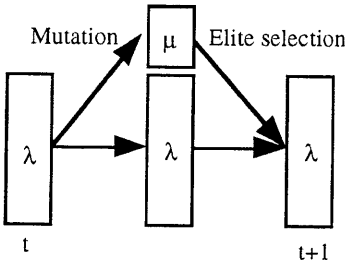


Figure 2.15 $(\lambda+\mu)$-evolutionary strategy

Evolution strategies optimize not only objective variables but also strategic parameters. This realizes a self-adaptation of the evolution strategies in itself. This is mainly used as self-adaptive mutation. Let x_i be a variable of an objective function for a numerical optimization problem.

$$x_{j+n,i} = x_{j,i} + N(0, a_i \times fitness + b) \tag{2.15}$$

where a_i and b are coefficients for scaling and offset in order for the variance of normal random variable not to be zero, respectively. This self-adaptive mutation adds small perturbation when fitness value is small.

2.5.2 Evolutionary Programming

The evolutionary programming originated from the idea of building a finite state machine for solving prediction tasks [13]. Nowadays it is mainly used for numerical optimization problems [13,73,74] such as the optimization of weight parameters of neural networks. We present an evolutionary programming for continuous optimization problems. The procedure of a standard evolutionary programming is as follows:

```
begin
  Initialization
  repeat
    Mutation
    Evaluation (simulation)
    Tournament_competition
  until Termination_condition = True
end.
```

First, initialization randomly generates an initial population. Evolutionary programming generates offspring from parents with a mutation perturbed by a normal random value with mean zero. A population of the next generation is reproduced by stochastic tournament selection (Figure 2.16).

We consider evolutionary programming in case of building a finite state machine to solve prediction tasks, that is, the evolutionary programming generates the procedure for solving a problem. Here we regard an individual in a population as a finite state machine. A finite state machine receives an input symbol from the environment and produce an output symbol that is likely to maximize the payoff with respect to the next symbol to emerge from the environment. After the prediction is made, the payoff for each symbol becomes a fitness of the machine.

Offspring is created by five mutations: change an output symbol, change a state transition, add a state, delete a state, or change the initial state. Mutations are chosen concerning a probability distribution, which is typically uniform. The number of mutations is also chosen concerning a probability distribution. A population of the next generation is reproduced by the tournament selection scheme. Fogel examined the use of the evolutionary programming for strategy optimizations such as a series of two-player zero-sum gaming.

If each trial solution is viewed as a separate species, the recombination operator is unreasonable. Fogel has stated that a crossover operator is merely a subset of all random mutations [14]. Further, evolutionary programming requires no gradient information to adjust objective variables. Therefore, evolutionary programming should be applied to optimization problems in which such information is unavailable.

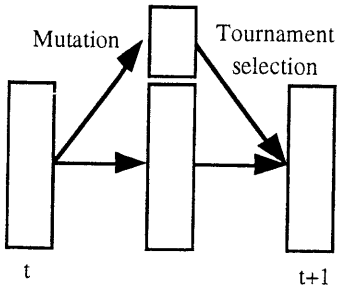


Figure 2.16 Evolutionary programming

2.6 Classifier System and Genetic Programming

This section presents other topics of genetic algorithms in brief. Genetic algorithms are easy to combine other techniques such as artificial intelligence, neural networks and fuzzy inference. In fact, the genetic algorithms have been hybridized with neural network [25,26], fuzzy reasoning [27,34,35], machine learning [12], and so on. Furthermore, the concept of genetic search is applied to classifier system and genetic programming. In this section, we present the classifier system and genetic programming in brief.

2.6.1 Classifier System

Classifier system [12] is a machine learning system that learns syntactically simple string rules to guide its performance in an arbitrary environment. The classifier system consists of three main components: rule and message system, apportionment of credit system, and genetic algorithm. A rule called a classifier, is represented as a production rule as follows:

if <condition> then <action>

The classifier is fired when the condition is satisfied. A classifier on genetic algorithms is represented as the string of a fixed length using binary coding. The apportionment of credit system ranks classifiers according to each reward from the environment using bucket brigade algorithm [12,14]. Based on the fitness of classifiers, the genetic algorithm generates new classifiers using reproduction, crossover, and mutation. In this way, a classifier system can generate a set of rules adapted to the environment.

2.6.2 Genetic Programming

J.Koza has proposed genetic programming [75,76]. The genetic programming works on hierarchically structured candidate solutions, while the genetic algorithm, in general, works on the simple string. This hierarchical structure enables the various representations for optimization problems.

The genetic programming provides a way to search the solution space of all possible programs composed of certain terminals and primitive functions to find a function which solves, or approximately solves a problem. The genetic programming has been applied to neural networks, control systems for autonomous robots, and so on [75,76] In the genetic programming, a population consists of hundreds or thousands of computer programs. The genetic programming also uses genetic operators based on Darwinian theory of evolution like the genetic algorithm. In general, a string of computer program has tree structure, and crossover operators are performed based on the tree-structure. After searching, genetic programming generates certain computer programs which can carry out the given task with sufficient performance.

The genotype of candidate solutions consists of primitive functions (+, -, *, /, and, or, etc.) and terminals (variables, coefficients). As a selection scheme, proportional selection, ranking

selection, and tournament selection are used. Crossover operator is performed by exchanging sub-trees between parents (Figure 2.17). Mutation operator is defined as a random change in the structure. The initial population is, in general, generated by three methods of 'full' method, 'grow' method, and 'ramped half-and-half' method. The 'full' method randomly generates a candidate solution whose length between the root and the endpoint is equal to the specific maximal depth. The 'grow' method randomly generates a candidate solutions whose length between the root and the endpoint is no greater than the specific maximal depth. Furthermore, the 'ramped half-and-half' is a hybrid method of the 'full' method and 'grow' method.

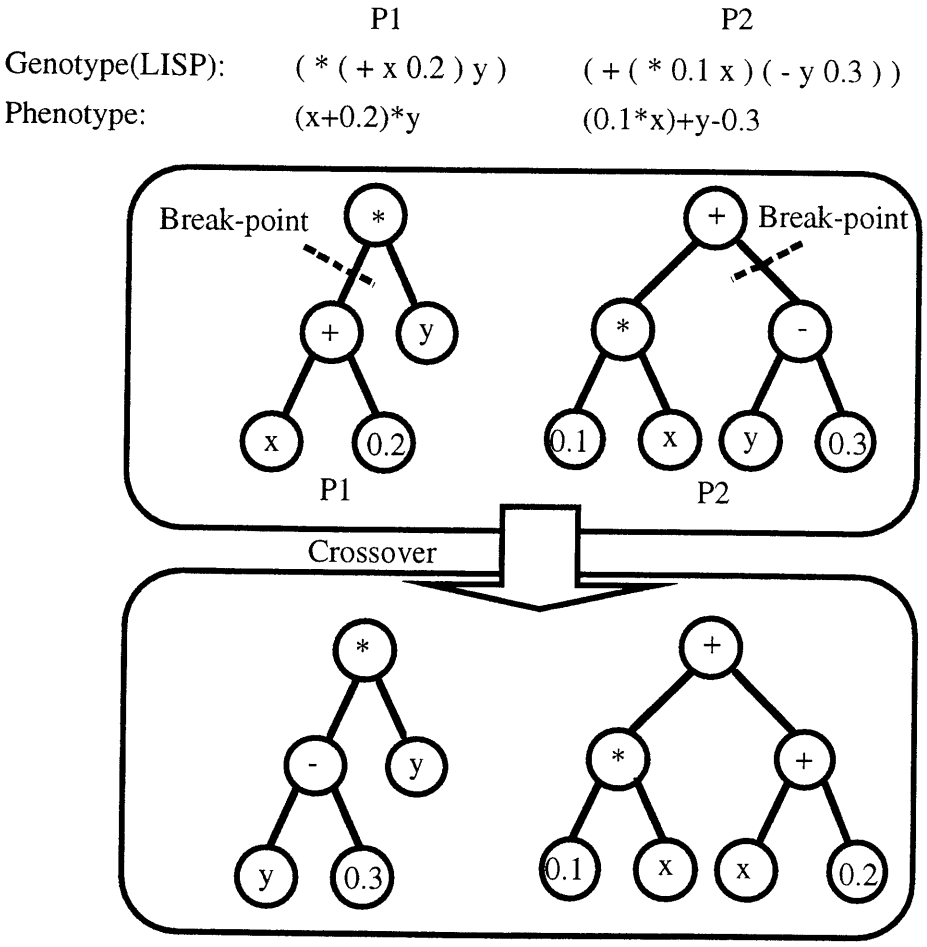


Figure 2.17 An example of crossover in genetic programming

2.7 Summary

This chapter presents evolutionary optimization methods in detail. Genetic algorithm works on the encoded string space and therefore various genetic operators have been proposed for improving the searching ability. The characteristic of genetic algorithms is as follows:

- Genetic algorithms, in general, require no prior knowledge of optimization problems, such as derivative information and continuity of the objective function.

- Genetic algorithms can attain optimal or quasi-optimal solutions without trapping local optimal solutions by simple symbolic operations.

- Selection schemes are classified into two main categories of a proportional selection scheme according to relative fitness and a competition selection scheme according to relative ranking in a population.

- There is interdependent relation between coding and crossover. The coding and crossover should be designed with considering the inheritance of genetic information from parents to offspring to effectively search the solution space.

- Some schemata generate new better schemata with the crossover operator and the number of a schema increases in a population with the selection operation. Building up bits of blocks called schemata, genetic algorithms can obtain optimal solutions (building block hypothesis).

- Operators and extension based on genetics, biology, ecology and mathematics can be easily incorporated into genetic algorithms to improve the searching ability.

Chapter 3 Genetic Algorithm with Age Structure

As mentioned in chapter 2, genetic algorithm simulates natural evolution and can find optimal or quasi-optimal solution. Evolutionary optimization methods have two approaches to improve searching ability. One is to mathematically improve optimization algorithms, and mathematical or computational power often gives us the quick solution. The other is to introduce natural phenomena, and nature often gives us effective ideas. In fact, ecological models based on Niche continuous alternation model of generation [63,67] and parallel models based on subpopulation [64~66], have been proposed and their effectivenesses have been also demonstrated through computer simulation. In the continuous alternation model of generation, offspring generated by crossover and mutation coexist with their parents and as the result, parents with high fitness are easy to survive. This fact indicates that the continuous alternation model of generation can maintain individuals with high fitness through all generations, but simultaneously the model has a possibility of premature local convergence. However, the continuous alternation model of generation does not have age structure as seen in nature. A population with age structure has an important characteristic of the death of an individual without considering its fitness. This chapter therefore proposes a genetic algorithm with age structure. Furthermore, the following sections discuss the effectiveness of the genetic algorithm with age structure.

3.1 Age Structure in Nature

This section presents a population with age structure in nature. In nature, there are, in general, two types of alternation models of generation: discrete model and continuous model [45]. Here the alternation of generation means an occurrence of within life cycle of an organism of two or more distinct forms. In the discrete alternation model of generation, the span between the birth of parents and the birth of offspring is one life cycle, that is, parents bear offspring and soon die. On the contrary, continuous alternation model of generation has overlapping span between the birth of parents and the birth of offspring, that is, the parents coexist with their offspring.

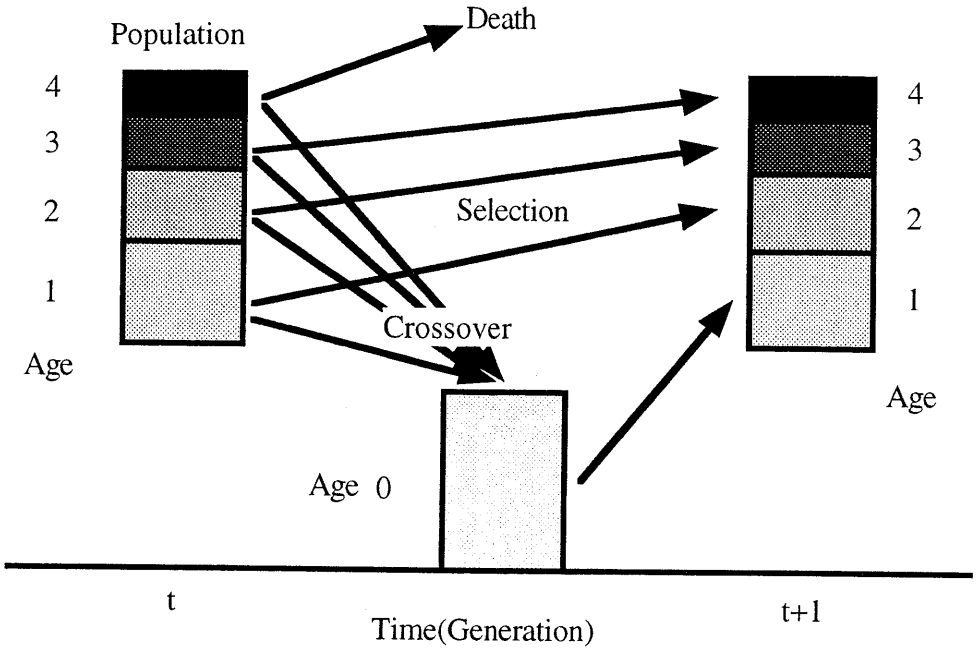


Figure 3.1 Population with age structure

Figure 3.1 shows a typical population with age structure in nature [45]. Here the age structure is considered as a discrete model, that is, we assume that all individuals age at the same time. Consequently, an individual of age x in generation t becomes age x+1 in generation t+1. In addition, each individual has a lethal age which means dying age. In Figure 3.1, the parents in generation t generate offspring at age 0 with recombination and then, parents at lethal age die and are removed from the population (the lethal age is 5 in this case). The remained parents and offspring are selected into the population in generation t+1. Let $n_{x,t}$ be the number of individuals at age x in generation t. Therefore, the number of offspring $n_{0,t}$ is as follows,

$$n_{0,t} = \sum_{x=1}^{m-1} b_x \cdot n_{x,t} \tag{3.1}$$

where b_x and m are birth rate of individual at age x and lethal age, respectively. Consequently, we obtain the population size at each age as follows,

$$n_{x,t+1} = p_x \cdot n_{x-1,t} \qquad (x = 1, \mathrm{L}\ , m-1) \tag{3.2}$$

$$n_{m,t+1} = 0 \tag{3.3}$$

where p_x is the survival rate at age x.

3.2 Genetic Algorithms with Age Structure

Genetic algorithm based on the idea that the better solution exists in the neighborhood of a candidate solution with high fitness, mainly utilizes proportional selection scheme and genetic operators. Consequently, an individual with high fitness can be selected into the next generation many times and this causes premature local convergence. We therefore propose genetic algorithm with age structure, to maintain genetic diversity in a population.

3.2.1 Aged Genetic Algorithm

We first introduce the concept of age into a discrete alternation model of generation. The discrete alternation model of generation forbids the coexistence of parents and their offspring. Figure 3.2 shows aged genetic algorithm; AGA. One generation is defined as a discrete time unit. Each individual in the AGA has parameters called *age* and *lethal age*. When its age attains to the predefined lethal age, it is removed from a population. All offspring, which are generated from parents by the crossover operator and mutation operator, is at age 0 and then their parents die. Consequently, parents can not coexist with their offspring. And each remaining individual increases in age every generation. Let N and $n_{x,t}$ be a population size and the number of individuals at age x in generation t, respectively. Therefore, the number of offspring, $n_{0,t}$, is as follows,

$$n_{0,t} = b\sum_{x=1}^{m-1} n_{x,t} \tag{3.4}$$

where b and m are the crossover possibility and lethal age, respectively. Furthermore, since the parents generated offspring are removed from the population, the number of all individuals, n_t', after aging is as follows,

$$n_x' = N - n_{m,t} = n_{1,t} + (1-b)\sum_{x=2}^{m-1} n_{x,t} \tag{3.5}$$

Furthermore, the next generation of a population is reproduced by the selection among the remaining individuals. The procedure of the AGA is as follows.

```
begin
  Initialization
  repeat
    Crossover
    Mutation
    Age_Operator
    Evaluation
    Selection
  until Termination_condition = True
end.
```

First, initialization randomly generates an initial population at age 0. Crossover and mutation generate offspring at age 0. Next, age_operator carries out aging process and removal process. The aging process adds each individual's age to one. Each individual therefore increases in age by one every generation. The removal process removes the parents which attain to the lethal age or generate offspring. Next, selection reproduces a new population from the remaining individuals.

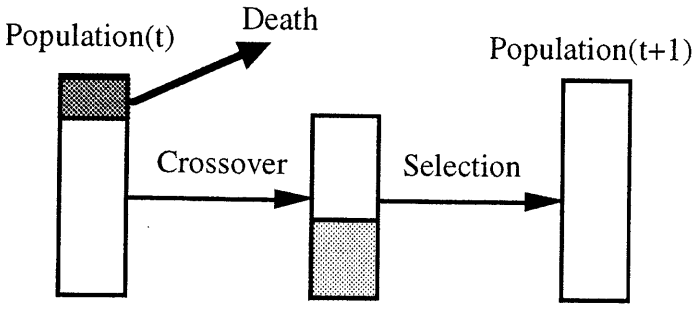


Figure 3.2 Aged genetic algorithm

3.2.2 Age Structured Genetic Algorithm

The AGA simply introduces the concept of age, but does not have coexistence of parents and their offspring. We therefore propose age-structured genetic algorithm; ASGA which is a continuous alternation model of generation (Figure 3.3). Each individual in ASGA has as same parameters of age and lethal age as the AGA. One difference from the AGA is as follows. When parents generate offspring, the parents don't die in the ASGA. This means that parents can coexist with their offspring. Therefore, though the number of offspring, $n_{0,t}$ is the same as that of the AGA, the number of all individuals, n_t', after aging is as follows,

$$n_x' = n_{1,t} + \sum_{x=2}^{m-1} p_x \cdot n_{x,t} \tag{3.6}$$

Where p_x is survival probability at age x. Each individual at age x survives into intermediate population, n_t', with survival probability p_x. This survival probability is introduced for simulating age structure in nature. Furthermore, the a population of next generation is reproduced by the selection among the remaining individuals. The procedure of the ASGA is as follows.

```
begin
  Initialization
  repeat
    Crossover
    Mutation
    Age_Operator
    Evaluation
    Selection
  until Termination_condition = True
end.
```

First, Initialization randomly generates an initial population at age 0. Crossover and mutation generate offspring at age 0. Age_Operator carries out aging process and survival process. The aging process adds each individual's age to one. Each individual therefore increases in age by one every generation. The survival process eliminates the individuals according to survival probability p_x. Here p_m is 0 because the individuals die of the lethal age. Next, selection reproduce a new population from the survived individuals.

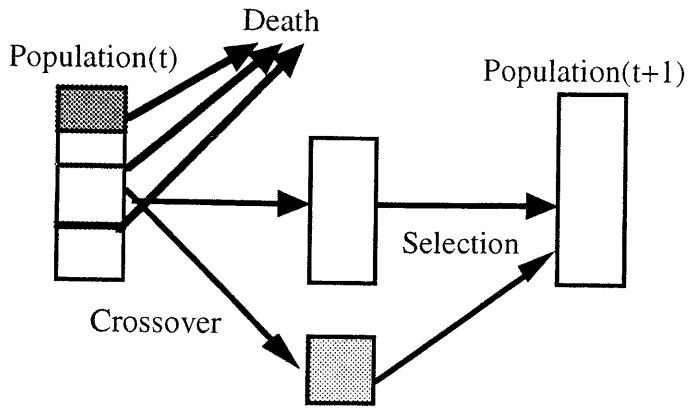


Figure 3.3 Age structured genetic algorithm

3.3 Features of Genetic Algorithm with Age Structure

There are two major differences between standard genetic algorithm and the genetic algorithm with age structure. First, the parents can survive into the next generation in the ASGA like steady-state genetic algorithm. That is, the parents are permitted the coexistence with their offspring. However, this feature is realized by steady-state model and genitor algorithm in the research of genetic algorithms [50,71]. The other feature is the removal of individuals without considering fitness value. An individual is removed from a population when its age attains to the lethal age in the AGA. In addition, individuals survives according to survival probability in the ASGA. However, the AGA has no age structure when crossover probability is 1.0, since all individual is replaced with their offspring. Consequently, the AGA under the crossover probability 1.0 corresponds to a standard genetic algorithm.

On the other hand, elitist selection scheme in genetic algorithms leaves the best individual into the next generation unconditionally, and it is proved that a population of genetic algorithm converges to the population with optimal solution under elitist selection scheme. However, the elitist selection scheme can cause premature local convergence at the same time. On the contrary, the AGA and the ASGA can control the increase of the same individual by lethal age and survival probability, respectively. Therefore, in the AGA and the ASGA, the design of lethal age and survival probability is very important. The selection pressure and convergence of a population are dependent on the design of lethal age and survival probability.

3.4 Application to Knapsack Problem

Knapsack problems, traveling salesman problems and scheduling problems are known as conventional and traditional combinatorial optimization problems [3]. We can solve these problems with an enumeration method in finite time, but an enumeration method takes much time as a problem size is large. Approximate optimization methods such as random search, simulated annealing and tabu search have been proposed for solving these problems. In this section, we apply genetic algorithm with age structure to a knapsack problem and compare genetic algorithm with age structure with other genetic algorithms.

3.4.1 Knapsack Problem

Combinatorial optimization problems can be fundamentally transformed into some types of 0-1 integer programming, but the transformed problems have a lot of constraint [3]. Knapsack problem, which is known as an NP-hard problem, is an integer programming problem of 0-1 variables [3,77]. This problem is represented by only the values 0 or 1. Consider that an event may or may not occur. If the event occurs, the value is 1. Otherwise, the value is 0. In this way, the 0-1 integer programming represents yes-no decisions. We then explain a knapsack problem.

Suppose n items are to be selected for carrying in a knapsack (Figure 3.4). The item i has value v_i and weight w_i. The objective of this problem is to select items to maximize their total value while their total weight is equal to or less than W. Let x_i be decision variables. If the item i is selected then x_i=1. Otherwise, x_i=0. The objective function is, therefore, defined as follows:

$$\text{maximize} \quad V_{sum} = \sum_{i=1}^{n} v_i x_i \tag{3.7}$$

$$\text{subject to} \quad W_{sum} = \sum_{i=1}^{n} w_i x_i \le W \qquad x_i = 0,1 \quad i = 1,\ldots,n \tag{3.8}$$

In this way, we can formulize the knapsack problem. We consider how to apply genetic algorithms to the knapsack problem. The genotype is represented by binary code. The locus (position) on a chromosome (string) represents the item number. For example, we consider a

string '0110' where the string length is 4 (i.e. the number of all items is 4). The string '0110' represents that the second and third of items are selected, while the first and fourth of items are not selected. As mentioned in chapter 2, a simple genetic algorithm is composed of roulette wheel selection, one-point crossover, and simple mutation. Simple mutation replaces a character on a string with the other character according to mutation probability. In the simulation, roulette wheel selection scheme and elitist selection scheme are used for the knapsack problem. The fitness value is defined as follows:

$$fitness = \begin{cases} V_{sum} & if \quad W_{sum} \le W \\ V_{sum} - \alpha \cdot (W - W_{sum}) & Otherwise \end{cases} \tag{3.9}$$

where α is coefficient for penalty in the case that the constraint is not satisfied. If *fitness* is less than zero, *fitness* is zero.

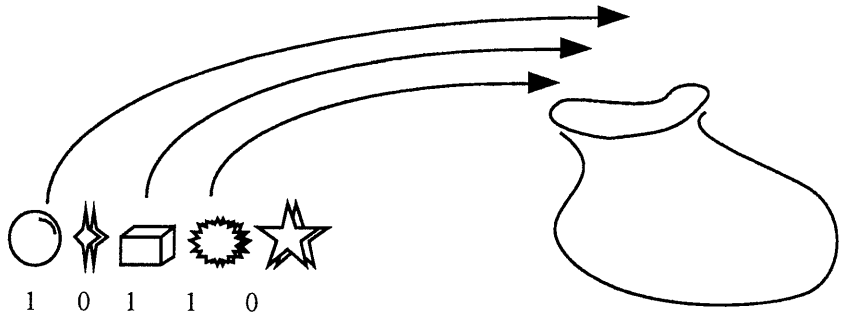


Figure 3.4 Knapsack Problem

3.4.2 Simulation Results of Knapsack Problem

The parameter concerning the knapsack problem is as follow. The number of items is 30. The weight and value of each item are randomly determined as the random value between 1 and 30. The upper bound of a selectable weight is 80% of the total weight of all items in this simulation example.

Crossover probability, mutation probability and maximal generation are 0.7, 0.001 and 150, respectively. Figure 3.5 shows typical simulation results of the knapsack problem with roulette wheel selection scheme. In Figure 3.5, the SGA represents a simple genetic algorithm. All algorithms converge toward the optima with small oscillation, since these algorithms does not keep the best individual at any generation. However, the AGA and the ASGA obtain better solutions than SGA. Figure 3.6 shows typical simulation results with elitist selection. In Figure 3.6, the EGA, the AGA_E and ASGA_E represent a genetic algorithm, AGA and ASGA with elitist selection schemes, respectively. Though the EGA can keep the fittest individual every generation, the EGA has a possibility to converge to local optima. However, the AGA and the ASGA have with very small oscillation in the early generation because even the best individual at any generation can be eliminated by the aging process. The oscillation of the AGA and ASGA in Figure 3.6 indicates that the best individuals eliminated from the population could not generate good offspring inheriting effective schema during their survival. Thus, the AGA and the ASGA can maintain genetic diversity more than the EGA. Table 3.1 shows simulation results of 30 trials. The AGA_E obtains the best solution in the simulation results. Consequently, we can obtain higher performance by introducing elitist selection scheme and aging process.

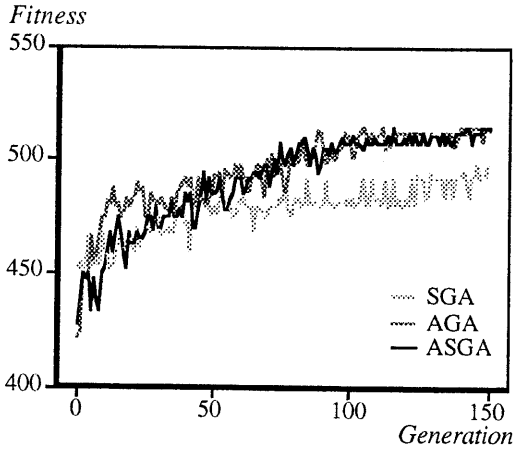


Figure 3.5 Simulation results of Knapsack problem by roulette wheel selection

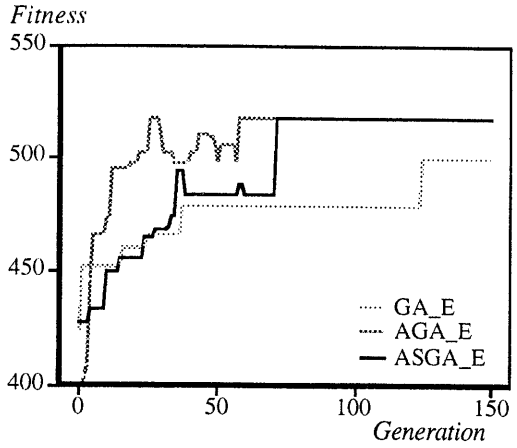


Figure 3.6 Simulation results of Knapsack problem by elitist selection

Table 3.1 Simulation results of Knapsack Problem of 30 trials

	SGA	AGA	ASGA
fitness	486.9	505.3	504.5
	GA_E	AGA_E	ASGA_E
fitness	493.5	508.5	507.3

3.5 Convergence of Age Structured Genetic Algorithm

We show the simulation results of a knapsack problem by genetic algorithm with age structure in the previous section. This section discusses the effectiveness of age structure and parameter setting concerning genetic algorithm with age structure.

3.5.1 Effectiveness of Introduction of Age Structure

We discuss the effectiveness of the introduction of age structure into genetic algorithm. We conducted a simple experiment. First of all, we consider only effect by the introduction of age structure. Consequently, we use the binary string whose length is one, that is, the genotype is '0' or '1'. The individual's fitness values of '0' and '1' are 12 and 10, respectively. We first prepare 100 individuals whose genotypes are '1'. Next, we compare how to increase the number of individuals of '0' by mutation and selection between the SGA and the AGA. The

procedure of this experiment is as follow;

Step 1: Initialization. The strings of 100 individuals are generated as '1'.
Step 2: Mutation
Step 3: Selection
Step 4: Count the genotype of '0'
Step 5: Go to step 2

Figure 3.7 shows experimental results of the SGA and the AGA. The AGA removes some individuals attaining the lethal age and as the result, the number of all individuals decreases in selection. Consequently, the selection pressure of the AGA is lower than that of SGA because the number of individuals of the AGA are less then that of the SGA. In general, the change of the number of a subpopulation size is dependent on its density in the population. Furthermore, the number of small subpopulation size have a lot of oscillation in increase, while the large population size is relatively stable in increase. Since the AGA reduces the selection pressure, the small size of subpopulation can reproduce relatively many individuals (see Figure 3.7). Thus, the AGA can control selection pressure by aging process.

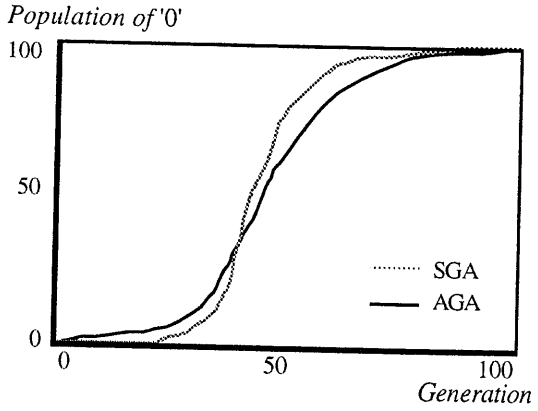


Figure 3.7 Change of population size of SGA and AGA

(fitness value of '0' :12, fitness value of '1':10)

Next, we consider the distribution of the relation between fitness values and age of individuals. Figure 3.8 shows fitness values of individuals against each age at generation 50 in solving the knapsack problem where the lethal age is 5. This shows that the individuals with high fitness

in the population survive into the next generation. And even the individuals with high fitness are eliminated by aging process. Consequently, the AGA can prevent individuals with high fitness from increasing in a population. On the other hand, there are some methods for maintaining genetic diversity such as ranking selection scheme and fitness scaling. Though these methods can reduce or increase the difference of fitness values between individuals, these methods can not control the number of over-increasing individuals. To the contrary, the AGA can control the number of over-increasing individuals by aging process, while the AGA performs selection between remaining individuals below a lethal age. Figure 3.9 shows fitness values of individuals against each age at generation 50 where the lethal age is 9. In this case, most of individuals are eliminated before attaining the lethal age, since the lethal age is relatively high. This indicates that the behavior of the AGA, whose lethal age is relatively high, is similar to that of SGA because most of individuals are eliminated by selection operation without relation to its age. To summarize, the AGA can control selection pressure by aging process and maintain relatively genetic diversity in a population.

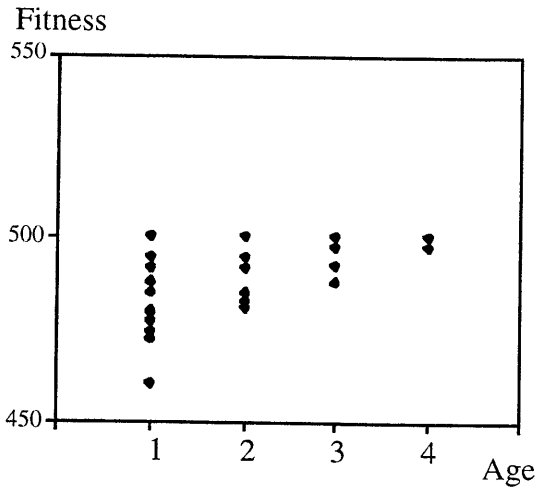


Figure 3.8 Fitness value of individual against age by AGA (lethal age = 5)

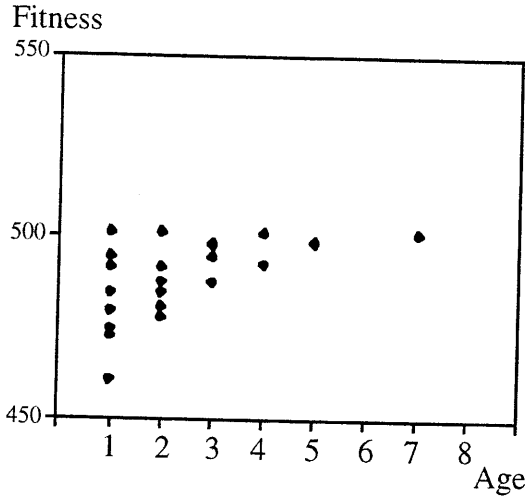


Figure 3.9 Fitness value of individual against age (lethal age = 9)

3.5.2 Discussion Concerning Lethal Age and Selection Pressure

Though the AGA can maintain relatively genetic diversity in a population, the AGA has the probability of the elimination of the best solution at each generation as same as genetic algorithm with proportional selection scheme. In this subsection, we discuss the relation between lethal age and crossover probability. Figure 3.10 shows the average of fitness values of the best individuals at generation 150 of 50 trial against crossover probability 0.4, 0.6 and 0.8 of the AGA. From Figure 3.10, the best lethal ages against crossover probability 0.4, 0.6 and 0.8 are 6, 5 and 4, respectively. This indicates that the AGA can obtain high performance when the lethal age is low as crossover probability is high. Figure 3.11 shows the average of fitness values in the case of the ASGA. From Figure 3.11, the best lethal ages against crossover rates 0.4, 0.6 and 0.8 are 6, 5 and 3, respectively. This result is the same as the AGA. Next, we show the death rate of individuals by attaining the lethal age in Figure 3.12 and Figure 3.13. These figures show that the death rate is low as the lethal age is high. From Figure 3.12 and Figure 3.13, the death rate corresponding to the best lethal age against crossover probability lie between 3% and 5%. To summarize, genetic algorithm with age structure has the close relation between crossover rate and lethal age, and we should design genetic parameters so that the death rate may lie between 3% and 5%.

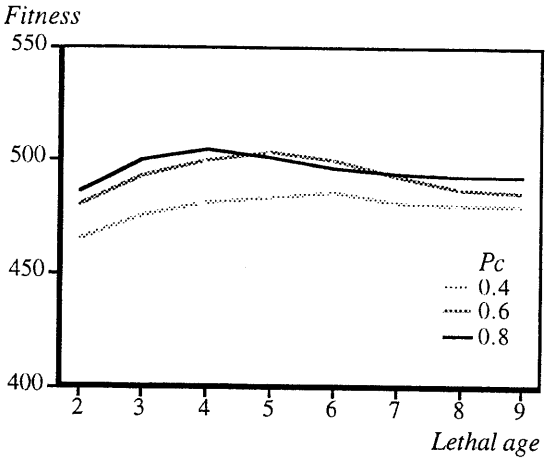


Figure 3.10 Average of fitness values concerning crossover probability of AGA

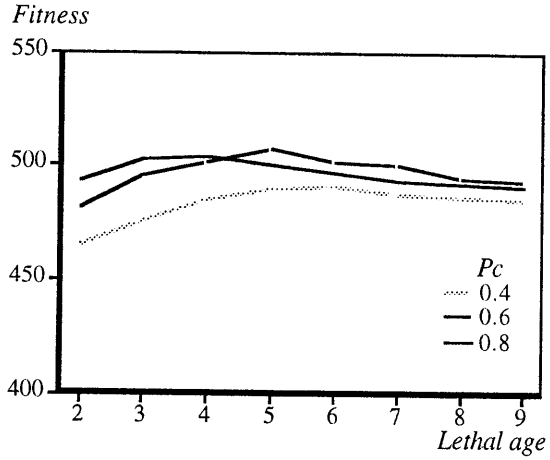


Figure 3.11 Verge of fitness values concerning crossover probability of ASGA

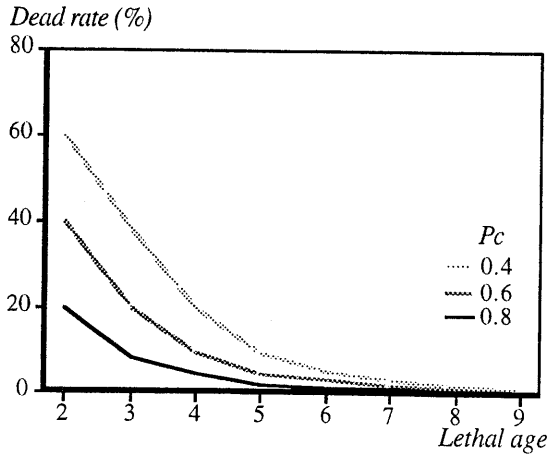


Figure 3.12 Death rate of individuals which attain the lethal age in case of AGA

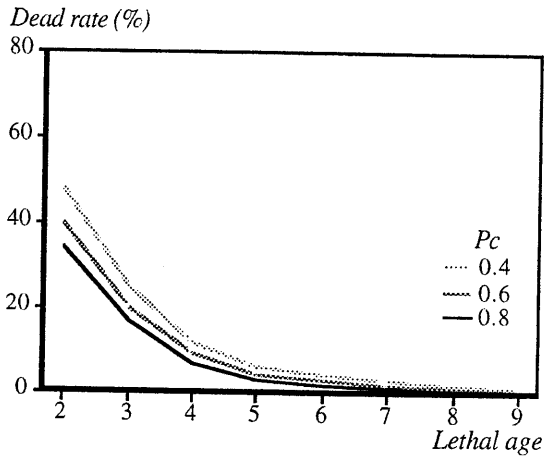


Figure 3.13 Death rate of individuals which attain the lethal age in case of ASGA

3.5.3 Discussion Concerning Lethal Age and Population size

In this subsection, we discuss the relation between lethal age and population size. Figure 3.14 and Figure 3.15 show the average of fitness values against the lethal age concerning population sizes of the AGA and ASGA, respectively. Though the design of the lethal age affects the performance of solutions in case of small size of population, the effect of lethal age reduces as the population size increases. The reason is that the AGA can maintain genetic diversity in a population without relation to lethal age since a large size of population naturally has genetic diversity. To the contrary, even the small size of population can obtain high performance by the design of lethal age and survival probability.

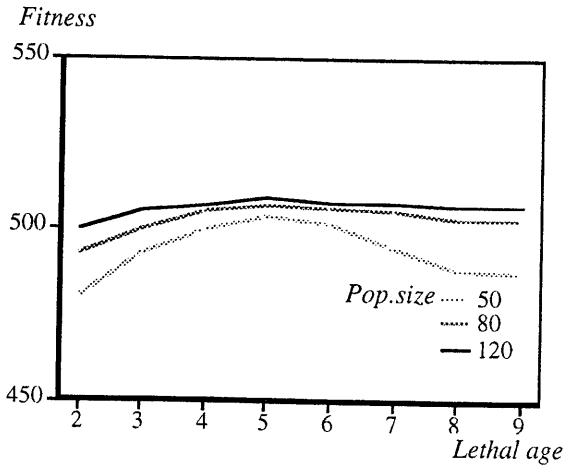


Figure 3.14 Average of fitness values concerning population size of AGA

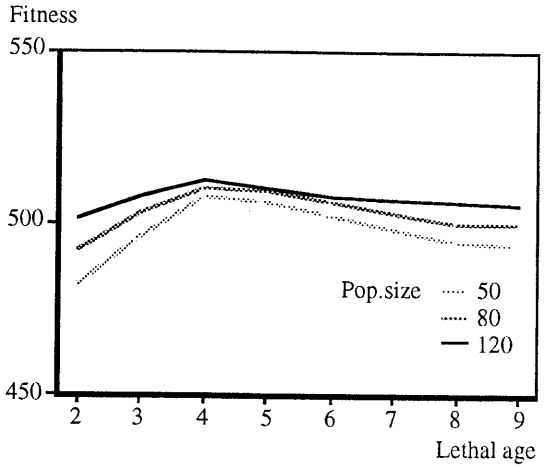


Figure 3.15 Average of fitness values concerning population size of ASGA

3.6 Summary

This chapter proposes a genetic algorithm with age structure, which has two major differences from a simple genetic algorithm. One is the coexistence of parents and their offspring. The other is the removal of individuals by aging process without relation to fitness value. Furthermore, the genetic algorithm with age structure is applied to a knapsack problem. Simulation results indicate the following points.

- Genetic algorithm with age structure can prevent individuals with high fitness from over-increasing in a population.

- Genetic algorithm with age structure can control selection pressure by aging process and maintain relatively genetic diversity in a population.

- Genetic algorithm with age structure has the close relation between crossover probability and lethal age, and we should design genetic parameters so that the death rate may lie between 3% and 5%.

Chapter 4 Evolutionary Optimization Methods based on Virus Theory of Evolution

Optimization methods can be divided into two categories of exact and approximate methods, as mentioned in chapter 1. Genetic algorithm fundamentally belongs to the approximate method and its searching ability lies in crossover operator which combine schemata between individuals. Schema theorem is well-known as a fundamental theorem of the genetic algorithm (see chapter 2). The increase of effective schemata enables the effective search of the solution space and makes all the population evolve toward optimal solutions. Though the genetic algorithm makes it possible to find optimal or quasi-optimal solutions with less computation cost, genetic algorithm has a problem of a premature local convergence (see chapter 2). One reason why the premature local convergence occurs in a population is that a proportional selection scheme increases not only effective schemata but also ineffective schemata. Therefore, this chapter proposes a virus-evolutionary genetic algorithm (VEGA) to improve searching ability of the genetic algorithm by operating only effective schemata.

4.1 Virus Evolutionary Genetic Algorithm

Genetic algorithm is a stochastic optimization method based on Neo-Darwinism [42]. Neo-Darwinism provides us effective ideas. In fact, the genetic algorithm has been extended by applying biologically inspired operators, ecological niche, and so on. However, genetic algorithm is based on the concept of natural selection. Natural selection plays the role of the increase of better individuals. Recently, there have been many questions concerning natural selection in evolution. One of the questions is that the driving force for evolution may not lie in natural selection. With the progress of molecular biology, some evolutionary theories have been proposed. We propose a genetic algorithm based on virus theory of evolution.

4.1.1 Virus Theory of Evolution

With the progress of molecular biology, various theories of evolution such as Neo-Darwinism, neutral theory of molecular evolution, Imanishi's evolutionary theory, serial symbiosis theory,

virus theory of evolution, have been proposed for explaining the mechanism of evolution [42~46]. However, most of evolutionary theories can not explain all evolutionary evidences, though they can explain the process of evolution in a sense.

The virus theory of evolution is based on the view that virus transduction is a key mechanism for transporting segments of DNA across species [43]. Here the transduction means the genetic modification of a bacterium by genes from another bacterium carried by a bacteriophage. Most of viruses can easily cross species barriers, and are often transmitted in nature directly from individuals of one phylum to another. This fact means that a virus can transmit its gene to a host population as horizontal propagation. Furthermore, whole virus genomes may be incorporated into germ cells and transmitted from one generation to the next generation as vertical inheritance.

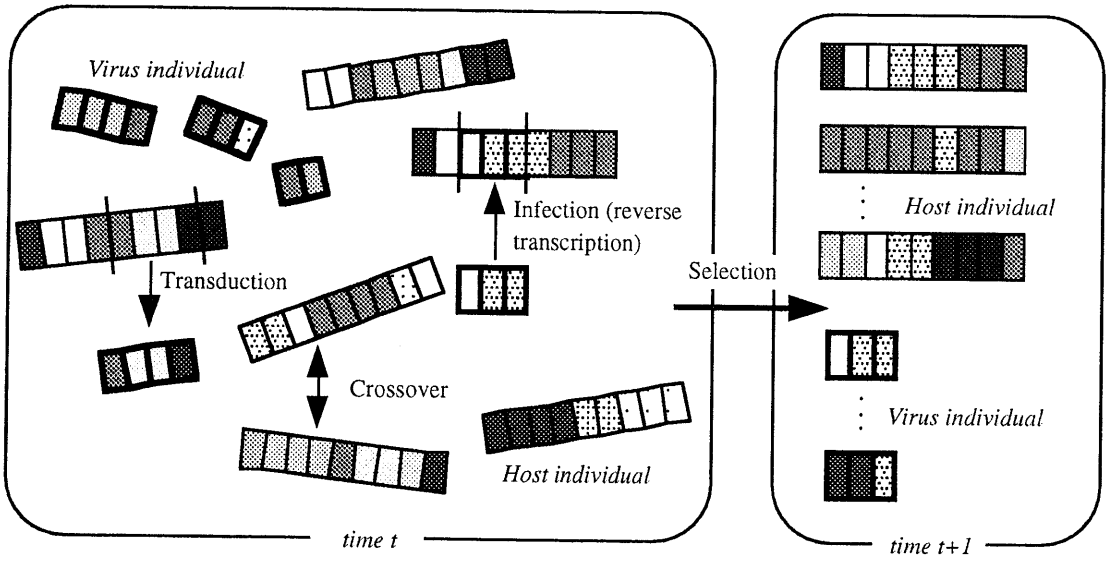


Figure 4.1 Virus-evolutionary genetic algorithm; VEGA

4.1.2 Virus-Evolutionary Genetic Algorithm Architecture

This subsection proposes a virus-evolutionary genetic algorithm; VEGA. In virus theory of evolution, living things evolve with horizontal propagation and vertical inheritance of genetic information. To realize horizontal propagation and vertical inheritance on genetic algorithm,

we use two types of populations; host population and virus population (Figure 4.1). Here the host population and virus population are defined as a set of candidate solutions and a substring set of the host population, respectively. And virus individuals transmit partial genetic information among host individuals by virus infection. To incorporate the virus infection mechanism, we adopt a steady state genetic algorithm; SSGA (see chapter 2). The SSGA is continuous generation model and is easy to regulate generation gap. The SSGA, in general, replaces a pair of individuals with new individuals generated by genetic operators every generation. Thus, the VEGA is composed of crossover, mutation, selection and virus infection. The procedure of the VEGA is as follows:

```
Initialization
  repeat
    Selection
    Crossover
    Mutation
    Virus _infection
    Replacement
  until Termination _condition = True
end.
```

Initialization randomly generates an initial host population, and then a virus individual is generated as a substring of a host individual. 'Delete least fitness' [71] is used as the selection scheme. Crossover and mutation are genetic operators dependent on an optimization problem. Virus infection operator is introduced as new searching operators. Here new individuals which virus individuals infected, are generated as children. After all virus' infections, if a fitness of a host individual is improved, then the generated child is replaced with its parent (Figure 4.2). Consequently, the successfully infected host individuals survive into the next generation. This indicates the virus infection operator is regarded as a local hill climbing operator. The string length of a host individual is predefined as a constant. And the length of a virus individual, which is defined as a variable, extends with evolution of the host population.

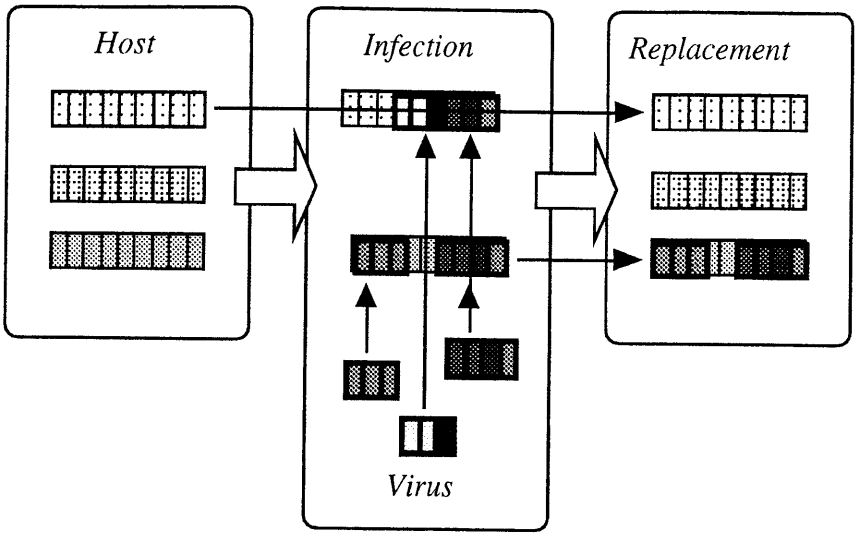


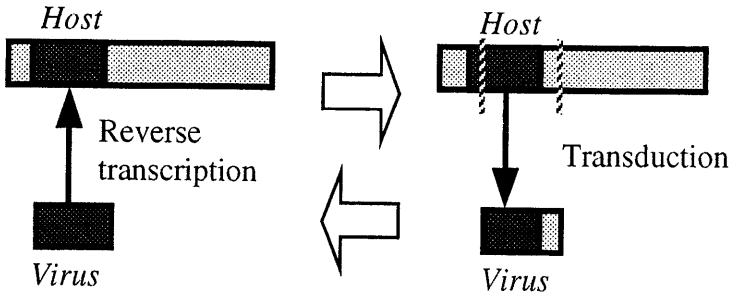
Figure 4.2 Virus infection and replacement in VEGA

4.1.3 Virus Infection Operators

Virus infection operators are main processes in the VEGA. As mentioned before, a virus has a capability to transmit segment of DNA in a host population. The virus infection operators are (a) the reverse transcription operator and (b) the transduction operator (Figure 4.3).

(a) Reverse transcription operator: A virus overwrites its substring on the string of a host individual for generating new host individuals.

(b) Transduction operator: A virus takes out a substring from the string of a host individual for generating new virus individuals.



(a) Reverse transcription operator (b) Transduction operator

Figure 4.3 Virus infection operators

A virus individual also has fitness value. Each virus has $fitvirus_i$ as a strength about the virus infection. We assume that $fithost_j$ and $fithost'_j$ are the fitness values before and after infections of an infected host individual j, respectively. The $fitvirus_{i,j}$ denotes the difference between $fithost_j$ and $fithost'_j$, which is equal to the value obtained by infecting to the host individual:

$$fitvirus_i = \sum_{j \in S} fitvirus_{i,j} \tag{4.1}$$

$$fitvirus_{i,j} = fithost'_j - fithost_j \tag{4.2}$$

where i is the virus number and S is a set of the host individuals that the virus i infected. Therefore, $fitvirus_i$ denotes the improvement of the fitness values of all the infected host individuals. The number of virus infections is controlled under its virus infection rate.
Each virus has infection rate, $infrate_i$, satisfying $0 \leq infrate_i \leq 1.0$, for performing a reverse transcription operator to a host population.

$$infrate_{i,t+1} = \begin{cases} (1+\alpha) \cdot infrate_{i,t} & if\ fitvirus_i \geq 0 \\ (1-\alpha) \cdot infrate_{i,t} & if\ fitvirus_i < 0 \end{cases} \tag{4.3}$$

where α (>0) is coefficient. When $fitvirus_i$ is high, $infrate_i$ becomes high. Furthermore, each virus has life force as follows:

$$life_{i,t+1} = r \times life_{i,t} + fitvisus_i \tag{4.4}$$

where t and r means the generation and the life reduction rate, respectively. The life force means an index of the contribution to a host population. In initialization in the procedure of the VEGA, these virus parameters are initialized as follows;

$$infrate_{i,0} = infrate_{init} \tag{4.5}$$

$$life_{i,0} = 0 \tag{4.6}$$

The procedure of a virus infection is shown in Figure 4.4. First, a virus performs the reverse transcription to a host individual randomly selected out of the host population. Next, the VEGA evaluates the fitness of the infected host individual and calculates $life_{i,t}+1$. If $life_{i,t}+1$ takes a negative value, the virus individual transduces a new substring with the transduction operator from a randomly selected host individual. Otherwise, the virus individual transduces a partially new substring from one of the infected host individuals with the transduction operator for evolving for itself. Thus, the host and virus populations coevolve through the genetic operators and virus infection operators.

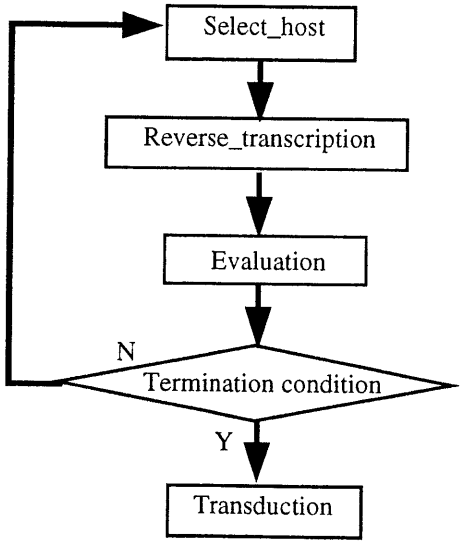


Figure 4.4 The procedure of virus infection

4.2 Features of Virus-Evolutionary Genetic Algorithm

The most significant characteristic of the VEGA is to increase schemata directly in a host population by virus infection, though a standard genetic algorithm increases schemata in a population as the results of crossover and selection. The virus infection operators enable the increase of effective schemata with reverse transcription and transduction. The reverse transcription plays the roles of crossover and selection simultaneously, since the reverse transcription generates new individuals overwriting on host individuals according to the virus infection rate. That is, the virus infection rate is equal to the frequency increasing a schema in a population. Furthermore, the transduction generates new virus individuals, and makes a virus population evolve. The roles of virus infection are simply summarized as follows,

1. Reverse transcription directly increases effective schemata.
2. Reverse transcription directionally generates new host individuals.
3. Transduction changes virus individuals every generation.

Coevolution of the virus population and host population enables the quick solution of an optimization problem.

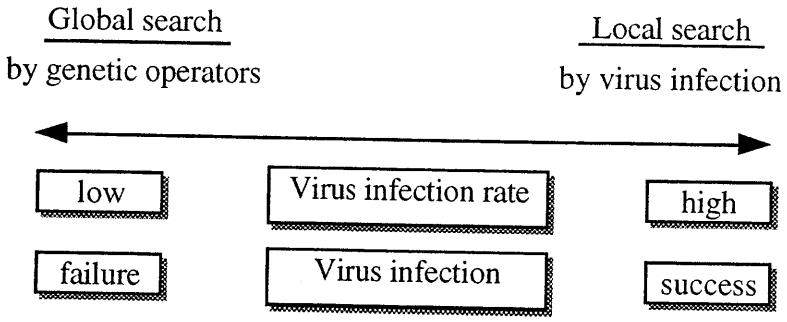


Figure 4.5 Searching ratio in VEGA

Furthermore, a virus infection rate is a key parameter for searching solution space (Figure 4.5). When a virus succeeds in its infection, the virus infection rate becomes high. As the result, the VEGA mainly performs local search by virus infection operators. To the contrary, when

the virus individual fails in its infection, the virus infection rate becomes low. Consequently, the VEGA mainly performs global search by genetic operators. In this way, the VEGA can self-adaptively change the searching ratio between global search and local search according to the current state of the virus and host populations.

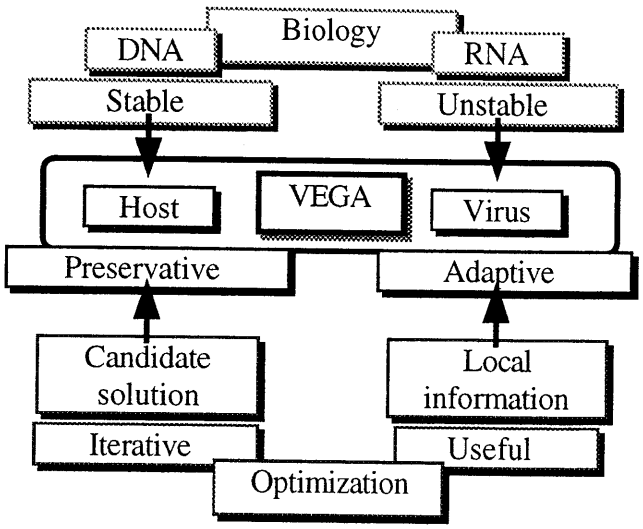


Figure 4.6 VEGA from view points from biology and optimization

We discuss the features of the VEGA from the view points of optimization and biology (Figure 4.6). First, we discuss the feature of the VEGA from the biological point of view. The structure of DNA is stable and hard to break, while the structure of RNA is unstable and easy to break in nature. That is, living things should basically inherit genetic information (DNA) adapted to their environments. The inheritance of genetic information from parents is performed by the reproduction in the VEGA. On the other hand, a virus can change its substring for itself and immediately adapts to its host population. From the view point of optimization, this kind of population-based search corresponds to multi-point and iterative search. The virus population has local information of a candidate solution and searches the solution space based on the local information. The local information is very important and useful for searching the solution space because the local information often gives a direction or index for search. For example, a subtour as local information is often used for solving a traveling salesman problem.

4.3 Ecological Virus Evolutionary Genetic Algorithm

Standard genetic algorithm is based on the concept of crossover, mutation and natural selection, but the concept of ecology is not introduced into the standard genetic algorithm. Ecological models of genetic algorithm have been proposed (see chapter 2) and their simulation results indicate the ecological model prevents a population of candidate solutions from converging to local optima and realizes the localized evolution of population. Therefore, this section proposes an ecological model of VEGA (E-VEGA).

The E-VEGA also has virus and host populations. A host individual is placed on a planar grid (Figure 4.7). Genetic operations such as selection and crossover are restricted to neighborhoods on the planar grid. A virus can move on the planar grid with its direction vector, and transmit local genetic information as a substring among host individuals.

The procedure of the E-VEGA is fundamentally the same as the VEGA. The selection replaces a host individual with the host individual selected from its neighborhood on the planar grid. Next, the crossover operator is performed between host individuals randomly selected from its neighborhood with crossover probability. Next, each virus infects to host individuals of its neighborhood according to virus infection rate. These processes are repeated until the termination condition is satisfied.

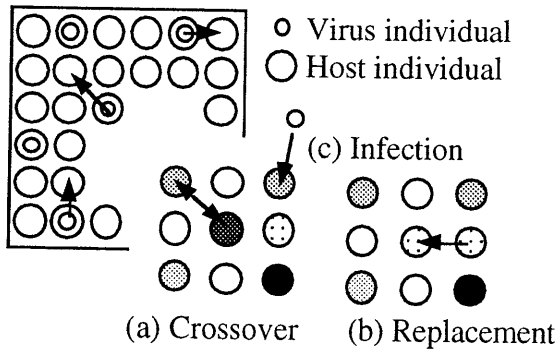


Figure 4.7 Ecological model of virus-evolutionary genetic algorithm

4.4 Virus Evolutionary Algorithm

In evolutionary programming, an individual is regarded as species and evolutionary programming does not use crossover operator because different species can not cross over [13,73]. However, virus theory of evolution states that a virus can transmit genetic information among different species. Accordingly, this section proposes virus evolutionary algorithm; VEA. The VEA also has a host population and virus population. An individual is represented as a set of variables since the VEA is mainly applied to numerical optimization problems. And a virus individual is a subset of variables and transmits the subset among host individuals by virus infection. The procedure of the VEA is as follows:

```
Initialization
  repeat
    Selection
    Mutation
    Virus _infection
    Replacement
  until Termination _condition = True
end.
```

Initialization randomly generates an initial host population with uniform random values, and then a virus individual is generated as a subset of a host individual. 'Delete least fitness' [7] is used as the selection scheme. As a mutation operator, self-adaptive mutation is used. Let x_i be a variable of objective function.

$$x_{j+n,i} = x_{j,i} + N(0, a_i \times fitness + b) \tag{4.7}$$

where a_i and b are a coefficient for scaling and offset in order for the variance of normal random variable not to be zero. This self-adaptive mutation adds small perturbation when fitness value is small. And a normal random value is used in the case of numerical minimization problem where the optimal value is zero. However, we don't know optimal value in real optimization problems. Therefore, we use the following mutation with which the maximal fitness value of a population is normalized to one.

$$x_{j+n,i} = x_{j,i} + N(0, a_i \times \frac{fitness}{\max fitness} + b) \tag{4.8}$$

Virus infection operators are also used here. A virus overwrites its substring on a host individual with reverse transcription.

4.5 Application to Conventional Optimization Problems

This section applies a virus-evolutionary genetic algorithm; VEGA, and a virus evolutionary algorithm; VEA, to conventional and traditional optimization problems. First, we apply the VEGA to a knapsack problem and a traveling salesman problem. Next, we apply the VEGA and VEA to a function optimization problem. Furthermore, we discuss the effectiveness of virus infection through computer simulation and analyze the behavior of the VEGA on a Markov chain.

4.5.1 Application to Knapsack Problem

4.5.1.1 Knapsack Problem

Knapsack problem is an integer programming problem of 0-1 variables, as mentioned chapter 3. This problem is represented by only the values 0 or 1. As mentioned in chapter 3, the objective function is as follows:

$$\text{maximize} \quad V_{sum} = \sum_{i=1}^{n} v_i x_i \tag{4.9}$$

$$\text{subject to} \quad W_{sum} = \sum_{i=1}^{n} w_i x_i \le W \qquad x_i = 0,1 \quad i = 1,\ldots,n \tag{4.10}$$

We then consider how to apply the VEGA to a knapsack problem as follows. The genotype is represented by binary code. The fitness value is defined as follows:

$$fitness = \begin{cases} V_{sum} & if \quad W_{sum} \le W \\ V_{sum} - \alpha \cdot (W - W_{sum}) & Otherwise \end{cases} \tag{4.11}$$

where α is coefficient for penalty in the case that the constraint is not satisfied. If *fitness* is less than zero, *fitness* is zero. We use uniform crossover (see chapter 2) and simple mutation (see chapter 2) as genetic operators. The uniform crossover generates new individuals according to a randomly generated mask pattern. As mentioned before, a virus individual can transmit a substring among host individuals. A substring of a virus individual consists of three characters {0,1,*} and is the same length as that of the host individual. The character '*' denotes don't care mark. A virus individual does not perform the reverse transcription in the same position where there is a '*'. In this case, while the length of the virus individual is constant, the order of the virus is variable. Figure 4.8 shows an example of the reverse transcription. The reverse transcription overwrites the substring of the virus individual on a randomly selected host individual. The transduction for evolving virus individuals has two types of operators(Figure 4.9). The first one is to copy genes from a host individual according to a copy probability per gene. The other is to replace some genes with the character '*' according to a cut probability per gene. If the virus improves the fitness values of host individuals, the copy operator is performed at the transduction probability. Otherwise, the replacement operator is performed. An initial virus population is generated from the host population with the use of the transduction operator.

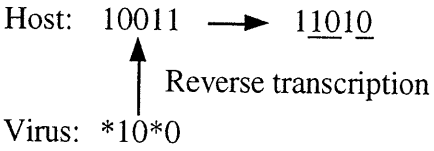
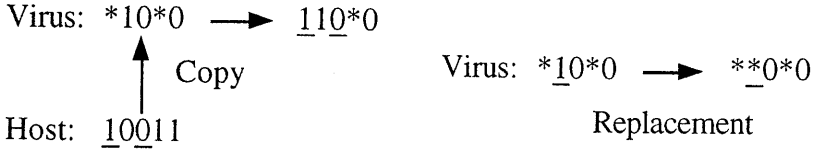


Figure 4.8 A reverse transcription operator for binary natation



(a) Copy operator (b) Replacement operator

Figure 4.9 A transduction operator for binary natation

4.5.1.2 Simulation Results of Knapsack Problem

The number of items in the knapsack problem is 50. The weight and value of each item are randomly determined as the random value between 1 and 30. The upper bound of a selectable weight is 80% of the total weight of all items in this simulation example. Table 4.1 shows the parameters of the SSGA, VEGA and virus infection. The number of evaluations is used in the numerical simulation so that the search times of the VEGA is equal to that of the SSGA. Here the search times means the sum of the times of crossover operator and reverse transcription. The infection rate means the rate at which a virus transcribes to a host population. The transduction rate means the rate at which a virus transduces after infection.

Table 4.1 Parameters of SSGA, VEGA, and virus infection

	SSGA	VEGA
Population size	100	100
String length	50	50
Crossover rate	0.8	0.8
Mutation rate	0.001	0.001
Evaluations	10000	10000

Virus infection	
Virus population size	10
Life reduction rate(r)	0.9
Max infection rate	0.1
Initial infection rate	0.05
Transduction rate	0.6
Copy /Cut rate	0.05

Figure 4.13 and Figure 4.11 show simulation results of the SSGA and VEGA, respectively. The fitness value in each figure is the average of 50 trials. On the whole, the convergence of the VEGA is faster than that of the SSGA. Furthermore, when the population size is small in both the SSGA and VEGA, on the average the SSGA and VEGA attain local minima because the premature convergence often occurs. On the contrary, the convergence of a large population size is slower, but the VEGA and SSGA reach global minima. Figure 4.12 shows the comparison of on-line performance which is the average of the highest fitness value at each generation. The on-line performance of the VEGA outperforms the SSGA in all population sizes. Table 4.2 shows the average of fitness values obtained after 10000 evaluations. The VEGA, whose population size is 100, attains the highest average of fitness values.

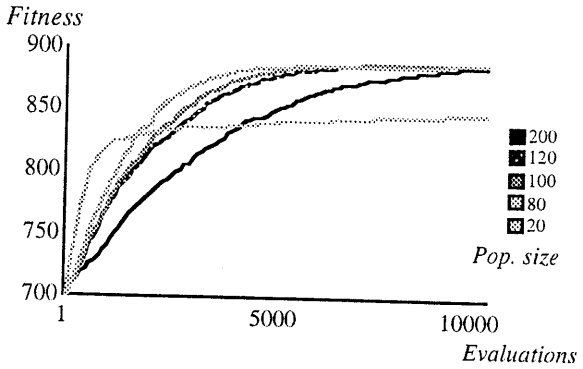


Figure 4.10 Simulation results of knapsack problem (SSGA)

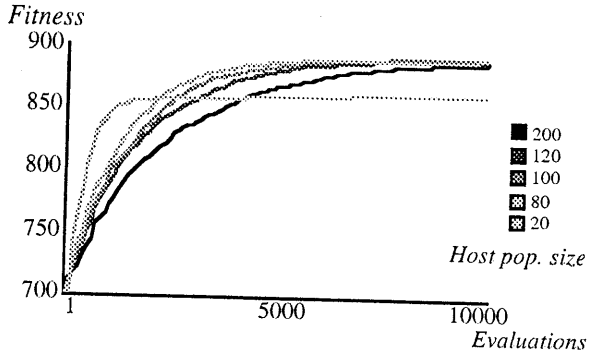


Figure 4.11 Simulation results of knapsack problem (VEGA)
(virus population size is 10% of host population, maximal infection rate = 0.1)

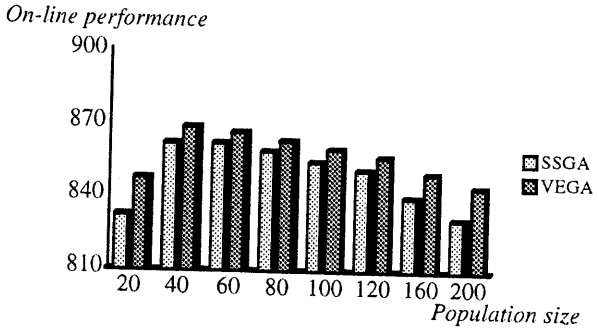


Figure 4.12 On-line performance of knapsack problem

Table 4.2 Maximal fitness value obtained at the last generation

Host pop. size	20	40	60	80	100	120	160	200
SSGA	848.74	879.78	885.23	887.82	888.80	889.11	888.66	887.15
VEGA	857.12	885.72	888.94	889.10	889.43	889.02	888.48	887.20

Next, we apply the SSGA and VEGA to the knapsack problem with 100 items as a more difficult problem. Consequently, the string length of an individual and the number of evaluations are altered into 100 and 20000, respectively.

Figure 4.13 shows simulation results of the mutation, SSGA and VEGA. In the case of the mutation, only a bit mutation is performed as the genetic operator. A fitness value in this figure is the average of 50 trials. On the whole, the convergence of the VEGA is fastest and best in the simulation results.

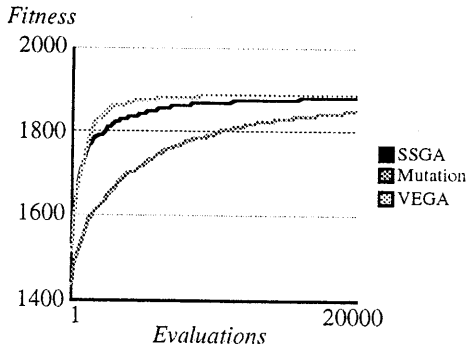


Figure 4.13 Simulation results of knapsack problem

4.5.1.3 Optimal Virus Infection Probability

The performance of the VEGA depends on the virus infection operators, since the times of reverse transcription of a virus individual to host individuals determines the frequency of the horizontal propagation in a host population. We discuss the effectiveness of the virus infection operators through numerical simulation of the knapsack problem.

Figure 4.14 shows the simulation results concerning virus population sizes, where the host population size and the infection rate are 100 and 0.1, respectively. The larger a virus population size is, the slower the convergence is. Next, we consider the case of a large size of virus population size. Figure 4.15 shows simulation results concerning high infection rates, where the host and virus population sizes are 100 and 200, respectively. When the infection rate is high, the virus infects almost host individuals. As a result, premature local convergence is easy to occur.

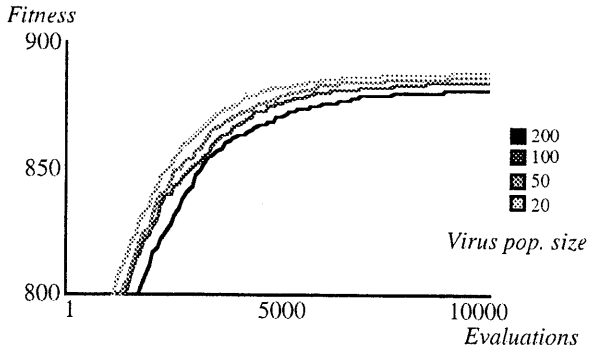


Figure 4.14 Comparison of simulation result concerning virus population size (host population size = 100, maximal infection rate = 0.1)

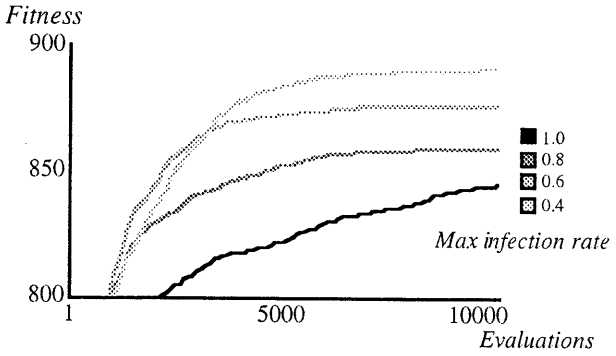


Figure 4.15 Comparison of simulation result concerning virus population size (host population size=100, virus population size=200)

Next, in order to discuss optimal parameters concerning the virus infection, we carry out some simulations concerning a small virus population size. Figure 4.16 shows the comparison of simulation results in the case of 60 host individuals. The VEGA attains the best fitness value when the virus population size and the maximal infection rate are 6 and 0.1, respectively.

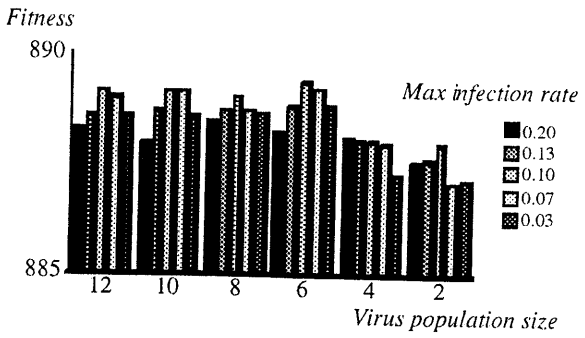


Figure 4.16 Comparison of simulation results concerning small virus population size (host population size=60)

These simulation results indicate that the VEGA attains the best solution when the host population size and maximal infection rate are about 10% of host population size and about 0.1, respectively. The reason would be as follows. When the maximal infection rate is high, a virus individual infects to almost host individuals. As a result, a genetic diversity lacks in the host population. To the contrary, when the maximal infection rate is very low, a virus individual can not quickly propagate effective schemata in the host population. Consequently, the virus infection is similar to crossover operator and selection operation.

4.5.2 Application to Traveling Salesman Problem

4.5.2.1 Traveling Salesman Problem

A traveling salesman problem (TSP) is also well-known as an NP-hard problem. In the TSP, a traveling salesman must visit each of n cities exactly once in order to minimize a total cost of the travel. Here we apply the VEGA to a plane TSP with 25 cities, and 3 dimensional lattice TSPs with 27 cities and 64 cities. The objective of the TSP is to minimize the length of a round tour. The sequence of a round tour is represented by various ways [79~82]. Here an explicit list of cities is used. Therefore, the traveling salesman problem can be transformed into the permutation optimization problem which has $(N\text{-}1)!/2$ possible solutions for N cities.

The application of genetic algorithms to the traveling salesman problem is shown as follows. The representation of a tour is defined as a list of cities. Consequently, the genotype is

defined as the positive number between 1 and N. The locus (position) on the chromosome (string) represents the traveling order in a tour, that is, the number i on the string denotes the city number. And the overlapping of the same number is prohibition on the string. In addition, the tour is closed by returning from the last city on the string to the first one. For example, consider that we solve a TSP with 5 cities from the number 1 to 5 as follows,

12345

This permutation is the same as a reverse order as follows:

54321

As mentioned in chapter 2, if we apply one-point crossover to the TSP, the crossover may create meaningless strings which have overlapping genes. Consequently, the overlapping of the same number is prohibition on the string. Therefore, we apply cycle crossover and partial matched crossover (PMX). As mutation operators, we use an exchanging mutation and an inversion. The exchanging mutation exchanges two randomly selected genes. And the inversion reverses the order of a substring of a randomly selected string.

4.5.2.2 Simulation Results of Traveling Salesman Problem

We compare simulation results of the TSP among the SSGA, VEGA and standard GA (SGA) which is utilized a roulette wheel selection. As selection schemes, the VEGA and SSGA use 'delete least fitness' [71], namely, the individual with the least fitness value is removed from the population. However, the VEGA and SSGA remove the individual with the highest fitness value from the population, since the TSP is a minimization problem.

First, we show the simulation results of 25 cities plane TSP (Figure 4.17). The population sizes of the simulations are 100. The crossover probability is 0.8 and the mutation probability is 0.001 per gene. The virus population size is 10. The maximal infection rate and initial infection rate are 0.1 and 0.05, respectively. Figure 4.18 shows an example of the reverse transcription which performs like PMX.

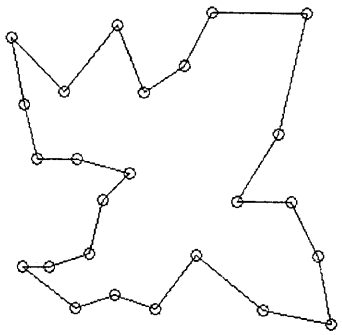


Figure 4.17 Typical solution of plane TSP with 25 cities

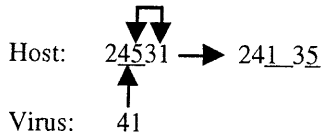


Figure 4.18 Reverse transcription for TSP

Figure 4.19 and Table 4.3 show a typical result and the average of 20 trials of the plane TSP and, respectively. In early generation, the VEGA is latest to reduce the fitness value, for there are little effective virus when a virus individual infects to a host individual. The reason is that a virus individual can not transduce effective schemata, since the host population have little effective schemata. In the last generation, the VEGA reduces the fitness value without trapping to local minima, since the virus population can transmit effective schemata in the host population. On the average, the VEGA obtains better solutions than others. Furthermore, we compared the average of computational time required for fulfilling the aspiration level (Table 4.3). These values of the lowest row in Table 4.3 are normalized by defining the average of the computational time of the VEGA as the value 1.0. The smaller the value is, the more quickly the genetic algorithm fulfills the aspiration level. The VEGA is superior to others in the comparison of the average of computational time.

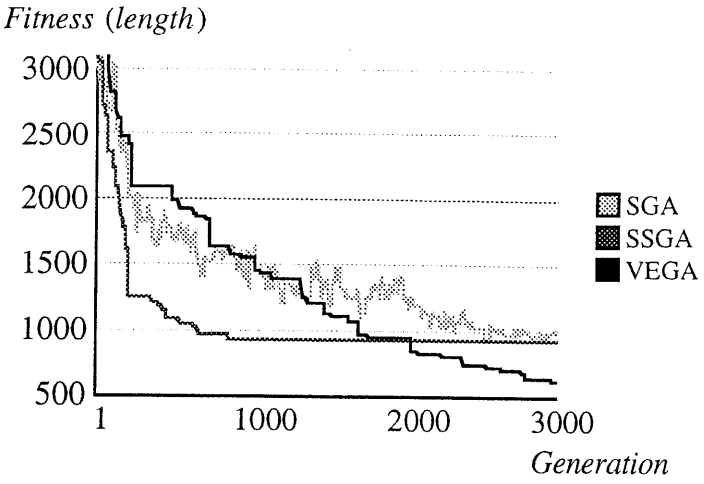


Figure 4.19 Typical simulation result of plane lattice TSP with 25 cities

Table 4.3 Simulation results of plane TSP

	SGA	SSGA	VEGA
Min_length	614	614	610
Mean_length	954	854	683
Max_length	1288	1020	749
Time	2.2	1.9	1.0

Next, we apply genetic algorithms to the 3 dimensional lattice TSP with 27 cities (Figure 4.20) and its optimal length is 700. Here, we compare the virus infection operator with other genetic operators. Figure 4.21 shows simulation results of the cycle crossover, PMX, inversion, exchanging mutation, and VEGA. VEGA performs only the virus infection operators in this simulation. In the simulation of the crossover, the exchanging mutation is also used because the genetic algorithm with only crossover converges to local optima soon. Fitness values in Figure 4.21 are the average of 50 trials. The VEGA attains global optima in all trails. The introduction of the virus infection improves the searching ability of genetic algorithms, since the virus population possesses schemata like subtours in the TSP. Furthermore, Figure 4.21 indicates that the inversion is a useful operator for solving the TSP and can perform the local search effectively.

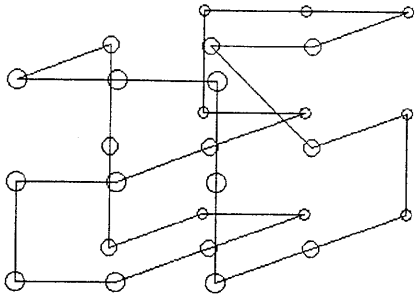


Figure 4.20 Typical solutions of 3D-lattice TSP with 27 cities

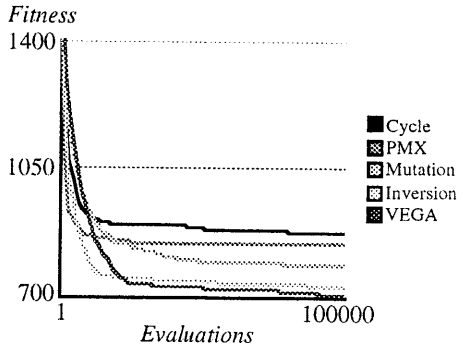
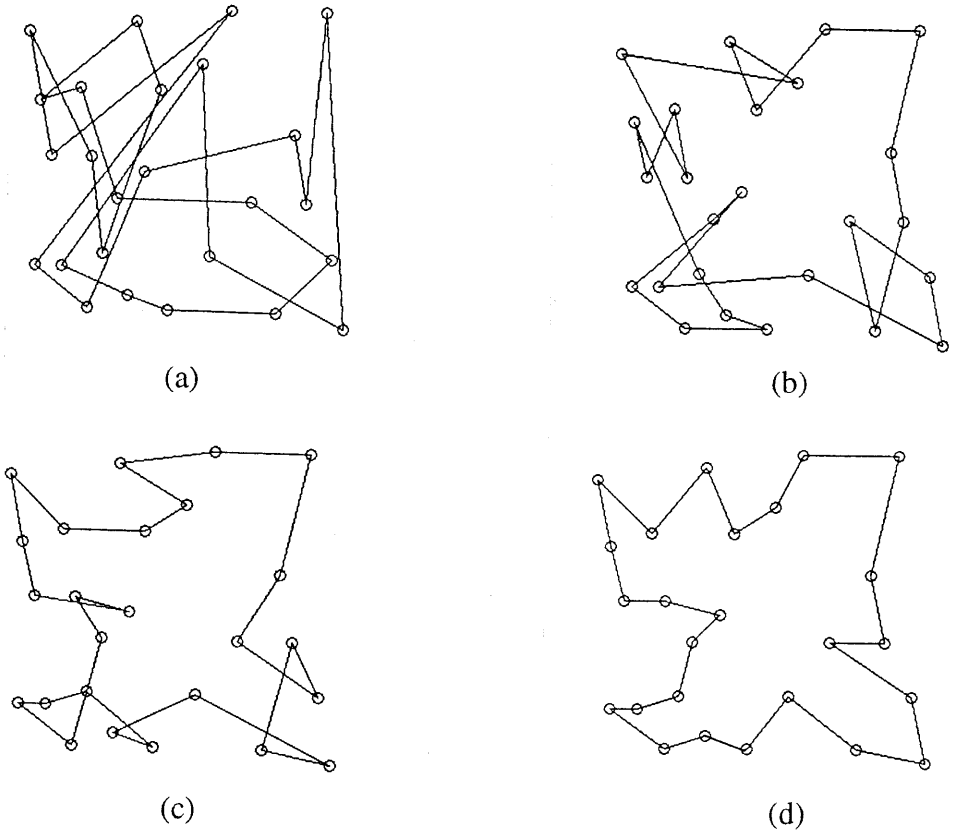


Figure 4.21 Simulation results of 3D-lattice TSP with 27 cities

4.5.2.3 Coevolution of Host and Virus Population

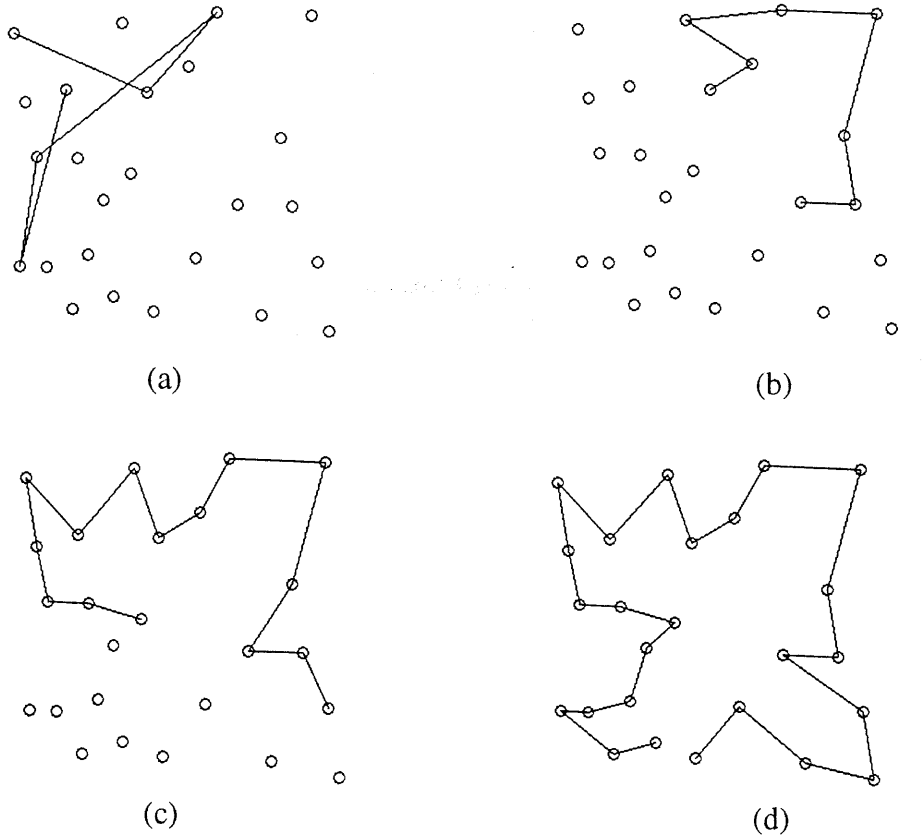
The above simulation results show that the VEGA attains the optimal solutions. We discuss whether VEGA transmits effective schemata or not by applying to the plane TSP with 25 cities. Figure 4.22 shows the evolutionary transitions of the host individual with the highest fitness value and Figure 4.23 shows that of the virus individual with the highest life force. In the early generation (a), the virus individual does not possess any schemata to seem to be effective. Then, the virus individual begins to be a substring of the host individual like an effective schema. The virus possesses a longer length substring generation by generation, and at last (d), the virus individual possesses the almost optimal solution.



(a) (b)

(c) (d)

Figure 4.22 Evolutionary transitions of the host individual



(a) (b)

(c) (d)

Figure 4.23 Evolutionary transitions of the virus individual

The host individual includes a subtour similar to the virus individual at each generation. This fact shows that the host population and the virus population have coevolved, since the virus individuals transduce from the host individuals and horizontally propagate their substrings into host individuals.

4.5.3 Application to Function Optimization Problem

Function optimization problems have been often applied as benchmark tests for genetic algorithms [12,66,83]. In this subsection, we apply the VEGA to two cases of function optimization problems. Case 1 is as follows:

$$f(x,y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\cos(4\pi y) + 0.7 \qquad (4.12)$$

where $-1.0 \le x, y \le 1.0$. This function is highly multimodal. The aim of this problem is to minimize the objective function. The global minimum is at x=0, y=0, which produces $f(x,y)$=0. Figure 4.24 shows the shape of this objective function.

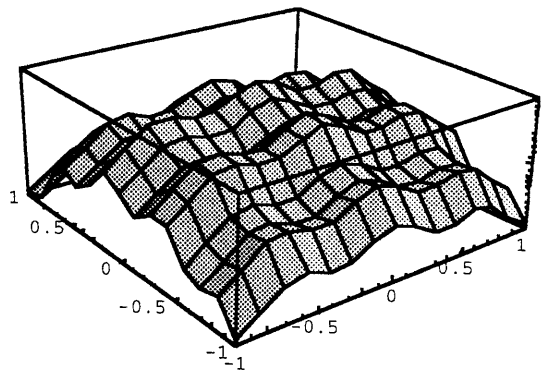


Figure 4.24 3D-space characterized by objective function $f(x,y)$

(The negative of $f(x,y)$ is depicted.)

We use the binary code as the representation of genotype. A string includes two variables in this objective function. Decoding the string, we obtain two integer numbers, and substituting the numbers into the X of following eq.(4.13), we obtain two variables.

$$z = \frac{X - Z}{Z} \tag{4.13}$$

Z denotes half the value of the maximal integer of X represented by the binary code. A uniform crossover and a bit mutation are used as genetic operators.

Figure 4.25 and Figure 4.26 show the simulation results of Case 1 of the SSGA and VEGA, respectively. The fitness value in each figure is the average of 50 trials. The convergence of the VEGA is faster than that of the SSGA, on the whole. Table 4.4 shows the average of fitness values obtained after 10000 evaluations. Since Case 1 is easy to solve, we apply to a more difficult function as Case 2,

$$g(x) = 5A + \sum_{i=1}^{5} (Ax_i - A\cos(2\pi x_i)) \tag{4.14}$$

Table 4.5 shows the average of fitness values after 20000 evaluations of 50 trials. When the host population size is 100, the VEGA attains the best value in these simulation results. Figure 4.27 shows the comparison of simulation results concerning virus population size. When the virus population size and the maximal infection rate are 10 and 0.1, respectively, the VEGA attains the best fitness value.

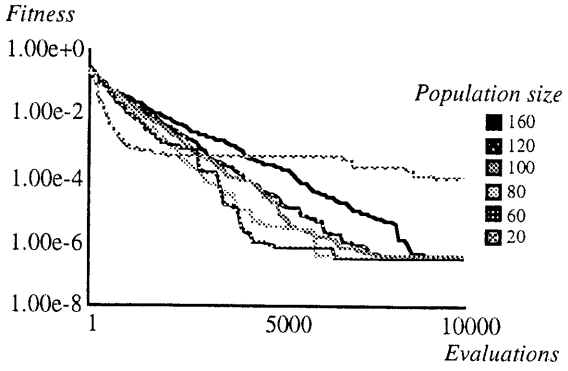


Figure 4.25 Simulation results of Case 1 (SSGA)

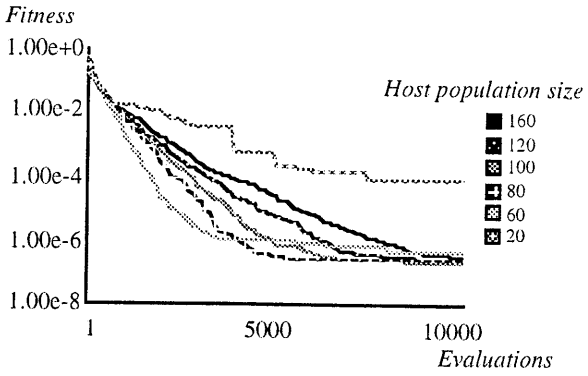


Figure 4.26 Simulation results of Case 1 (VEGA)

Table 4.4 Average of fitness value after 10000 evaluations in Case 1

Host pop. size	20	60	80	100	120	160
SSGA	1.00e-4	3.06e-7	3.42e-7	3.59e-7	4.04e-7	3.42e-7
VEGA	8.12e-5	4.24e-7	2.44e-7	2.12e-7	2.28e-7	3.23e-7

Table 4.5 Average of fitness value after 20000 evaluations in Case 2

Host pop. size	20	60	80	100	120	160
SSGA	3.71e-5	3.74e-5	4.46e-5	3.89e-5	3.72e-5	4.99e-5
VEGA	4.38e-5	3.70e-5	3.41e-5	2.65e-5	3.65e-5	1.51e-4

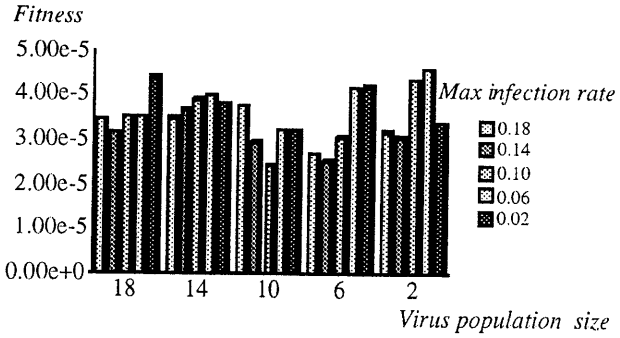


Figure 4.27 Comparison of simulation results concerning virus population size in Case 2

In general, it is known that one point or multi-point crossover is not effective in numerical optimization problems. Therefore, advanced genetic operators such as uniform intermediate crossover and self-adaptive mutation have been proposed for solving numerical optimization problems. In general, evolutionary programming outperforms to genetic algorithms in numerical optimization problems, but evolutionary programming does not use crossover operators. Therefore, we compare genetic algorithms introduced self-adaptive mutation with evolutionary programing. Evolutionary algorithms are based on the concept of species and evolutionary operators work on the phenotype of individuals, which is represented by real number, integer, and so on. The main operator of evolutionary algorithms is mutation. One of mutation operators is self-adaptive mutation as follows,

$$x_{j+n,i} = x_{j,i} + N(0, a_i \times \frac{fitness}{\max fitness} + b) \tag{4.15}$$

On the other hand, as mentioned in the section 4.4, we propose a virus-evolutionary algorithm (VEA). The VEA is composed of self-adaptive mutation, tournament selection and virus infection operators.

Here we apply evolutionary algorithms to two cases of numerical minimization problems as follows.

$$fitness = 0.1m + \sum_{i=1}^{m} (x_i^2 + 0.1\cos(3\pi x_i)) \tag{4.16}$$

$$fitness = 0.1m + \sum_{i=1}^{m-1} (x_i - x_{i+1})^2 + 0.1\sum_{i=1}^{m-1} \cos(3\pi(x_i - x_{i+1})) \tag{4.17}$$

where m is problem size (m = 50 in this case). In eq.(4.16), there is no association among variables and eq.(4.16) is easy to solve. To the contrary, eq.(4.17) has association among variables.

The individual is represented as variables, that is, each individual has m variables. The objective is to minimize fitness value. Figure 4.28 and Figure 4.29 show simulation results of eq.(4.16) and eq.(4.17), respectively. Fitness value in each figure denotes the average of 30 trials. EP_C in these figures denotes the evolutionary algorithm with elitist crossover. In the

elitist crossover, an individual selects a mating individual by using roulette wheel selection. In the EP_C, self-adaptive mutation is performed after the elitist crossover. From Figure 4.28, EP_C outperforms to others because good individuals have independent and good variables and elitist crossover can generate good individuals. These results indicate that the elitist crossover can inherit good genetic information from parents for the self-adaptive mutation. From Figure 4.29, the VEA outperforms to others. The convergence speed of the EP_C is the fastest in the early generation, but the VEA attains the best solution at the last generation. The reason is that the performance of the EP_C depends on the elitist crossover, the elitist crossover does not successfully work on the population at the last generation since the population in the last generation converge to local solutions and as the result, the elitist crossover results in failure. To the contrary, VEA can control virus infection rate and therefore the virus infection rate is very small in the last generation.

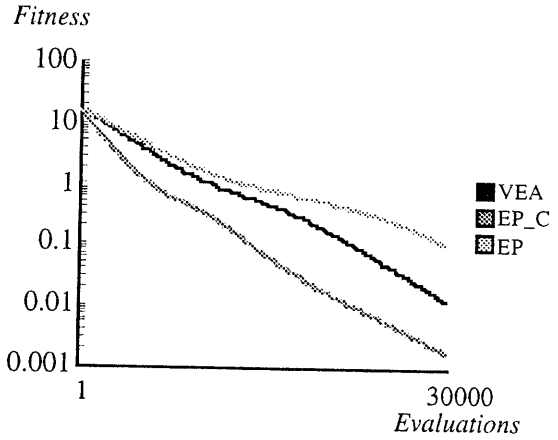


Figure 4.28 Simulation results of eq.(4.16)

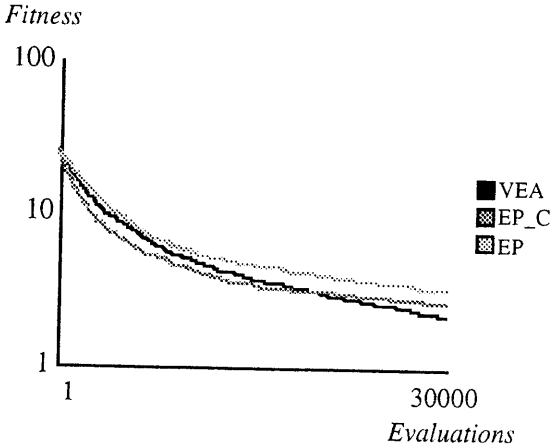


Figure 4.29 Simulation results of eq.(4.17)

4.6 Evolutionary Transition of Virus Evolutionary Genetic Algorithm

The previous section showed the effectiveness of the VEGA by applying conventional and traditional optimization problems. In general, the performance of algorithms depends on the relation among coding design, genetic operators, selection scheme, and control parameters. Especially, the coding design has a very close relation with the genetic operators. However, it is difficult to discuss these elements simultaneously. When considering the behavior of genetic algorithms, we often use Markov chain analysis [58~62]. Markov chain can completely model the behavior of a genetic algorithm when the size of the optimization problem is comparatively small, but it is physically impossible to model real large problems and also impossible to describe all evolutionary transitions of the large problems. Other approaches are to analyze the fitness correlation coefficient of a genetic operator [62] and to discuss the probability of improvement by genetic operators [61]. These approaches are comparatively simple, but the experimental results are very effective for evaluating the performance of genetic operators. In this section, we discuss the evolutionary transition by genetic operators based on these approaches.

4.6.1 Markov Chain Analysis and Fitness Landscape Analysis

Markov chain analysis is very important because its results often indicate good theories. A finite Markov chain is characterized by a finite set of states and a matrix of transition probabilities from state i to state j [5]. Let $p(t)$ be a population of candidate solutions at time step t. We can describe the behavior of a GA as the following state transition from $p(t)$ to $p(t+1)$,

$$p(t+1) = s(g(p(t))) \tag{4.18}$$

where $s(\bullet)$ and $g(\bullet)$ are intermediate transition matrices of the selection operations and genetic operators, respectively (Figure 4.30). The genetic operators explore the search space generated by all the population, while the selection operations exploit the search space for generating new candidate solutions. However, we can not describe all evolutionary transitions in the case of the large number of the problem size. In this section, we therefore focus on how much improvement, i.e. the difference between fitness values of parents and their children, Δf, and the probability of improvement by genetic operators, $P(\Delta f > 0)$.

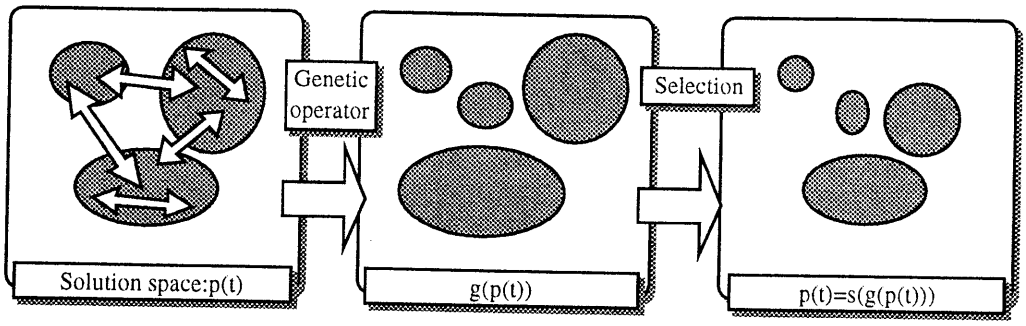


Figure 4.30 Evolutionary state transition of population by genetic operators and selection (p(t): population at generation t, $g(\bullet)$: genetic operators, $s(\bullet)$: selection operations)

On the other hand, Manderick [62] discussed a fitness landscape analysis by the following correlation coefficient of genetic operators,

$$\rho(fit_p, fit_c) = \frac{C(fit_p, fit_c)}{\sigma(fit_p)\sigma(fit_c)} \tag{4.19}$$

where fit_p and fit_c is fitness values of parent and its child, $C(fit_p, fit_c)$ is the covariance between the values fit_p and fit_c, and $\sigma(fit_p)$ and $\sigma(fit_c)$ are the standard deviations of the values fit_p and fit_c. This analysis demonstrates the relation between the fitness values of parents and children by genetic operators. However, the selection is not considered in Manderick's analysis, but the selection limits the search space. In fact, a population evolved by a set of genetic operations, is different from a population evolved by another set of genetic operations because of the difference in the generation method of candidate solutions. A population, in general, evolves through the interaction of the selection and genetic operators. In this section, we discuss evolutionary transitions concerning fitness values by genetic operators considering the role of selection. The component individuals of the population evolved by genetic operations are different from random generated individuals and a population varies from generation to generation. Therefore, a phase of evolution of a population (evolutionary phase) is represented using average fitness value of the component individuals. Consequently, we evaluate genetic operators on the population with average fitness value, $Ave\_fit(x)$, which is evolved by a set of particular genetic operations. Here x is defined as an evolutionary phase number. The procedure of simulation concerning fitness value is as follows.

Step 1 Initialization: generate randomly initial population, $p(0)$, and initialize evolutionary phase number; x=0.

Step 2 Evolution: make $p(x)$ evolve with a set of genetic operations; **G**, until the average fitness value of $p(x)$ is $Ave\_fit(x)$.

Step 3 Evaluation: perform each genetic operator to the individuals of $p(x)$, and evaluate the improvement from its parents 10000 times.

Step 4 x=x+1.

Step 5 go to step 2.

In step 3, the generated children are not replaced with their parents here in order to keep the simulation condition of a population evolved by particular genetic operations. In this way, a population gradually evolves by a set of genetic operations and we evaluate genetic operators on the limited search space of the population.

4.6.2 Evolutionary Transition in Knapsack Problem

This subsection discusses the performance of genetic operators for the knapsack problem in the previous section. The sets of genetic operations, **G**, are as follows,

G_1: SSGA (bit mutation)
G_2: SSGA (uniform crossover + bit mutation)
G_3: VEGA (virus infection + uniform crossover)

The population size is 100 and $Ave\_fit(x)$={1100, 1200, ···, 1700}. Figures 4.31~4.33 show simulation results of each 50 runs of the above experiments by initializing a population. We compare simulation results of (uniform) crossover, (bit) mutation and virus infection. $P_{improve}$ denotes the probability of improving fitness values of randomly selected parents in 500000 trials (10000 times * 50 runs). *Improvement* denotes the average of improvement when fitness value is improved by a genetic operator. *Expected improvement* denotes the product of $P_{improve}$ and *Improvement*. Here we don't consider negative improvement because we basically use the SSGA which eliminates the worst individuals. *Correlation coefficient* is the average of correlation coefficients calculated by eq.(4.19).

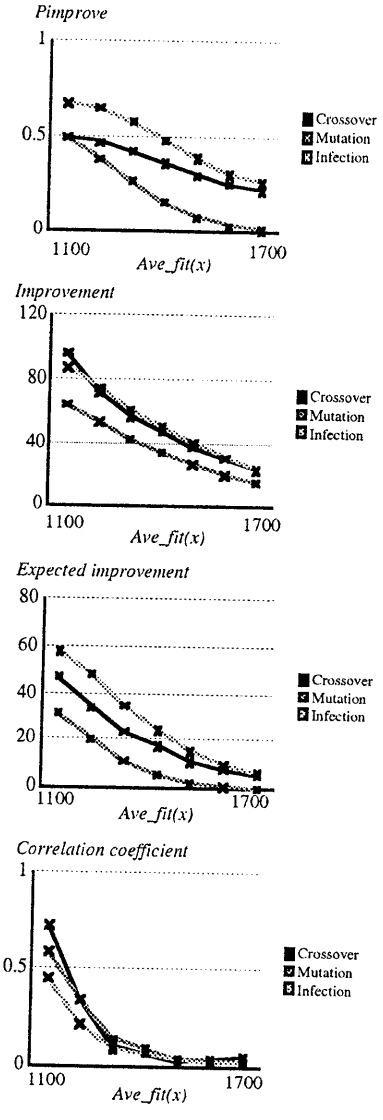


Figure 4.31 Results against the population evolved by G_1

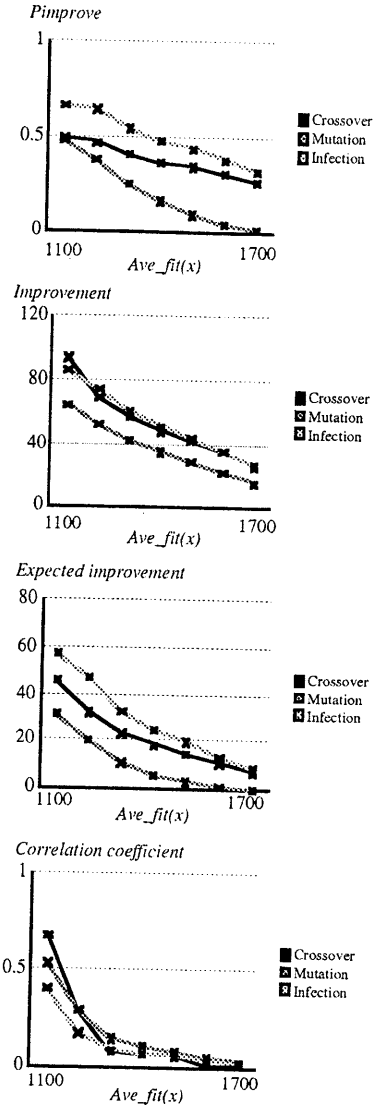


Figure 4.32 Results against the population evolved by G_2

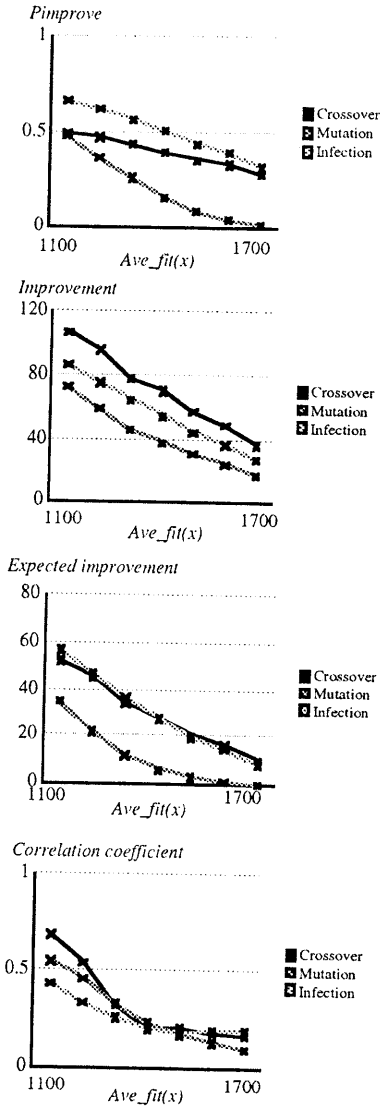


Figure 4.33 Results against the population evolved by G_3

From the simulation result on the whole, the value of each index in these figures decreases as the average of fitness values of the population (evolutionary phase number) increases. This is clear because the state transition probability of an individual to bad one is, in general, higher in the case of the population with the higher average of fitness values due to the stochastic search. However, it is desirable that the $P_{improve}$ by genetic operators is high even in the case of good individuals. The simulation results concerning $P_{improve}$ indicate that the virus infection succeeds well to improve the fitness values of candidate solutions with higher probability than others, since the successful viruses have high virus infection rate and overwrites their effective substrings on the strings of candidate solutions. Next, from the simulation results

concerning *Improvement*, the improvement by the crossover in the population evolved by G_3 is more than others, since the virus infection increases effective schemata in the population and the crossover combines these schemata. In addition, $P_{improve}$ and *Improvement* of crossover are better than those of mutation in all cases. This reason is inferred from the building block hypothesis [12], and the crossover generates new candidate solutions based on genetic information between individuals, not random exchanging. On the other hand, we could not obtain the good results concerning *Correlation coefficient*. This is probably due to the design of fitness function, especially, the design of penalty when the constraint is not satisfied. In the knapsack problem, much penalty is given to a candidate solution which does not satisfy the constraints, and therefore the correlation between fitness values is weak. This correlation is very important for GAs, but the design of fitness function for constraints is out of scope in this simulation.

4.6.3 Evolutionary Transition in Traveling Salesman Problem

This subsection discusses the performance of genetic operators for the TSP in the previous section. Since the TSP is a permutation optimization problem, genetic operators must generate feasible candidate solutions. The sets of genetic operations, **G**, for the TSP are defined as follows,

G_4: SSGA (exchanging mutation)
G_5: SSGA (cycle crossover + PMX)
G_6: VEGA (virus infection)

And the population size is 100 and $Ave\_fit(x)$={2800, 2600, ···, 1800}. We use a round tour length as fitness value in this simulation.

Figures 4.34~4.36 show simulation results of each 50 runs of the above experiments by initializing a population. We compare simulation results of genetic operators; cycle crossover, PMX, inversion, exchanging mutation, and virus infection. The indices in the figures are those of Figures 4.31~4.33. From the simulation results concerning $P_{improve}$, the improving probability of virus infection is only higher than 0.5 at first, and succeeds to improve fitness values with high probability. In addition, the improving probability of the exchanging mutation is the lowest in the all cases since the exchanging mutation is easy to break subtours in the TSP. In the population evolved by G_5, the cycle crossover and PMX succeed to improve

fitness value with high probability through all evolutionary phases and this result indicates that the population evolved with crossover operators is robust to crossover operators. This reason is also inferred from the building block hypothesis [12] and the crossover generates new candidate solutions based on the subtours between individuals. In the population evolved by G_6, *Improvement* of the crossover operators are higher than those of G_4 and G_5, and this result indicates that the population evolved by the virus infection has many effective schemata and successful crossover operators can easily obtain high improvement.

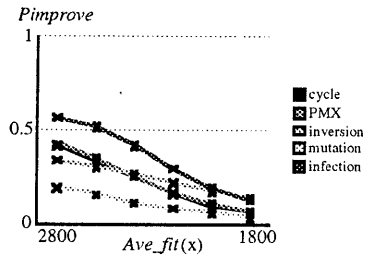


Figure 4.34 Results against population evolved by Case 1

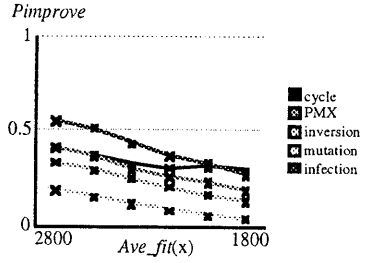


Figure 4.35 Results against population evolved by Case 2

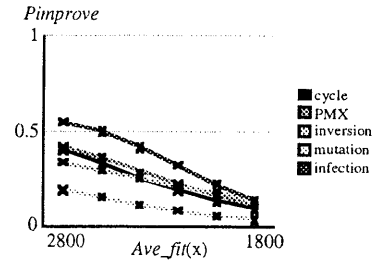
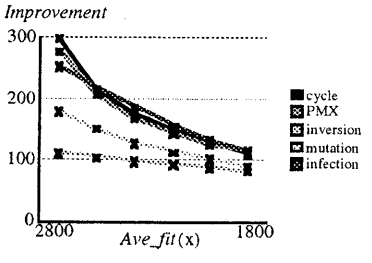
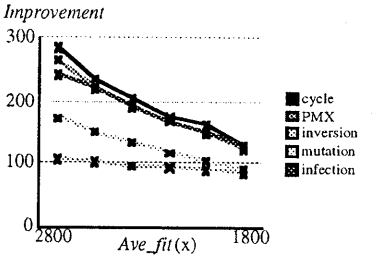
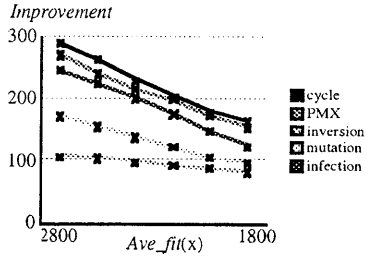
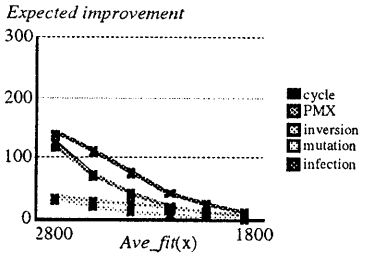


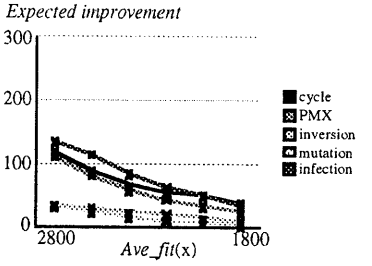
Figure 4.36 Results against population evolved by Case 3

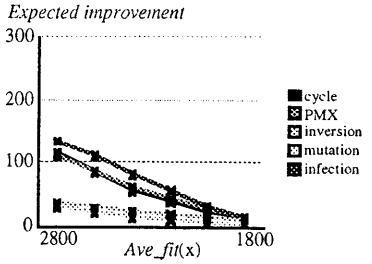












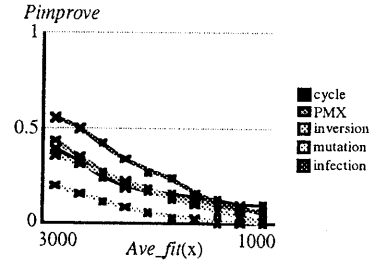
In this simulation, a population evolves with a single type of operator, and therefore a population are easily trapped to local minima. Next, we conduct experiments by using the following sets of genetic operators,

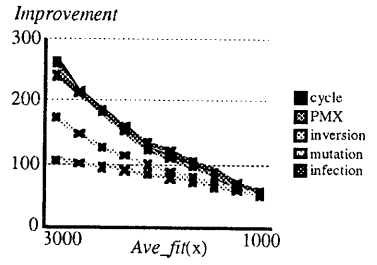
G_7: SSGA (inversion + exchanging mutation)

G_8: SSGA (cycle crossover + PMX + inversion + exchanging mutation)

G_9: VEGA (virus infection + cycle crossover + PMX)

Each set of genetic operators is often used for solving the TSP. And the population size is 100 and $Ave\_fit(x)$={3000, 2600, ···, 1000}.





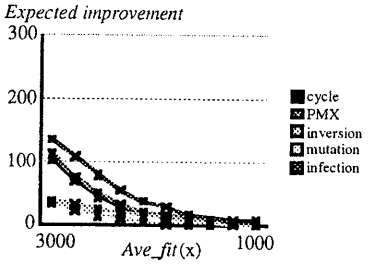
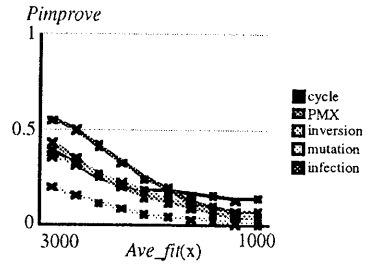
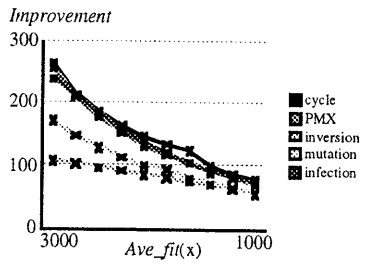


Figure 4.37 Results against population evolved byG_7





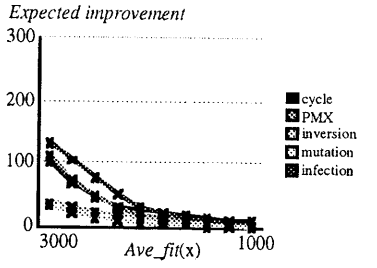
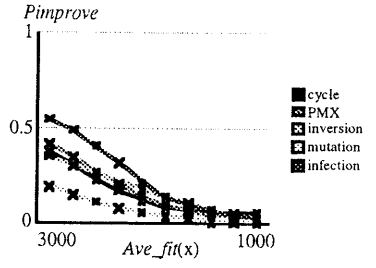
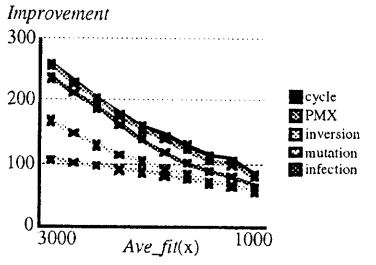


Figure 4.38 Results against population evolved by G_7





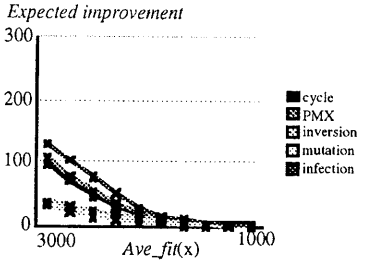


Figure 4.39 Results against population evolved by G_7

Figures 4.37~4.39 show simulation results of each 50 runs of the above experiments by initializing a population. The tendencies of G_7 ~ G_9 are relatively similar to those of G_4 ~ G_6, respectively. This indicates that the population evolved by multi-operators also inherits the fundamental feature of the main genetic operator. In the population evolved by G_9, the correlation coefficients of the inversion through all evolutionary phases are relatively high.

The reason would be as follows. The population includes many individuals which can be improved by the inversion since G_6 does not include the inversion. In fact, we apply the VEGA with inversion to the TSP, and Figure 4.40 shows the simulation result of the VEGA with inversion (VEGA* in Figure 4.40). This simulation result indicates that the introduction of the inversion causes the improvement of the performance. Furthermore, the correlation coefficients of the mutation and inversion are relatively high at first. This is due to the design of fitness function and small change by the mutation and inversion as the local search. The mutation and inversion generate new individuals as local searches and the distance concerning genotype between parents and children is comparatively short. On the other hand, the crossover can generate feasible children which partially inherit the stings of their parents as the global search, but it is possible that the crossover may generate children far away from their parents. Consequently, the correlation coefficients of crossover operators concerning fitness values can be low.

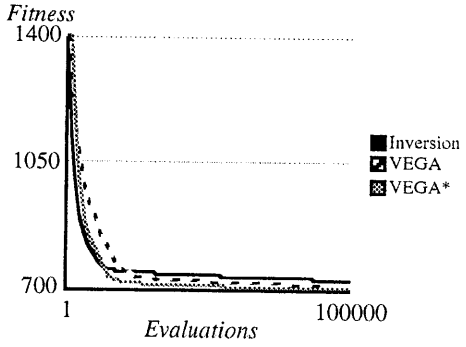


Figure 4.40 Simulation results of VEGA with inversion

To conclude, the virus infection has comparatively high probability for improving fitness value at evolutionary phases of populations evolved with any genetic operators and its improvement is also comparatively high. This indicates that the virus infection operators can generate effective schemata and propagate them well. Furthermore, these simulation results indicate that particular genetic operators give a population their peculiar potentiality of generating candidate solutions, that is, genetic operators determine the direction of evolution. In addition, the performance of a genetic operator depends on the evolutionary phase of a population. However, these simulation results indicate that a genetic operator can improve fitness values of individuals effectively, but do not indicate that the genetic operator can generate optimal solutions. In fact, the GAs have selection operation which selects individuals from the current population according to fitness values, and genetic operators generate new

individuals from the selected individuals. Consequently, the selection pressure and generational model also influence the performance of the GA. The behavior of any evolutionary computation can not be anticipated by analyzing its components [61].

4.7 Summary

This chapter proposes two types of evolutionary optimization methods based on virus theory of evolution; virus evolutionary genetic algorithm (VEGA) and virus evolutionary algorithm (VEA). Furthermore, this chapter discuss the role of virus infection in the VEGA. The VEGA has two features of horizontal propagation and vertical inheritance of genetic information. The essential of the VEGA lies in the virus infection operators which perform a complex hill-climbing search in the long run. The reverse transcription plays the role of a crossover and a selection simultaneously. The VEGA searches the solution space with reverse transcription operator by generating new candidate solutions with overwriting host individuals partially. The virus infection is similar to a proportional selection scheme since a virus individual performs the reverse transcription with the frequency of virus infection rate. However, the VEGA differs from other genetic algorithms in the generation of new individuals. The reverse transcription directly generates a candidate solution with overwriting its substring on host individuals, though a crossover generates new candidate solutions with randomly combining substrings in a standard genetic algorithm. Therefore, the VEGA can generate new candidate solutions with a directionality in the search.

The VEGA simulates coevolution of a virus population and a host population. A virus individual evolves by transducing from the infected host population. Therefore, the best parameters concerning virus infection operators exist and the convergence of the host population depends on the frequency of the virus infection operators.

Furthermore, we discuss evolutionary transitions by the virus infection in the VEGA. The simulation results indicate two points. The first one is that genetic operators determine the direction of evolution, especially, a population evolved with crossover operators is robust to crossover operators. In addition, the performance of a genetic operator depends on the evolutionary phase of a population. The other is that virus infection operators can generate effective schemata and propagate them to a population evolved with any genetic operators. However, these simulation results do not indicate that genetic operators can generate optimal solutions.

Chapter 5 Application to Engineering Optimization Problems

Evolutionary optimization methods have been successfully applied to various fields of engineering such as robotics, manufacturing system and mechatronics. The roles of the evolutionary optimization methods in engineering are as follows.

1. Design of hardware and software.
2. Tuning of control parameters
3. Modeling and planning in software level

The design and parameter tuning are to determine the performance of hardware and software, and the modeling is to build a model based on the results of the cause-effect analysis of the problem, and the planning is to determine how to perform a given task and to solve a problem.

This chapter applies evolutionary optimization methods to some engineering optimization problems. First, we present trajectory planning for redundant manipulators. The form of the reconfigurable redundant manipulator is dynamically reconfigured according to its environment and given tasks. Virus-evolutionary genetic algorithm; VEGA, is applied to a trajectory planning problem based only on forward kinematics. The simulation results of trajectory planning show that the VEGA can generate a collision-free trajectory. Second, we present a pallet allocation problem as an example of self-organizing manufacturing systems, in that a process effectively self-organizes according to other processes. The simulation results show the effectiveness of the proposed problem. Finally, we present a self-tuning fuzzy controller as an example of a hybrid method with other soft computing. The VEGA is applied to the tuning of fuzzy controller with radial basis function for a Cart-Pole system.

5.1 Application to Trajectory Planning for Redundant Manipulator

Recently, robots have been seen in various fields. In general, robots can be divided into mobile robots and arm robots (robot manipulator). The main aim of the mobile robot is to

carry materials, products, tools and others, while the aim of the robot manipulator is to handle them. Lately, the mobile robots with manipulators have been developed for improving the performance and flexibility.

A robot receives a task from a human operator and performs the task in the workspace including a lot of obstacles such as human bodies, machining centers and other robots. The robot should therefore take into account the collision avoidance with these obstacles. Furthermore, the robot should automatically generate its motion for performing the task without human assistance. Consequently, the problems for performing given tasks are fundamentally path planning problems, trajectory planning problems and task planning problems (Figure 5.1). Here we define these planning problems as follows. The path planning problem is to generate a collision-free path of the robot from a given starting point to goal points, which satisfies the spatial constraints. The trajectory planning problem is to generate a trajectory of the robot satisfying the time constraints. The task planning problem is to decide a sequence of primitive motion commands for solving a given task. In fact, each definition of these problems is conceptually differentiated from other planning problems and therefore each planning problem shares the some elements of other planning problems. This section focuses on trajectory planning of redundant manipulators to a given task.

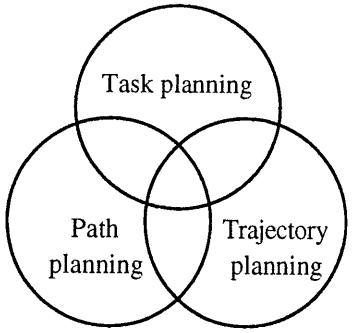


Figure 5.1 Motion planning of robot for performing a given task

Various kinds of approaches for trajectory planning and collision avoidance planning have been proposed [84~87]. Lozano-Perez proposed the V-graph algorithm for finding the shortest path which does not collide with any obstacles in the workspace of a robot manipulator [84]. Brooks developed a method for finding the free passageway called "Freeway" [85]. Recently, new approaches have been proposed [87].

Redundant degrees of freedom enable a manipulator to perform various works where a workspace includes a number of obstacles. However, it is difficult to solve collision avoidance problems in such cases. To solve these problems, we propose a hierarchical trajectory planning method for a reconfigurable redundant manipulator. The hierarchical trajectory planning generates some intermediate positions of the reconfigurable redundant manipulator. Combining these intermediate positions, the hierarchical planning generates a collision-free trajectory. In this section, we apply the VEGA to the trajectory planning.

5.1.1 Self-Organizing Manipulator System

A cellular manipulator system is composed of a large number of tools and parts which are called cells. The form of a cellular manipulator system is dynamically reconfigured according to its environment and given tasks (Figure 5.2). However, it is difficult to generate a trajectory to achieve the given task in a workspace including a number of obstacles. In inverse kinematics using Jacobian matrix, problems such as singularity avoidance must be took into account. One of trajectory planning methods without Jacobian is a trajectory planning method using genetic algorithm. The trajectory planning uses only a forward kinematics concerning the redundant manipulator.

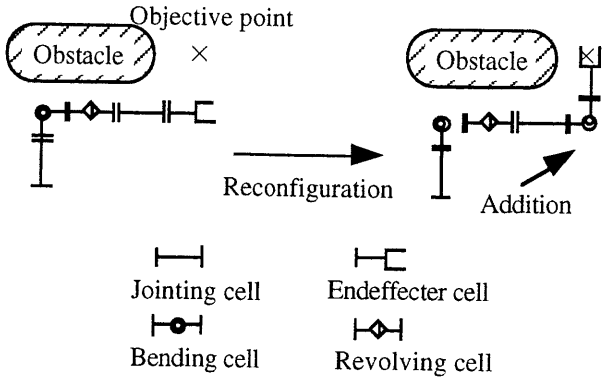


Figure 5.2 Reconfiguration example of cellular manipulator system

5.1.2 Hierarchical Trajectory Planning

The trajectory planning problems partially overlap the path planning problems. For example, a trajectory planning problem for a robot manipulator often includes a path planning problem of the end-effector from an initial position to a final position. In general, forward kinematics maps the joint angle space of the robot manipulator into the Cartesian space, while inverse kinematic maps the Cartesian space into joint angle space. The trajectory planning problem is to solve the inverse kinematics of the robot manipulator. It is generally difficult to represent a robot manipulator as a point because of some degrees of freedom (DOF), though a mobile robot can be regarded as a point in the work space. Furthermore, the trajectory planning problem is more difficult as the DOF increases, since the number of configurations of the robot manipulator corresponding to a point in the Cartesian work space increases (Figure 5.3).

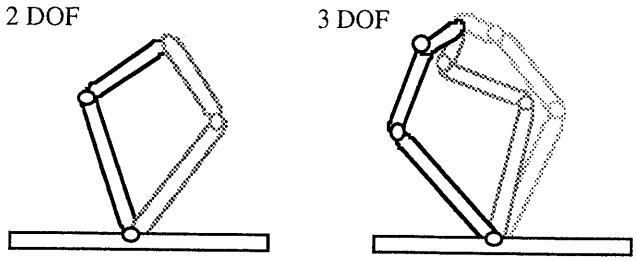


Figure 5.3 Configurations corresponding to a point in the Cartesian space

In this section, we consider trajectory planning and collision avoidance planning of a redundant manipulator made up of only revolving and bending joints in the workspace which includes some static obstacles (Figure 5.4). Various method for finding a collision-free trajectory of a redundant manipulator have been proposed so far. One of them is a trajectory planning method by connecting collision-free intermediate positions of a manipulator from an initial position to a final position. In general, these intermediate positions are generated step by step. And the method has a problem that some deadlocks may occur. A trajectory planning of a redundant manipulator is closely related to its collision avoidance planning. Therefore, we propose a trajectory planning method in the case that initial and final positions are only given.

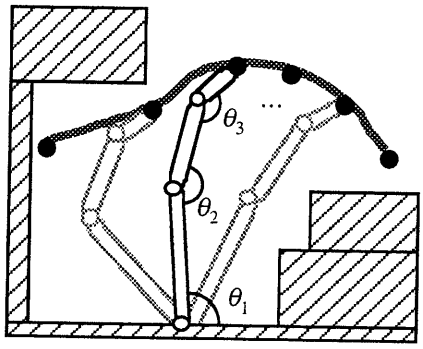


Figure 5.4 Trajectory planning in the work space including some static obstacles

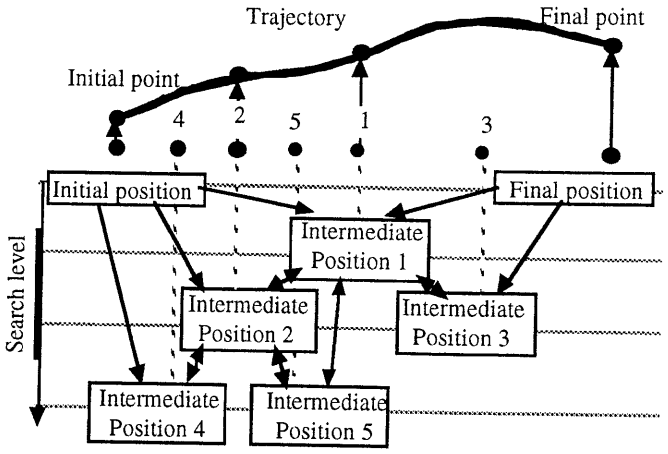


Figure 5.5 Trajectory planning and collision avoidance planning

We propose a top-down approach of trajectory planning based on initial and final positions (Figure 5.5). Here a position is expressed by a set of joint angles. First, the first intermediate position of the manipulator is generated based on the initial and final positions which are given. As a next search level, the second intermediate position is generated based on the initial and the first intermediate positions. The intermediate positions into the lower level are generated between two positions, if the trajectory connecting these two positions can not avoid the obstacles (Figure 5.6). In this way, collision-free intermediate positions are hierarchically generated. Last, applying a spline interpolation method to the joint angles of these intermediate positions, we obtain a collision-free trajectory.

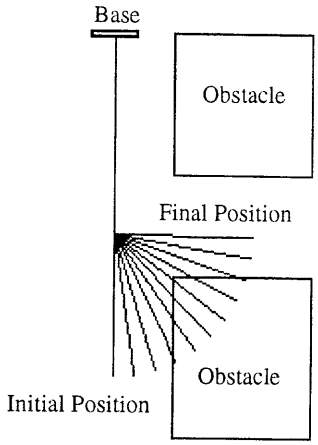


Figure 5.6 Collision check between manipulator and obstacles

An appropriate intermediate position for the collision avoidance is generated by VEGA which optimizes a objective function based on the distance between the manipulator and obstacles. To measure the distance between the manipulator and obstacles, we apply the concept of pseudo-potential [86]. We assume that the workspace is divided into N*N*N cells. Furthermore, let us assign a pseudo-potential value p to each cell to evaluate its closeness to the obstacles in the workspace. A pseudo-potential value of a cell can only take positive integer or zero. The closer the distance from the obstacles is, the larger the pseudo-potential value is. A cell with the pseudo-potential value zero is comparatively far away from obstacles.

We apply VEGA to the generation of intermediate positions of a reconfigurable redundant manipulator. As mentioned before, an intermediate position is hierarchically generated. Assume two positions, p_A and p_B be already generated. We consider a generation of an intermediate position p_C between position p_A and p_B into a lower level. The variables to be optimized are joint variables of the redundant manipulator. The objective is to generate a trajectory realizing minimum distance from the initial point to the final point and farther from the obstacles. To achieve the objective, we use the following fitness function,

$$fitness = w_1 f_p + w_2 f_d + w_3 f_{fr} + w_4 \max_{pot}{}^2 + w_5 \mathrm{sum}_{pot} \qquad (5.1)$$

where w_1, ···, w_5 are weight coefficients. The first term, f_p, in eq.(5.1) denotes the sum of squares of the distance between the manipulator's end of p_A and that of p_C, and the distance between that of p_B and of p_C, that is defined as:

$$f_p = (p_{A\_end} - p_{C\_end})^2 + (p_{B\_end} - p_{C\_end})^2 \tag{5.2}$$

where p_{X_end} is the end of a position p_X. The second term f_d in eq.(5.1) denotes the sum of squares of the difference between the each joint angle of p_A and that of p_C, and the difference between that of p_B and that of p_C, that is defined as:

$$f_d = \sum_{i=1}^{n} \{(\theta_{A\_i} - \theta_{C\_i})^2 + (\theta_{B\_i} - \theta_{C\_i})^2\} \tag{5.3}$$

where θ_{X_i} is the i-th joint variable of a position P_X. The third term, f_{fr}, in eq.(5.1) denotes the sum of the evaluation function using a normal distribution(Figure 5.7) to make each joint be within an available range, which is defined as:

$$f_{fr} = n - \sum_{i=1}^{n} fr_i \tag{5.4}$$

where f_{ri} is the evaluation value of a joint i. To evaluate the distance between the manipulator and the obstacles, we introduced the concept of pseudo-potential space, as mentioned before. Let us take some sampling points on the manipulator to measure the distance between the manipulator and obstacles. The fourth term, max_{pot} in eq.(5.1) denotes the maximum value in the pseudo-potential values on the sampling points. If the max_{pot} is the maximal pseudo-potential value p, the manipulator collides with obstacles. The last term, sum_{pot} in eq.(5.1) denotes the sum of pseudo-potential values on all sampling points, which is defined as:

$$\mathrm{sum}_{pot} = \sum_{i \in SP} pot_i \tag{5.5}$$

where SP is a set of all sampling points and pot_i is the pseudo-potential value of an element i included in SP. Using these terms, we solve the multi-objective optimization problem of collision avoidance. Therefore, the objective of the trajectory planning is to obtain intermediate positions to minimize the fitness value.

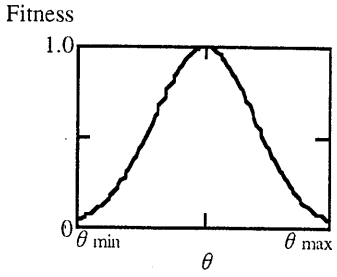


Figure 5.7 Evaluation function using normal distribution

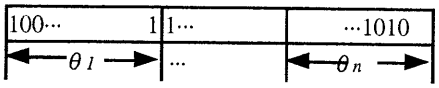


Figure 5.8 The representation of joint angle in VEGA

Intermediate positions are generated by the VEGA in the top-down approach of trajectory planning. To apply the VEGA, we need to determine the coding method into the string space first of all. Each joint variable θ_i is represented as binary coding (Figure 5.8). A string (individual) includes all joint variables. Decoding the string, we obtain an integer number x_i. Inserting the x_i into eq.(5.6), we obtain each joint angle.

$$\theta_i = \theta_{\min\_i} + \theta_{rng\_i} \cdot \frac{x_i}{X} \tag{5.6}$$

θ_{min_i} is the lower bound of the joint angle θ_i, θ_{rng_i} is the movable range of the i-th joint, and X is the maximal value obtained by decoding into integer. For example, if the length of a bit string per joint variable is 10, then $X = 2^{10} = 1024$. Consequently, this optimization problem results in a 0-1 combinatorial optimization problem.

As mentioned before, a virus individual can transmit a substring between host individuals, that is, the virus has two virus infection operators: the reverse transcription and the transduction. A substring consists of three characters {0,1,*} and is the same length as a host individual. The character '*' denotes 'don't care mark'. A virus individual does not carry out the reverse transcription in the same position where there is '*'. In this case, the length of a virus is constant, but the order of a virus is variable. Figure 5.9 shows an example of the reverse transcription. The reverse transcription overwrites the virus' substring on a randomly selected host individual. The transduction has two types of operators(Figure 5.10). One is to copy

genes from a host individual with a copy rate per gene. The other is to replace some genes with character '*' with a cut rate per gene. If the virus improves the fitness of host individual, the copy operator is carried out with the transduction rate. If not, the replacement operator is carried out. An initial virus population is generated from a host population using the transduction operator.

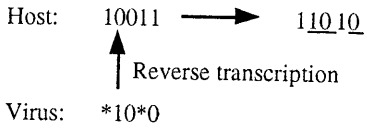


Figure 5.9 Reverse transcription operator

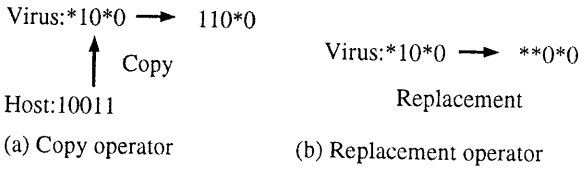


Figure 5.10 Transduction operator

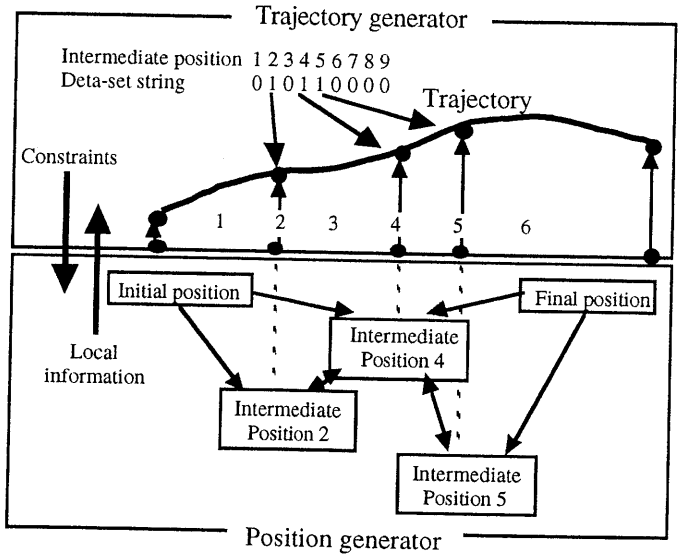


Figure 5.11 Hierarchical trajectory planning

The top-down approach for trajectory planning does not always obtain an optimal trajectory, since the top-down approach does not have any global evaluation function about the trajectory. To obtain an optimal trajectory, we propose a hierarchical trajectory planning method which is composed of two layers: a trajectory generator and a position generator (Figure 5.11).

The position generator generates some intermediate positions of the redundant manipulator between the given initial and final positions. All intermediate positions are generated based on their before and after positions simultaneously by using the intermediate position generator in the top-down approach. An intermediate position satisfied the aspiration level in VEGA is sent to the string on some individuals of the trajectory generator.

The trajectory generator generates a collision-free trajectory combining some intermediate positions generated in the position generator. We also apply the VEGA to the trajectory generator. VEGA in the trajectory generator has a set of candidate solutions including joint angles. Figure 5.12 shows a deta-set string for intermediate positions and a set of joint angles of some intermediate positions. The VEGA initially sets zero on all deta-set strings. Each individual gradually evolve by receiving a set of joint angles of intermediate position from the position generator. A mutation operator changes a randomly selected gene on the deta-set string to 0. A virus individual has a subset of intermediate positions and transmits among the host individuals (candidate solutions).

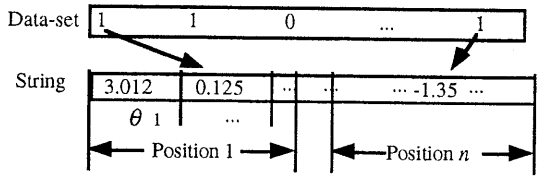


Figure 5.12 Coding in trajectory generator

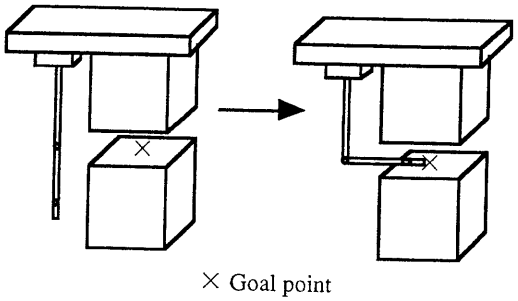
The intermediate positions of the best individual are the constraint for generating intermediate positions, that is, the position generator generates other intermediate positions based on the intermediate positions of the best individual in trajectory generator. Therefore, the hierarchical trajectory planning results in a co-optimization problem of a trajectory and intermediate positions.

5.1.3 Simulation Results

This section presents some numerical simulation of the trajectory planning. We consider the trajectory planning problem of three types of redundant manipulators in the workspace where there are some obstacles (Figure 5.13, Figure 5.14, Figure 5.15). The reconfigurable redundant manipulators have 7 degrees of freedom (DOF) similar to a human arm and 10 DOF, respectively.

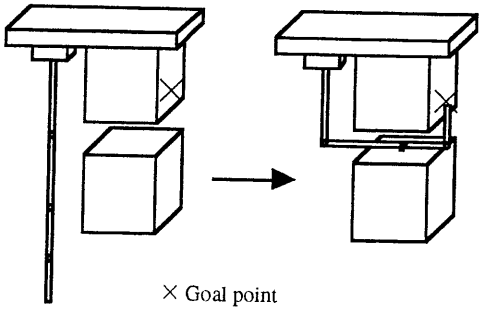
The objective is to generate a collision-free trajectory from an initial position to a final position. The workspace is divided into 30*30*30 cells. The pseudo-potential space has a maximum 5.

We compare VEGA with SSGA and EP. Table 5.1 shows the parameter of the SSGA, VEGA and the EP. Table 5.2 shows the parameters of the virus infection and Table 5.3 shows the link parameters of the 7 DOF manipulator.



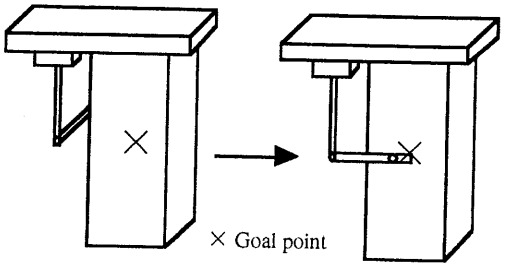
(a) Initial position (b) Final position

Figure 5.13 Simulation example (case A) of 7 DOF manipulator



(a) Initial position (b) Final position

Figure 5.14 Simulation example (case B) of 10 DOF manipulator



(a) Initial position (b) Final position

Figure 5.15 Simulation example (case C) of 7 DOF manipulator

Table 5.1 Parameters of the VEGA, the EP

	SSGA, VEGA	EP
Population size	100	100
String length	70	7variables
Crossover rate	1.0	-
Mutation rate	0.006	1.0
Opponent	-	10
Generation	300	300

Table 5.2 Parameters of the virus infection

Virus population size	20
Life reduction rate(r)	0.9
Initial infection rate	0.2
Max infection times	5
Transduction rate	0.6
Copy /Cut rate	0.05

Table 5.3 Link parameters of 7 DOF manipulator

i	1	2	3	4	5	6	7
α_i	-90	90	90	90	-90	-90	90
ai	0	0	0	0	0	0	0
di	0	0	10	0	6	0	1
$\theta i\_initial$	0	90	90	0	0	0	0
θi_final	0	90	90	-90	90	0	0
θmin	-180	-90	-180	-120	-180	-140	-5
θmax	180	135	180	120	180	140	140

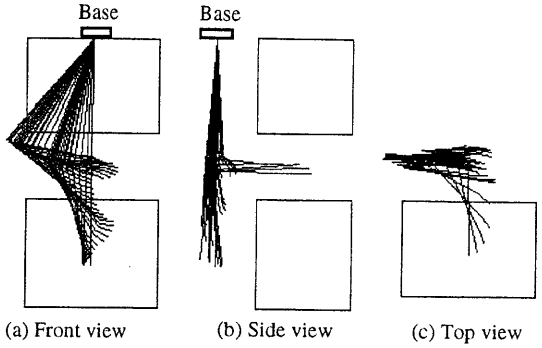


Figure 5.16 Collision-free trajectory in example A

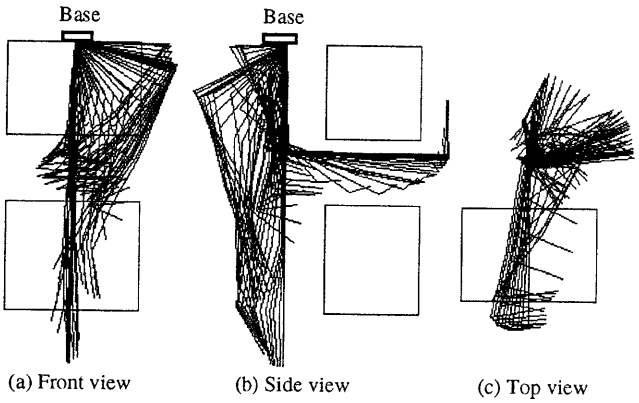


Figure 5.17 Collision-free trajectory in example B

Figure 5.16 and Figure 5.17 show simulation results of the top-down trajectory planning of the case A and case B by using the VEGA, respectively. In these figures, the side view shows that each redundant manipulator avoids colliding with the obstacles. Each redundant manipulator achieves the final position without colliding with obstacles and the obtained trajectories are farther away from obstacles. Furthermore, the generated trajectory in Figure 5.16 seems similar to the motion of a human arm (left arm in this case). Furthermore, Figure 5.18 shows the joint angles obtained by the top-down planning with the VEGA.

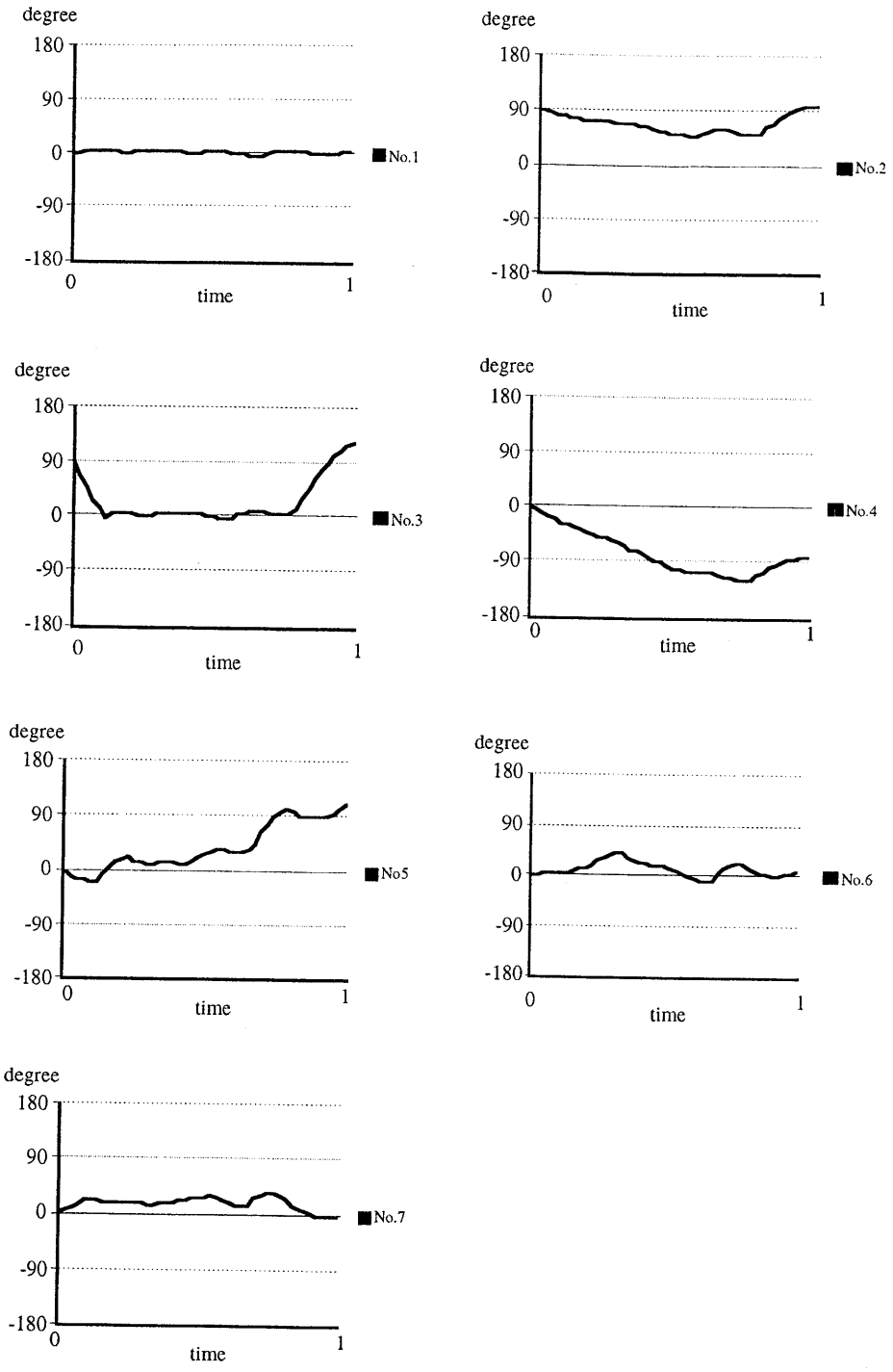
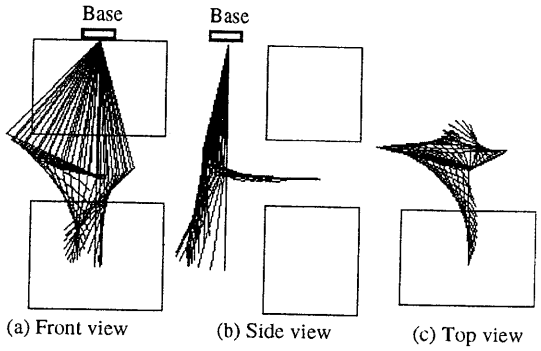
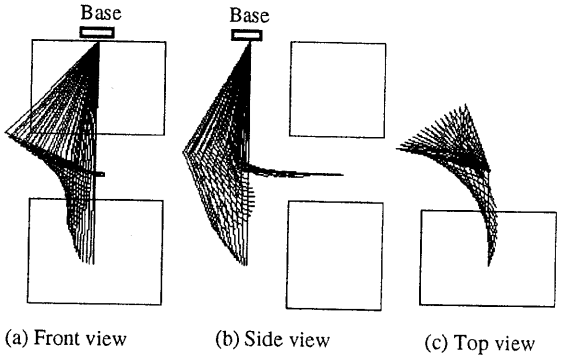


Figure 5.18 Joint angles obtained by top-down planning with VEGA

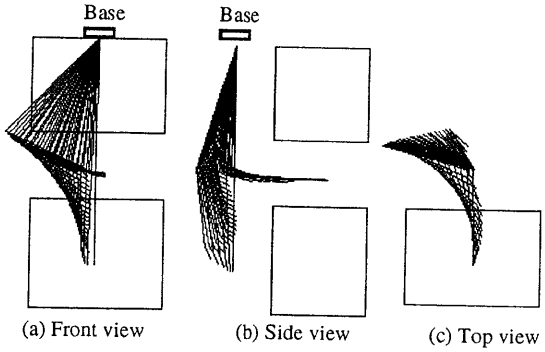
Figure 5.19 shows the simulation result of hierarchical trajectory planning of the simulation example A. No.1~3 in Figure 5.19 show the evolutionary transitions of a trajectory generated by the VEGA. At the early generation (No.1), though the hierarchical trajectory planning obtains a collision-free trajectory, the combination of the intermediate positions is not optimal. Finally (No.3), the hierarchical planning obtains the best trajectory.



No.1 (generation 50)



No.2 (generation 75)



No.3 (generation 100)

Figure 5.19 Evolutionary transition of collision-free trajectory generated by hierarchical trajectory planning with VEGA

The advantage of the hierarchical trajectory planning is to find better solution than the top-down approach quickly, since the hierarchical trajectory planning can search all the solution space simultaneously. Figure 5.20 and Figure 5.21 show simulation results of case C by the top-down approach and hierarchical approach, respectively. In case C, the tow-down approach is difficult to generate the first intermediate position between the given initial and final positions, since it is difficult that feasible intermediate positions without colliding with the obstacles in the workspace is generated to satisfy the minimization of f_p and f_d in fitness function eq.(5.1) simultaneously. On the other hand, the hierarchical trajectory planning is easy to generate a collision-free trajectory because of the simultaneous generation of some intermediate positions.

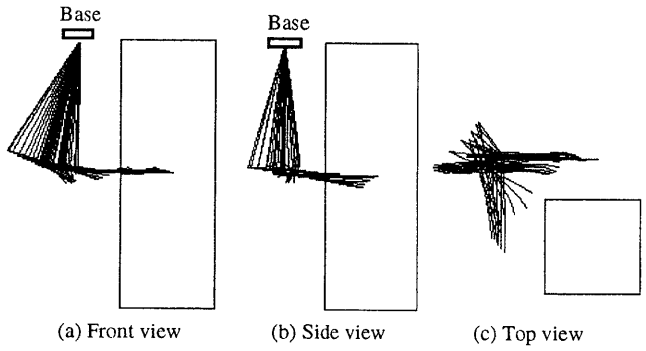


Figure 5.20 Simulation result of case C by the top-down trajectory planning

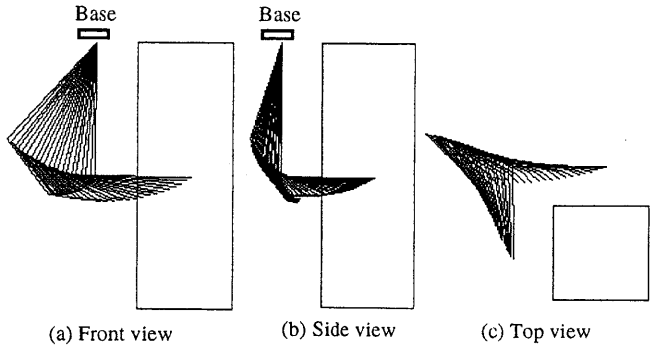


Figure 5.21 Simulation result of case C by the hierarchical trajectory planning

And a large population size would obtain higher performance. However, the trade-off between the quick search and the performance of the solution exists. Furthermore, it is difficult to design and to evaluate a multi-objective fitness function itself. Here we focus on the quick search with the aspiration level. Therefore, the hierarchical trajectory planning realizes the quick generation of a trajectory based on only forward kinematics. In fact, the hierarchical trajectory planning can generate a trajectory satisfied the aspiration level about three times as quickly as the top-down trajectory planning in the average.

Furthermore, we compare the VEGA with the SSGA and EP. The objective of this simulation is to obtain the first intermediate position between the initial and final positions in the top-down approach. In the EP, an individual has real variables of joint angles. Figure 5.22 shows the comparison of simulation results of the VEGA, SSGA and EP. The fitness values in Figure 5.22 are average values of 30 trials. At first, the EP outperforms others since the virus population in VEGA does not have effective schemata. Therefore, the virus population can not effectively carry out reverse transcriptions. The virus population gradually become to have effective schemata and propagate them between host individuals. At last, the VEGA obtains best solution than EP. One reason of this fact is as follows. Since the problem space is approximated to a combinatorial space in the VEGA, the solution space of the VEGA is smaller than the real problem space. In addition, the VEGA finds a solution satisfied the aspiration level more quickly than the EP concerning computational time. However, the EP can obtain the optimal solution in numerical optimization problems by tuning mutation's parameters. To summarize, the VEGA is effective in the case of obtaining better solutions with shorter calculation time, not best solutions.

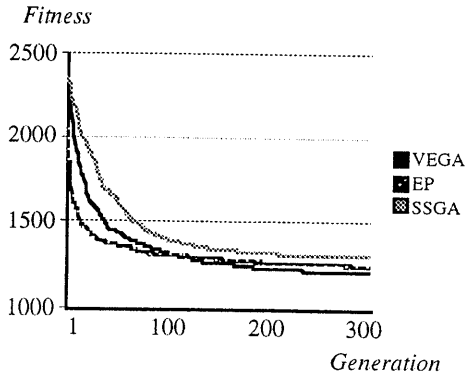


Figure 5.22 Comparison of simulation results

5.1.4 Summary

This subsection applied evolutionary optimization methods to trajectory planning of reconfigurable redundant manipulator. We proposed a hierarchical trajectory planning method using the VEGA. First, we proposed a trajectory planning method based on only forward kinematics. Next, we proposed a hierarchical trajectory planning which generates some intermediate positions and optimizes the combination of the intermediate positions simultaneously. The simulation results of the hierarchical trajectory planning show that the VEGA can generate effective intermediate positions and a collision-free trajectory. Since the trajectory planning uses only forward kinematics, the trajectory planning is an effective method for a reconfigurable redundant manipulator system. As future subjects, we must develop the optimization methods of the structure and the form of cellular manipulator system.

5.2 Application to Intelligent Manufacturing System

Intelligent manufacturing system comprises a new concept for coping with a large number of products and manufacturing processes in a flexible manufacturing system (FMS) [88~98]. We have proposed a self-organizing manufacturing system (SOMS), in which processes self-organize effectively according to other processes. The SOMS is based on the concept of the self-organizing cellular robotic system (CEBOT) which is composed of a number of autonomous robotic units with simple functions [88].

A machine scheduling problem is one of the most important issues in the FMS. In general, the machine scheduling problem is to order n operations on m machines for minimizing cost functions. There are, for example, a job-shop scheduling problem, a flow-shop scheduling problem, a open-shop scheduling problem. The job shop scheduling problem is a special case of the machine scheduling problems and has been solved as a typical problem of machine scheduling problems. However, there are many restrictions concerning usable resource, transportation methods and the location of machining centers in the real manufacturing systems. The problems concerning these restriction are resource allocation problems, path planning problems and optimal location problems, respectively. As to the optimal location problems, the large size of machining centers can not be easily relocated to other space. In general, it is required to take these restrictions into account when performing a task. In fact, a lot of machining lines in the real manufacturing system are controlled under a flow shop scheduling. If machining centers are sequentially located according to given jobs, the path planning of

automated guided vehicles or the locations of belt conveyers becomes easier to design. Therefore, we consider the optimization of a manufacturing system based on the flow shop scheduling defined as a sequencing. Furthermore, there are many researches about scheduling problems according to the several available machines in the FMS, but there are few researches to optimize the location of machining centers according to a given job schedule. Consequently, in this section, we apply a virus-evolutionary genetic algorithm (VEGA) to a manufacturing design problem in the SOMS.

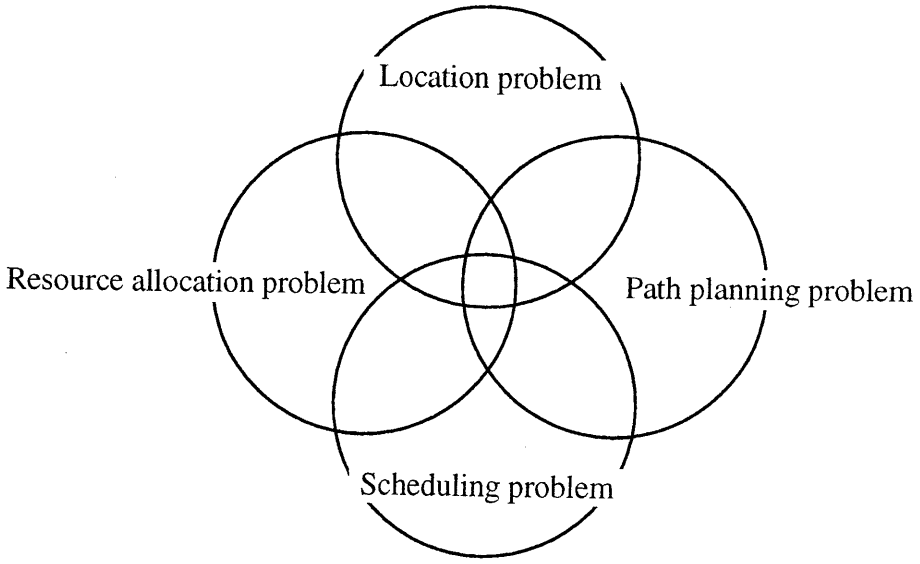


Figure 5.23 Optimization problems in manufacturing system

This section presents an application to a pallet allocation problem in intelligence manufacturing system. In order to create an ideal manufacturing environment, the optimization of each process in the flexible manufacturing system is required. The flexible manufacturing system includes many optimization problems which are ill-defined structures.

5.2.1 Self-organizing Manufacturing System

In general, flexible manufacturing system (FMS) is composed of numerically controlled machines, machining centers, assembling stations and robots, and so on. Furthermore, belt conveyors, transportation vehicles, monorail cars are handled in FMS as automatic transportation methods. The purpose of the FMS is to perform the processes automatically such as design, machining, control, and management by integrating flexible machines and computerized control.

The effectiveness of FMS lies in the processing capability corresponding to high variety. However, FMS has very difficult optimization problems which are ill-defined structures, since FMS has a lot of constraints such as space capacity, machining ability, and time restriction. These problems can not be easily solved and therefore optimization methods such as dynamic programming, branch-and-bound method, neural network, simulated annealing, and evolutionary computation, have been applied for solving these problems.

To create an ideal manufacturing environment, we have been proposed the self-organizing manufacturing system (SOMS) that a process self-organizes according to other processes (Figure 5.24). A module in Figure 5.24 represents a process in the manufacturing system. The module self-organizes based on inputs from other modules and generates outputs. The SOMS is capable of reorganizing hardware as well as software of the manufacturing system.

The flow of the machining process is generally summarized as follows: (1) design of the machining center, (2) planning of a manufacturing schedule, and (3) producing products according to the schedule. In SOMS, it is important to optimize all modules in order to reduce the manufacturing cost. However, the optimization of all modules is very difficult, and the optimized system is vulnerable to breakdowns of the system, delays in the schedules, and so on. Therefore, the self-organization according to other processes in the SOMS is effective with regard to the flexibility of all the manufacturing system.

As an example of the SOMS, we consider a manufacturing system composed of a number of automated guided vehicles (AGVs) and machining centers (Figure 5.25). A machining center has the capability to perform a variety of operations with exchangeable tools. Further, the machining center can produce various products, since the machining center has a redundancy as to operations. Here a redundancy means that a machining center has a capability to equip with more tools than required.

In the SOMS, machining control and manufacturing management are performed not by the centralization but by the decentralization. An AGV transmits not only materials/products but also exchangeable tools. The machining information about a material is kept by the bucket holding the material. Therefore, the AGV can transmit a bucket to the machining center according to its machining information. Since the AGVs go on transmitting a material to an determinate machining center, the machining speciality occurs in the machining center.

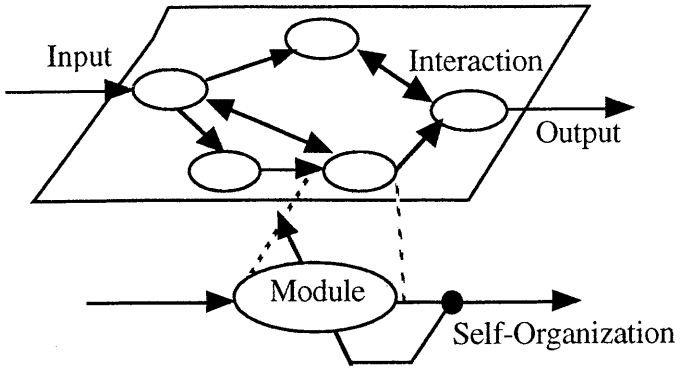


Figure 5.24 Self-organizing Manufacturing system

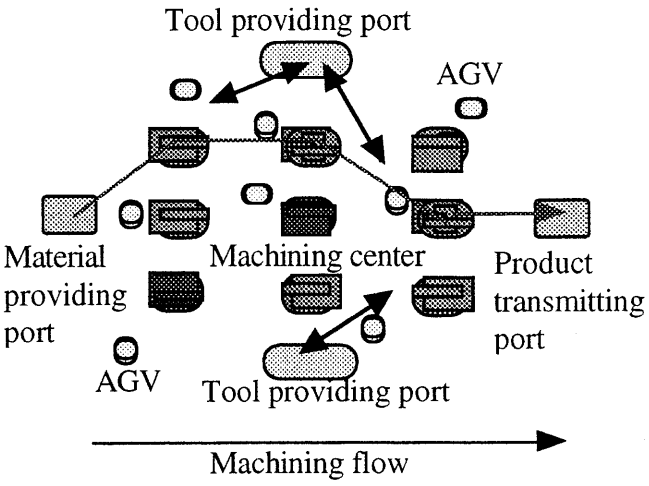


Figure 5.25 Typical self-organizing manufacturing system

5.2.2 Application to Press Machining Line

As an application example of the SOMS, the VEGA is applied a pallet location problem for a press machining line (Figure 5.26). The press machining line is capable of reorganizing in itself and has a redundancy from the viewpoint of the machining process. Here the redundancy of the press machining line means that the number of press machines on the machining line is more than the number of the operations per job. Consequently, the press machining line can

process several different types of jobs. A material is transported from the place for material to the press machine by an AGV. After machining, the product is transported to the place for product by the AGV. Therefore, the machining is performed according to a flow shop scheduling. The assumptions of the press machining line are as follows:

- •A work is composed of n jobs and the job sequence is fixed.
- •A job requires pressing operations by m molds and its pressing sequence is fixed.
- •A pallet on a press machine can be equipped with three types of molds.
- •A press machine takes in a material from the entrance, and puts the material on the exit after pressing.
- •The mold which a job requires is automatically selected on the pallet and its pressing operation is performed.

The assumptions of the AGV are as follows:

- •If there is a material on the exit of a press machine or the place for material, the AGV loads with the material.
- •If the press machine in front of the AGV is in process, the AGV stands by till finishing the pressing operation and loads with the material.
- •If there is not the mold which the job requires in the press machine, the AGV passes by the press machine.

A location optimization problem is to select molds on the pallet to suit to a given work. Here the objective is to reorganize molds on the pallet for minimizing the makespan defined as consuming time until finishing all jobs. Let F_i be a finishing time of job i. Makespan F is defined as follows:

$$F = \max(F_1, F_2, \cdots, F_n) \tag{5.7}$$

Assume that the transporting time of the AGV is ignored in comparison with the pressing time on a press machine and the loading/unloading time of material is the constant.

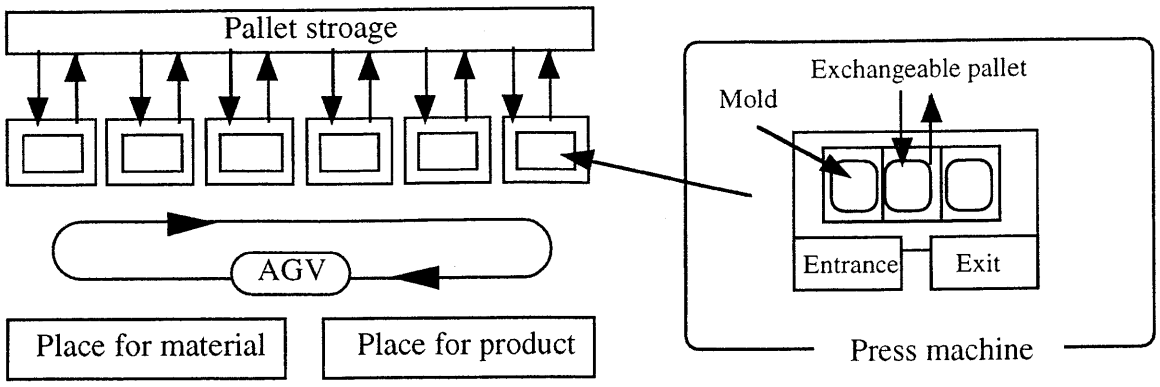


Figure 5.26 Press machining line

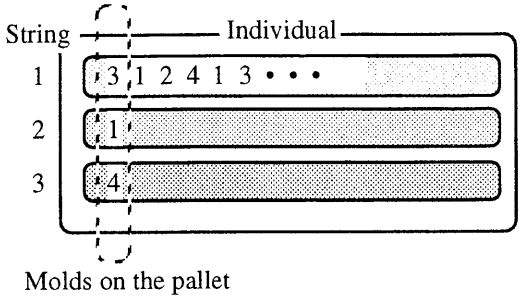


Figure 5.27 Representation of genotype

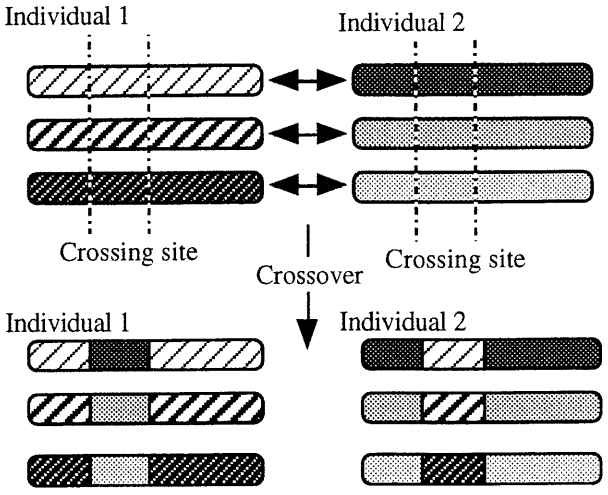


Figure 5.28 Crossover of multi-strings

In this problem, an individual has three strings whose length is equal to the number of the press machine. A gene stands for the mold type and the gene's locus, i, denotes the position of the press machine. That is, the set of the i-th genes on each string in the individual is the pallet of the molds on the i-th press machine (Figure 5.27). Furthermore, the overlapping of the same type of mold on the pallet is not permitted. We use a multi-point crossover operator, a mutation, and transposition as genetic operators. The multi-point crossover performs between the same number of string of each individual (Figure 5.28). The mutation operator replaces a randomly selected gene with one of other genes except genes on the pallet. The transposition operator replaces the position of strings in an individual. The VEGA and SSGA replace the individuals with the highest fitness value with the individuals generated by the multi-point crossover. The fitness function; $fitness_x$ of an individual x is as follows:

$$fitness_x = \max_{i \in S} F_i - aF_x \tag{5.8}$$

where a is a scaling score defined as the constant.

5.2.3 Simulation Results

This subsection shows simulation results and comparison results. We compare simulation results of the VEGA with ones of the SSGA, standard GA (SGA), age structured GA (ASGA). Table 5.4 shows the parameters of work1~3. Table 5.5 shows the machining sequence of the given jobs on work 1. Table 5.6 shows the machining time of each mold on work 1. The population size of SSGA/host and the virus population size are 100 and 10, respectively. The string length of each work is the number of press machine in each work, respectively. The crossover rate and the mutation rate per gene are 0.8 and 0.001, respectively.

Table 5.4 Parameters of work1~work3

	Work1	Work2	Work3
Jobs	10	10	10
Pressing operations	5	7	10
Press machines	8	12	15
Mold types	5	7	7

Table 5.5 Machining sequence of the given jobs on work 1

Job	Sequence of machining
1	2 5 1 3 2
2	1 4 5 4 2
3	5 4 5 2 3
4	4 1 4 5 2
5	4 3 5 2 3
6	5 2 1 4 3
7	4 2 4 1 4
8	1 5 1 2 5
9	1 5 1 5 4
10	1 4 5 2 1

Table 5.6 Machining time of each mold on work 1

Mold type	1	2	3	4	5
Machining time	7	6	8	4	1

Here the simulation results are compared among the SGA, ASGA, SSGA and VEGA. Figure 5.29 and Table 5.7 show simulation results of work1 and simulation results of 10 trials of all works, respectively. From Figure 5.29, the makespans before and after the reorganization are 129 and 114, respectively. The reorganization of molds enables shortening of the makespan by 15. This result shows that the reorganization of a machining center can improve the performance. Furthermore, the VEGA also obtained the best solutions concerning the work 3 which is more difficult works.

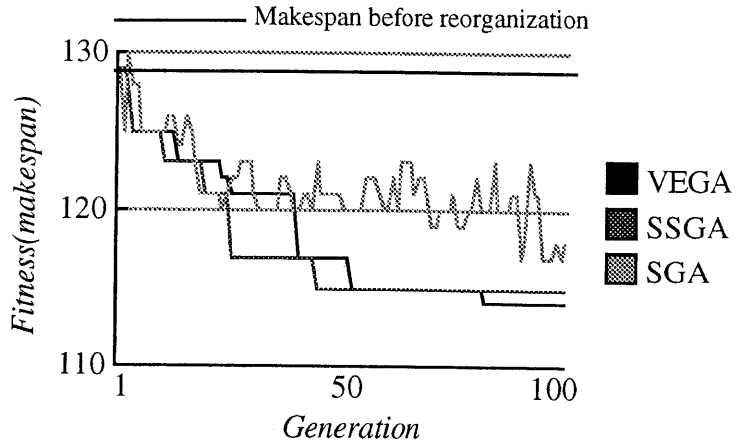


Figure 5.29 Simulation results of pallet location problem

Table 5.7 Simulation results of 10 trials

		SGA	ASGA	SSGA	VEGA
Work1	Min	115	114	114	114
	Mean	120	116	116.40	115.10
	Max	128	123	121	117
Work2	Min	271	257	259	257
	Mean	281.10	264.20	270.30	262.5
	Max	295	273	280	269
Work3	Min	326	307	310	307
	Mean	338.60	320.40	325.20	318.80
	Max	352	331	333	329

5.2.4 Summary

This subsection presented the application to a self-organizing manufacturing system; SOMS. As an example of SOMS, VEGA is applied to a pallet location problem of a press machining line. Simulation results show that the SOMS is capable of shortening the makespan of the press machining. The essential of SOMS lies in the reorganizing ability.

5.3 Application to Fuzzy Controller

Fuzzy systems such as fuzzy logic controllers, fuzzy reasoning, and fuzzy modeling are applied in many fields for engineering, medical engineering, and even social sciences. Some fuzzy control systems are already applied and can be seen in some of home appliance, transportation systems, manufacturing systems, and so on. Fuzzy systems have a characteristic to represent human knowledge or experiences as fuzzy rules. However the fuzzy systems have some problems. In most fuzzy systems, the shape of membership functions of the antecedent, the consequent, and the fuzzy rules have been determined and tuned through trial and error by operators, and it therefore takes much time to determine and tune them, and it is very difficult to design the optimal fuzzy system in detail. This problem is more serious, when the fuzzy system is applied to more complex systems.

In order to solve this problem, some self-tuning methods have been proposed such as a fuzzy neural network [29~33] that applies the back propagation algorithm for learning, a fuzzy learning controller applying radial basis functions [32], a fuzzy logic controller utilizing GAs for deciding the shapes of membership functions and fuzzy rules [27,34,35]. These methods can learn faster than neural networks. However an operator must determine the number and shapes of membership functions before learning, and the learning ability and accuracy of approximation are related to the number or shape of membership functions. Fuzzy inference with many membership functions and fuzzy rules has high learning ability, however there are some redundant rules or unlearned rules. The number of rules is the product of the number of membership function for each input, and the number of rules is increased as exponential with increase of the input dimension. Therefore operators must pay much attention to decide the structure of the membership functions.

The fuzzy inference based on the Radial Basis Function which adds a new rule for the maximal error point through learning process, has been proposed. In these methods, fuzzy rules depend on the learning data set and if the learning data is biased, there are some unlearned areas or redundant fuzzy rules, therefore the learned fuzzy rules are not optimized. Besides, these methods do not integrate or delete a fuzzy rule, only add a new fuzzy rule. Therefore they also have the problem of the increasing number of fuzzy rules that is the cause of consuming more calculation time and memory. Self-tuning fuzzy inference has been proposed for solving these problems. The membership function of the antecedent is expressed by the radial basis function with an insensible range. The supervised/unsupervised learning algorithms are based on the genetic algorithm, and the supervised learning are also utilized the gradient

decent method to tune the shapes of membership functions and the consequent values. We apply the VEGA for a self-tuning fuzzy controller as a learning method to obtain fuzzy rules and membership functions which can carry out with high performance. Furthermore, the proposed fuzzy controller is applied to the cart-pole problem, and its effectiveness is shown through computer simulation.

5.3.1 RBF based Fuzzy System with VEGA

5.3.1.1 Fuzzy System Based on Radial Basis Function

First of all, we present the calculating formulas of the RBF between input variables and output variables. The fitness value of the rules and the output value Y are expressed by eqs. (5.9) and (5.10),

$$Y = \frac{\sum_{i=1}^{I} \mu_i W_i}{\sum_{i=1}^{I} \mu_i} \tag{5.9}$$

$$\mu_i = \prod_{j \in J} \mu_{ij} \tag{5.10}$$

where i, j, and p are the input number, the fuzzy rule's number, and the data set's number, respectively. The consequence is expressed by real number W_i. The shapes of the membership functions are the RBF with an insensible range c that is useful for reducing the membership functions and fuzzy rules (Figure 5.30). The membership function in the i-th input value and the j-th fuzzy rule is expressed by

$$f_i(\mathrm{I}_j) = \begin{cases} -b_{ij}(|\mathrm{I}_j - a_{ij}| - c_{ij})^2 & \text{if } |\mathrm{I}_j - a_{ij}| > c_{ij} \\ 0 & \text{if } |\mathrm{I}_j - a_{ij}| \le c_{ij} \end{cases} \tag{5.11}$$

$$\mu_{ij} = \exp\{f_i(\mathrm{I}_j)\} \tag{5.12}$$

where a, b, and c are the coefficients that decide the shape of membership functions shown in Figure 5.30.

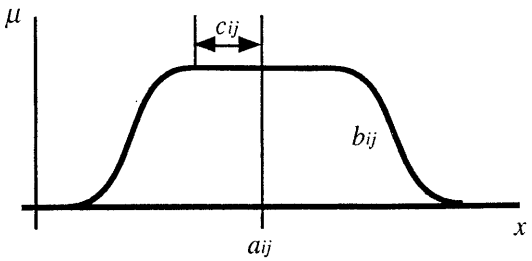


Figure 5.30 Membership function based on RBF

5.3.1.2 Coding, Selection, and Genetic Operators

We use a binary coding to encode membership functions. An antecedent part is expressed by a binary number of $(3n+1)$ bits every membership function in this simulation: each coefficient a, b, c needs n bits, 1 bit is used as a flag of the membership function's validness. A consequent part is encoded to the m bits binary number. The following eqs ,(5.13), (5.14), and (5.15), are used to decode the binary number into the parameters of membership functions (Figure 5.30 and Figure 5.31). Equation (5.16) is used to decode into the value of the consequent parts.

$$a_{ij} = \frac{2Sa_{ij}}{2^n - 1} - 0.5 \tag{5.13}$$

$$b_{ij} = A\left(\frac{Sb_{ij}}{2^n - 1}\right)^3 \tag{5.14}$$

$$c_{ij} = \frac{Sc_{ij}}{B\left(2^n - 1\right)} \tag{5.15}$$

$$W_i = \frac{2w_i}{2^m - 1} - 0.5 \tag{5.16}$$

where A and B are coefficients of a slope of the membership function and a range of the insensible region, respectively.

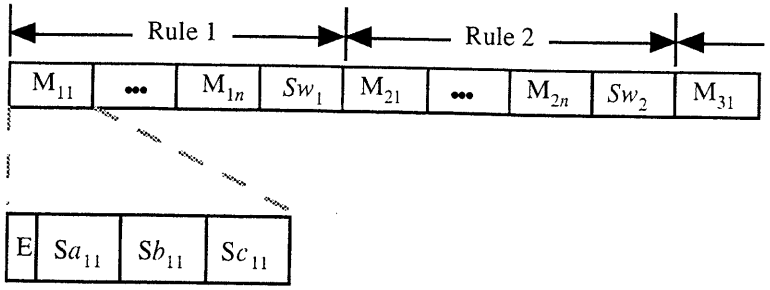


Figure 5.31 Coding of fuzzy rule

Fitness function consists of the performance index and the numbers of the membership functions and fuzzy rules as following equations:

$$F = \alpha P + \beta R_n + \gamma M_n \tag{5.17}$$

where P, R_n, M_n, and α, β, γ means the performance index, e.g. the difference between the output of the system and the desired performance, the number of rules, the number of membership functions, and the coefficients, respectively. In this equation, coefficients are classified into two types, one is the performance (α), the other is the size of fuzzy system (β and γ).

The objective is to acquire a well performed fuzzy controller without redundant fuzzy rules and membership functions. Therefore, the objective results in the minimization problem of the fitness function F. The VEGA uses the 'delete least fitness' method as a selection operator. Thus VEGA removes host individuals with the worst fitness value from the host population. The selection operation can acquire the preferable fuzzy system such as small fuzzy system (β and γ are larger than α), or high accurate fuzzy system (α is larger than β and γ), by setting these coefficients.

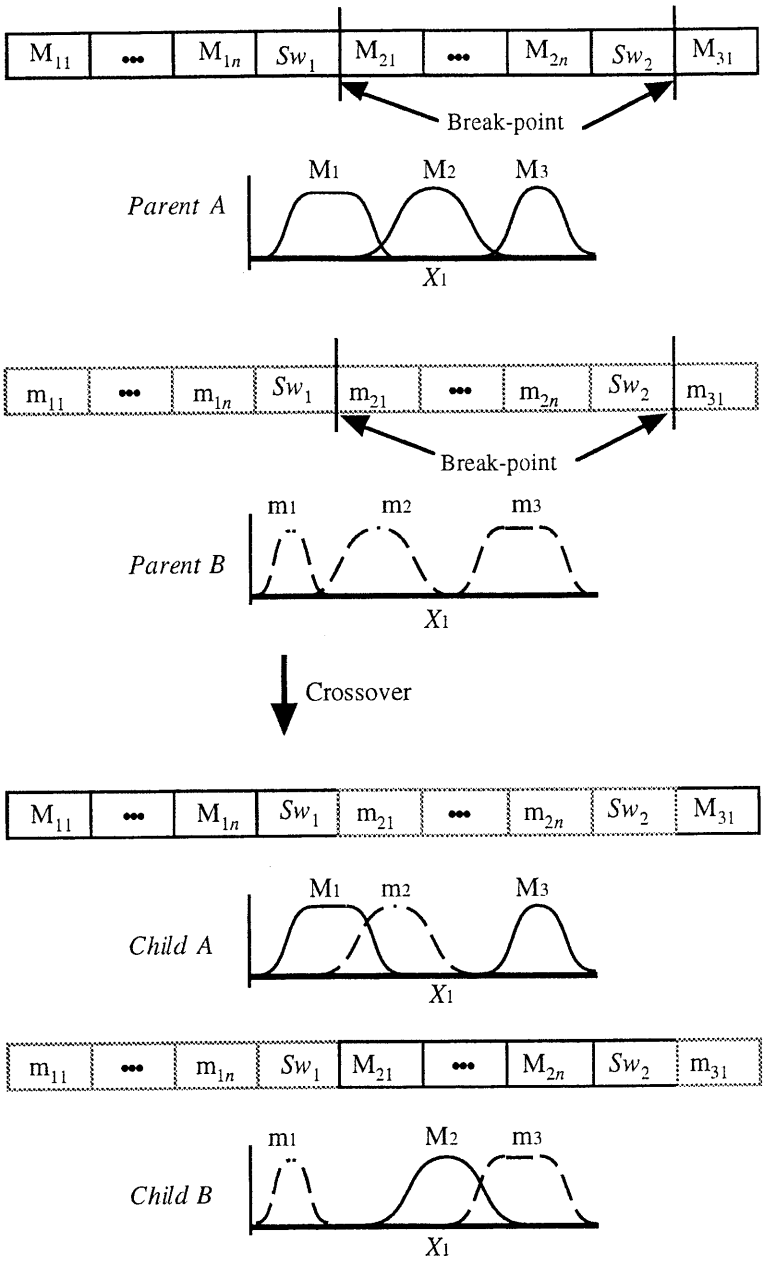


Figure 5.32 Crossover operator for fuzzy system

In order to generate a new set of membership functions and rules, we apply the two-point crossover operator (Figure 5.32). The crossover operator randomly selects the target individuals. The crossover is carried out between two individuals, and then some rules are exchanged each other. We use two types of mutation operators, (A) uniform distribution random set based mutation operator (Figure 5.33) and (B) normal distribution random number based

mutation operator (Figure 5.34). In both cases, the target strings and mutation sites are randomly selected. The mutation operator A changes some bits of the strings and uses for global and rough search. This operator can change the enable/disable flag of the membership function. The mutation operator B does not change the bits of the chromosomes, but adds (or subtracts) random values to (from) the parameters of the membership functions, S_a, S_b, and S_c, and the consequent values W.

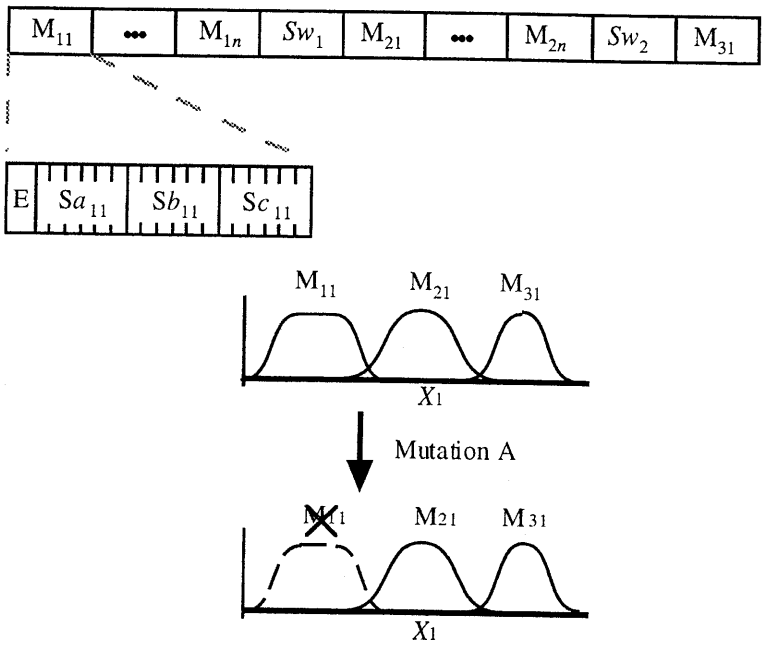


Figure 5.33 Mutation operator A on genotype

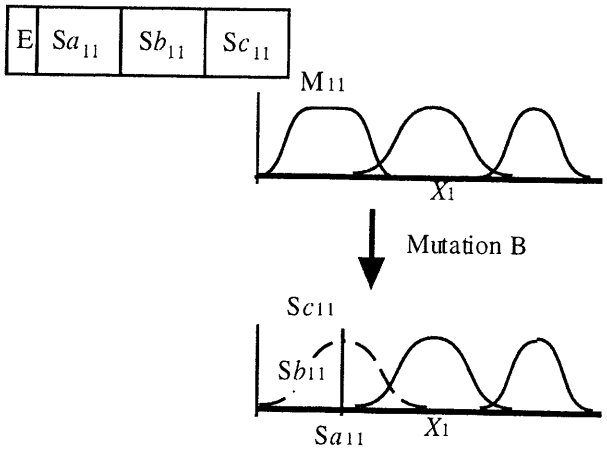


Figure 5.34 Mutation operator B on phenotype

A virus has the information of some rules, membership functions, and a central point of the transduction/reverse transcription area on the input space. A virus transduces some substrings from a host individual(Figure 5.35). This operator selects effective schemata (effective fuzzy rules here) to be transmitted between host individuals. A substring of a virus is generated by selecting some fuzzy rules from the string of a host individual. The fuzzy rules to be selected are the most frequently used ones in the fuzzy rules of the host individual. At the same time, the central point of the virus is defined according to the transduced fuzzy rules. An example of reverse transcription operator is shown in Figure 5.36. In this example, the virus individual and the host individual are one fuzzy rule and three fuzzy rules, respectively. First, the virus deletes the fuzzy rules that include the central point of the virus. Next, the virus overwrites its fuzzy rules on the host's string.

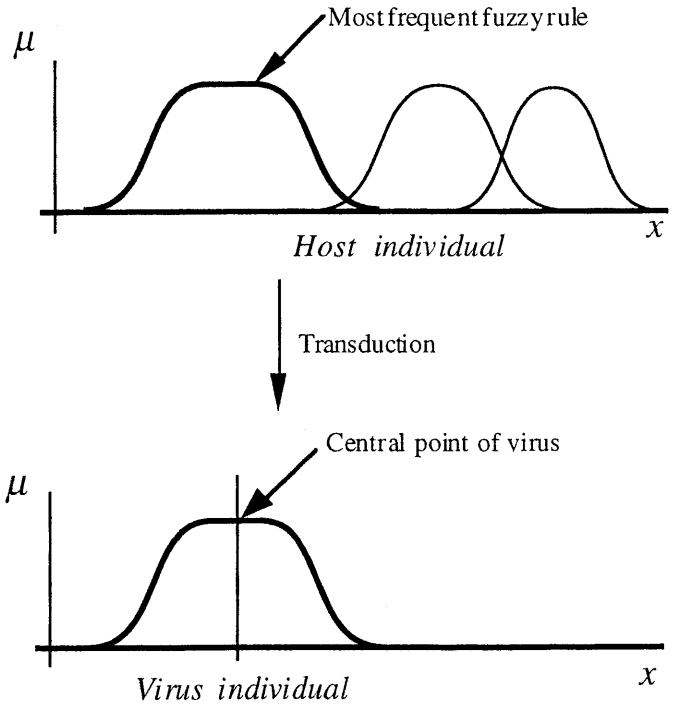


Figure 5.35 Transduction operator for fuzzy rules

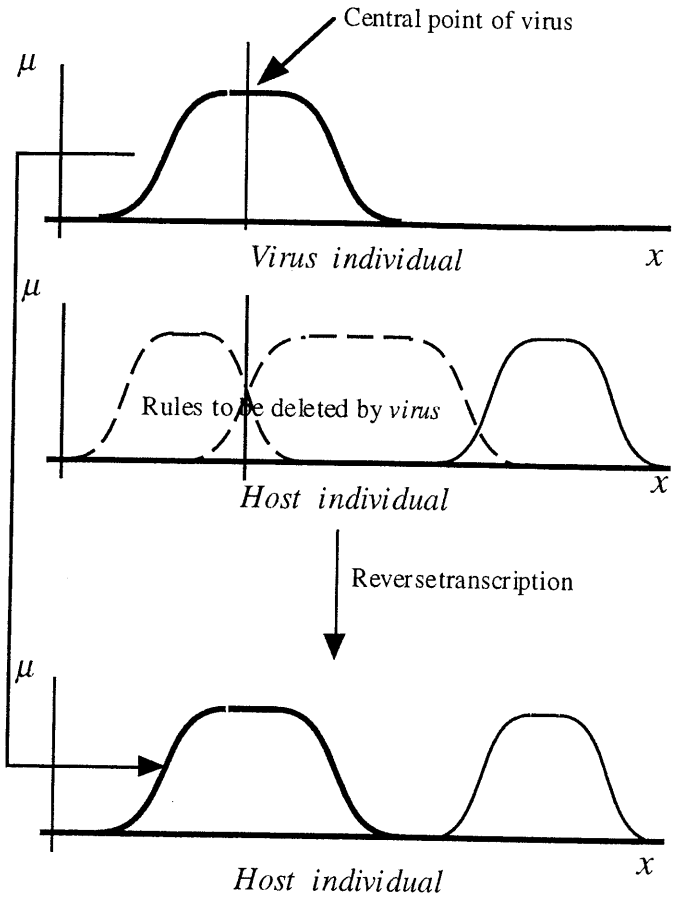


Figure 5.36 Reverse transcription operator on membership function

5.3.2 Application to RBF Fuzzy Controller for Cart-Pole System

We apply the proposed fuzzy control system for a cart-pole system (Figure 5.37). The pole is controlled from pendant position to upright position and then keep it up by the RBF fuzzy controller (Figure 5.38). The cart-pole system is described by following equations :

$$\ddot{r} = \frac{F - \mu_c sign(\dot{r}) + \tilde{F}}{M + \tilde{m}} \tag{5.18}$$

$$\ddot{\theta} = -\frac{3}{4l}\left(\ddot{r}\cos\theta + g\sin\theta + \frac{\mu_p \dot{\theta}}{ml}\right) \tag{5.19}$$

where

$$\tilde{F} = ml\dot{\theta}^2 \sin\theta + \frac{3}{4} m \cos\theta \left(\frac{\mu_p \dot{\theta}^2}{ml} + g \sin\theta \right) \tag{5.20}$$

$$\tilde{m} = m\left(1 - \frac{3}{4}\cos^2\theta\right) \tag{5.21}$$

and M = 1.0(kg), μ_c = 0.0005(N), μ_p= 0.000002(kg•m), r, θ, l=0.2(m), and m mean the cart mass, the friction of the cart on track, the friction at hinge between the cart and the pole, the cart position, the pole deviation from vertical, the pole length, and the pole mass, respectively.

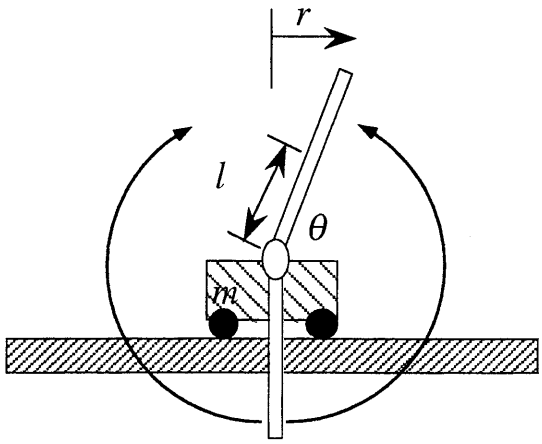


Figure 5.37 Cart-pole system

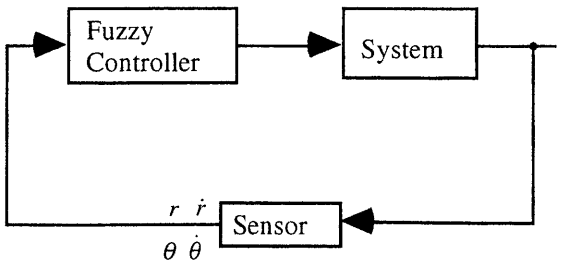


Figure 5.38 RBF Fuzzy Controller

The fitness function used in this simulation is as follows:

$$Object = \alpha T_{standup} - \beta \sum_{t=0}^{T} (\theta_d - \theta(t))^2 - \gamma \sum_{t=0}^{T} (r_d - r(t))^2 - \delta R_n - \eta M_n \qquad (5.22)$$

where T is simulation time for fuzzy controller. The second and third terms in eq.(5.22) mean the summation of the error of the pole angle and the cart position, respectively. The fourth and fifth terms mean the number of fuzzy rules and the number of membership functions, respectively.

The Iteration times of generation is 500. The population size is 200 and the virus population size is 20. Figure 5.39 and Figure 5.40 show one of simulation results of the acquired fuzzy controller. These figures show that acquired fuzzy controller can swing up and keep the pole at the upright position with desired cart position.

5.3.3 Simulation Results

Figure 5.41, Figure 5.42 and Figure 5.43 show the change of the number of membership functions, the number of fuzzy rules, and the fitness value through the iterations, respectively. VEGA can reduce the rules more effectively than the previous study obtaining the same performance. The number of final rules is 7 with 17 membership functions. In Figure 5.43, the GA outperforms the VEGA at first, since the virus population in the VEGA does not have effective schemata. The virus population can not carry out effective reverse transcriptions. Then, the virus population gradually become to have effective schemata and propagate them between host individual. At last, both of the VEGA and GA obtain better solutions, but the VEGA can reduce the rules more effectively than the GA. The reason why the VEGA reduces the rules lies in the reverse transcription operator, which removes some redundant rules from a host individual. As the result of reverse transcription operator, the fuzzy rules of host individual are refined without redundancy.

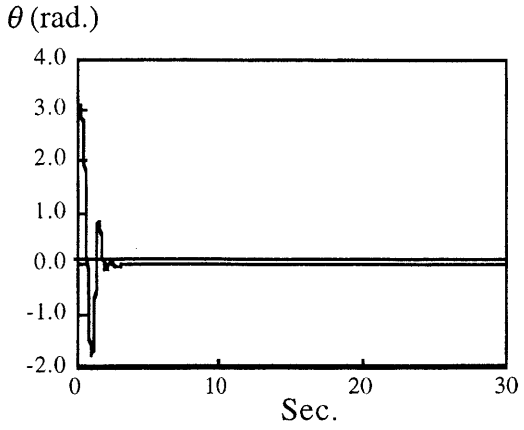


Figure 5.39 Simulation result of the pole angle

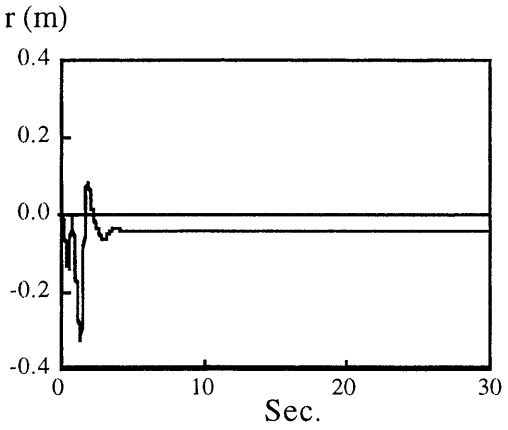


Figure 5.40 Simulation result of the cart position

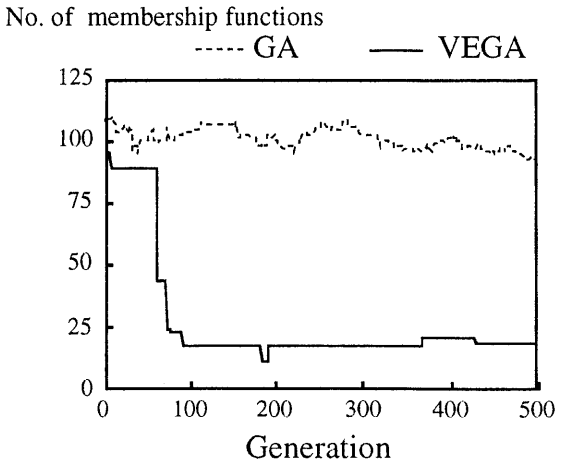


Figure 5.41 Change of number of membership functions

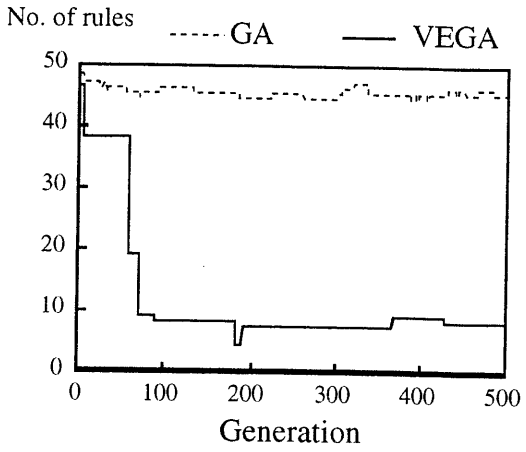


Figure 5.42 Change of number of fuzzy rules

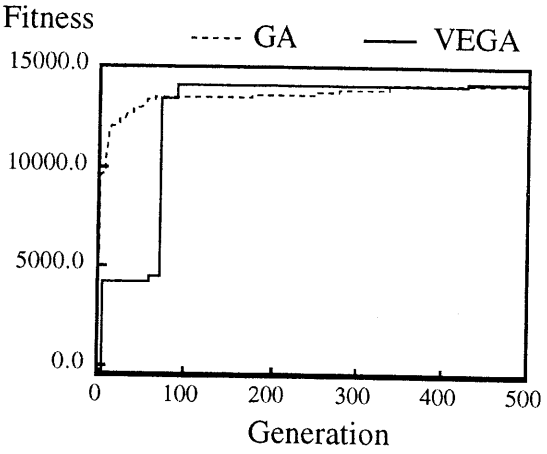


Figure 5.43 Change of fitness value

5.3.4 Summary

In this section, we proposed a new fuzzy control system using radial basis functions and a learning method based on the VEGA. Further, we showed the effectiveness of the fuzzy controller through simulation results of a cart-pole problem. The VEGA can reduce redundant fuzzy rules effectively without performance decline. Therefore, the acquired fuzzy controller can carry out the tasks with small fuzzy rules.

5.4 Summary

This chapter presents the three types of engineering applications of evolutionary optimization methods and its effectiveness through computer simulation.

First, we present the application to trajectory planning for redundant manipulators and propose a hierarchical trajectory planning method composed of a position generator and a trajectory generator, which are based only on forward kinematics. The simulation results of the hierarchical trajectory planning show that the VEGA can generate effective intermediate positions and a collision-free trajectory.

Second, we propose a self-organizing manufacturing system (SOMS) in which a process effectively self-organizes according to other processes and present a pallet allocation problem and present the application to a pallet location problem of a press machining line. Simulation results show that SOMS is capable of shortening the makespan of the press machining and the VEGA can reduce the makespan.

Last, we propose a new fuzzy control system using radial basis functions and a learning method based on the VEGA. Furthermore, we showed the effectiveness of the fuzzy controller through simulation results of a cart-pole problem. The simulation results show that the VEGA can reduce redundant fuzzy rules effectively without performance decline. Therefore, the acquired fuzzy controller can carry out the tasks with small fuzzy rules.

Chapter 6 Conclusions

6.1 Concluding Remarks

Biology and evolution often provide effective but simple ideas and we can apply these ideas to various fields in the human society. In this dissertation, we propose biologically inspired evolutionary optimization methods, which are age-structured genetic algorithm, virus-evolutionary genetic algorithm and virus evolutionary algorithm. Furthermore, we discuss their effectivenesses through some computer simulation results of conventional and traditional optimization problems and engineering optimization problems.

However, evolution is sure to proceed to a better state or organism, not the best one. On the other hand, optimization has a trade-off between the performance of the solution and searching speed. The evolutionary optimization methods are basically composed of a selection operation and genetic operators. The genetic operators explore the search space generated by all the population, while the selection operation exploits the search space for generating new candidate solutions. The evolutionary optimization methods have two essentials concerning optimization.

The first one is the availability to apply to various optimization problems and solve them with better solutions satisfying their aspiration levels. Various techniques of coding and genetic operators make it possible to search a feasible solution space effectively. Furthermore, the evolutionary optimization methods can be easily incorporated with local search methods. In addition, if we would know the features of optimization problems in detail, optimization methods dependent on the problem can solve the problem better than evolutionary optimization methods which have much vain in the search. However, the real problems have a lot of uncertainty, vagueness and ambiguity. The evolutionary optimization methods can be successfully applied to these kinds of problems which are defined as black boxes.

The other is the solvability of better solution with less computational time. The solvability lies in the symbolic operations based on schemata. The crossover operators can efficiently search the potential solution space with simple symbolic operations inheriting genetic information from parents. This feature is different from simple random searches, and building block hypothesis indicates that the crossover operators accelerate the evolution of a population. In addition, genetic operators determine the direction of evolution, and particular genetic

operations give a population of individuals their peculiar potentialities for generating new individuals. Furthermore, the virus infection operator can increase effective schemata in a population and realizes the quick solution of optimization problems with the directionality in search.

However, the evolutionary optimization methods have the following disadvantages. The first one is the certainty of solutions obtained by evolutionary optimization methods. We can discuss the feasibility of the solutions by using the designed aspilation level against the optimization problem, but we can not estimate the certainty of solutions since the solution space is dependent on the optimization problems. If the optimization problem is clear, we can estimate the certainty of solutions. The other is that it is difficult to discuss the effectiveness of genetic operators and selection operations. The evolutionary optimization methods does not have general criteria to evaluate them, since the solution space is problem-dependent and the behavior of any evolutionary computation can not be anticipated by analyzing its components.

Nowadays, natural evolution has not understood clearly and precisely, and the research of evolution has still continued. And the genetic algorithms are simple applications of the evolutionary theories. Therefore the genetic algorithm will evolve by the new theory of the evolution in nature. Furthermore, we hope that future genetic algorithms can explain the mechanism of evolution in nature.

6.2 Future Works

Evolutionary optimization methods are very difficult to analyze. We intend to analyze the behavior of evolutionary optimization methods by using Markov chain and other methods. In addition, we require optimization methods robust to any types of optimization problem with minor change of the optimization method but powerful and quick. This is a very difficult problem. To the contrary, we can model optimization problems suitable for optimization methods. In fact, human being can alter its environment in nature. We therefore intend to develop a co-evolutionary method optimizing both of the solution space of the optimization problem and optimization method toward the optimal solutions.

References

[1] Burch, J.G., Strater, F.R., Information Systems - Theory and practice, California: Hamilton Publishing Com., 1974.

[2] Saga, A.P., Manufacturing System Engineering and Management, Proceedings of 1996 IEEE International Conference on System, Man, Cybernetics, China, 1996, pp.1-9.

[3] Ecker, J., Kupferschmid, M., Introduction to Operations Research, New York: John Wiley & Sons Inc., 1988.

[4] Hastigs, K., Introduction to the Mathematics of Operations Research, New York: Marcel Dekker Inc., 1989.

[5] Ash, R.B., Information Theory, New York: Dover Publishing Inc., 1990.

[6] Wiener, N., Cybernetics or Control and Communication in the Animal and the Machine, Massachusetts: The M.I.T Press, Second Edition, 1962.

[7] Dixon, L.C.W., Szegö, G.P., Toward Global Optimization, Proceedings of a Workshop at the University of Cagliari, Italy, Amsterdam: North-Holland Publishing Com., 1975

[8] Russell, S.J, Norvig, P., Artificial Intelligence A Modern Approach, New Jersey: Prentica-Hall, Inc., 1995.

[9] Chiang, A.L., Elements of Dynamics Optimization, New York: McGraw-Hill, Inc., 1992.

[10] Baba, N., Convergence of a Random Optimization Method for Constrained Optimization Problems, Journal of Optimization Theory and Applications, 1981, vol.33, no.4, pp.451-461.

[11] Kirkpartick, S., Gelatt Jr, C., and Vecchi, M, Optimization by Simulated Annealing, Science, 1983, no. 220, pp.671-680.

[12] Goldberg, D., Genetic Algorithms in Search, Optimization and Machine Learning, Massachusetts: Addison Wesley Publishing Company Inc., 1989.

[13] Fogel, D., Evolutionary Computation, New York: IEEE Press, 1995.

[14] Holland, J., Adaptation in Natural and Artificial Systems , First edition, Massachusetts, The MIT Press, 1992.

[15] Schwefel, H., On the Evolution of Evolutionary Computation, Computational Intelligence Imitating Life, New York: IEEE Press, 1994, pp.116-124.

[16] Rechenberg, I., Evolution Strategy, Computational Intelligence Imitating Life, New York: IEEE Press, 1994, pp.147-159.

[17] Atmar, W., Notes on the Simulation of Evolution, The IEEE Transaction on Neural Network, 1994, pp.130-147.

[18] Michalski, R., Tecuci, G., Machine Learning, A Multistrategy Approach, Vol.IV, California: Morgan Kaufmann Publishing, 1994.

[19] Stamatious V. Kartalopoulos, Understanding Neural Networks and Fuzzy Logic, New York: IEEE Press, 1996.

[20] Rumelhart, D.E., McClelland, J.L. and The PDP Research Group, Parallel Distributed Processing, MIT Press, 1986 , vol.1, p.547; vol.2, p.611.

[21] Aarts, E. and Korst, J., Simulated Annealing and Boltzmann Machines -A Stochastic Approach to Combinatorial Optimization and Neural Computing-, Chichester: John Wiley & Sons Ltd, 1989.

[22] Jang, J.-S.R., Sun, C.-T., Mizutani, E., Neuro-Fuzzy and Soft Computing, New Jersey: Prentica-Hall, Inc., 1997.

[23] Sousa, J.M., Babuska, R., Verbruggen, H.B., Some Computational Issues in Fuzzy Predictive Control, Proceedings of International Workshop on Artificial Intelligence in Real-Time Control, Slovenia, 1995, pp.66-70.

[24] Fukuda, T., Shimojima, K., Arai, F., Matsuura, H., Multi-Sensor Integration System based on Fuzzy Inference and Neural Network, Journal of Information Sciences, 1993, vol.71, no. 1 and 2, pp. 27-41.

[25] Whitley, D., Strakweather, T. and Bogart, C., Genetic Algorithms and Neural Networks: Optimizing Connection and Connectivity, Parallel Computing 14, 1990, pp.347-361.

[26] Baba, N., Utilization of Stochastic Automata and Genetic Algorithms for Neural Network Learning, Parallel Problem Solving from Nature 2, Elsevier Science Publishers B. V., 1992, pp.431-440.

[27] Lee, M.A, Takagi, H., Integrating Design Stages of fuzzy systems using genetic algorithms, Proceeding of Second IEEE International Conference on Fuzzy System, 1993, pp. 612-617.

[28] Shimojima, K., Fukuda, T., Arai, F., Matsuura, H., Multi-Sensor Integration System utilizing Fuzzy Inference and Neural Network, Journal of Robotics and Mechatronics, 1992, vol.4, No.5, pp.416-421.

[29] Higgins, C., M., Goodman, R., M., Learning Fuzzy Rule-Based Neural Networks for Function Approximation, Proceeding of IJCNN, 1992 , vol.1, pp.251-256.

[30] Horikawa, S., Furuhashi, T., Uchikawa, Y., On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm, IEEE Trans. Neural networks, 1992, vol.3, no. 5, pp. 801-806.

[31] Katayama, R., Kajitani, Y., Kuwata, K., Nishida, Y., Self generating radial basis function as neuro-fuzzy model and its application to nonlinear prediction of chaotic time series, IEEE International Conference on Fuzzy Systems 1993, pp. 407-414.

[32] Linkens, D.A., Nie, J., Fuzzified RBF Network-based learning control: structure and self-construction, The IEEE International Conference on Neural Networks, 1993, pp. 1016-1021.

[33] Nomura, H., Hayashi, I., Wakami, N., A self-tuning methods of fuzzy control by decent method, Proceeding of 4th IFSA Congress, Engineering, 1991, pp. 155-158.

[34] Ishigami, H., Fukuda, T., et al., Structure Optimization of Fuzzy Neural Networks by Genetic Algorithm, Fuzzy Sets and Systems vol.71, no.3, 1995, pp.257-264.

[35] Wu, K.H., Chen, C.H., Lee, J.D., A Cache-Genetic-Based Modular Fuzzy Neural Network for Robot Path Planning, Proceedings of 1996 IEEE International Conference on System, Man, Cybernetics, China, 1996, pp.3089-3094.

[36] Langton, C.G., Taylor, C., Farmer, J.D., Rasmussen, Artificial Life II, Massachusetts: Addison Wesley Publishing Company, 1992.

[37] Nadel, L., Stein, D.L., 1992 Lectures in Complex System, Lecture Vol.V, Massachusetts: Addison Wesley Publishing Company, 1993.

[38] Langton, C.G. ed., Artificial Life -An Overview, Massachusetts, The MIT Press, 1995

[39] Holland, J., Hidden Order - How Adaptation Builds Complexity, Massachusetts: Addison Wesley Publishing Company Inc., 1995.

[40] Adleman, L.M., Molecular Computational of Solutions to Combinatorial Problems, Science, 1994, Vol.266, 11 November, pp.1021-1023.

[41] Lipton, R.J., DNA Solution of Hard Computational Problems, Science, 1995, Vol.268, 28 April, pp.542-545.

[42] Ridley, M., Evolution, 1st ed., Massachusetts: Blackwell Scientific Publications Inc., 1993.

[43] Anderson N., Evolutionary Significant of Virus Infection, Nature, 1970, vol. 227, pp. 1346-1347.

[44] Darwin, C., On the Origin of Species, London: John Murray, 1859.

[45] Crow, J., Basic Concepts in Population, Quantitative, and Evolutionary Genetics, New York: W. H. Freeman and Company, 1986.

[46] Nakahara, H., Sagawa, T., Virus Theory of Evolution, Tokyo: Tairyusha, 1989, in Japanese.

[47] Campbell, R.C., Statics for Biologists, Third ed., Cambridge, Cambridge University Press, 1989.

[48] Fogel, D.B., Phenotype, Genotype, and Operators in Evolutionary Computation, The 1995 IEEE International Conference on Evolutionary Computation, 1995, pp.193-198.

[49] Bäck T., Selective Pressure in Evolutionary Algorithms: A Characterization of Selective Mechanisms, The First IEEE Conference on Evolutionary Computing, Frorida, 1994, vol.1, pp.57-62.

[50] Whitley, D., The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproduction is Best, The Third International Conference on Genetic Algorithms, Virginia, pp.110-1115.

[51] Bäck T., Generalized Convergence Models for Tournament- and (μ, λ)-Selection, The First IEEE Conference on Evolutionary Computing, Frorida, 1994, vol.1, pp.57-62.

[52] Jones, T., Crossover, Macromutation, and Population-based Search, Proceeding of The Sixth International Conference on Genetic Algorithms, 1995, pp.73-80.

[53] Davidor, Y., Analogous Crossover, The Third International Conference on Genetic Algorithms, Virginia, 1989, pp.98-103.

[54] Radcliffe, N., A Studying Set Recombination, The Fifth International Conference on Genetic Algorithms, Virginia, 1993, pp.23-30.

[55] Bäck T., The Iteration of Mutation Rate, Selection, and Self-Adaptation Within a Genetic Algorithm, Parallel Problem Solving from Nature, 2, Elsevier Science Publishers, B.V., 1992, 85-94.

[56] Back, T., Optimal Mutation Rates in Genetic Search, The Fifth International Conference on Genetic Algorithms, Virginia, 1993, pp.2-9.

[57] Mühlenbein, H.,How Genetic Algorithms Really Work I.Mutation and Hillclimbing, Parallel Problem Solving from Nature, 2, Elsevier Science Publishers, B.V., 1992, 15-25.

[58] Horn J., Finite Markov Chain Analysis of Genetic Algorithm with Niching, The Fifth International Conference on Genetic Algorithms, Virginia, 1993, pp.110-127.

[59] Suzuki , J., A Markov Chain Analysis on a Genetic Algorithm, The Fifth International Conference on Genetic Algorithms, Virginia, 1993, pp.146-153.

[60] Rudolph, G., Convergence Analysis of Canonical Genetic Algorithm, IEEE Transaction on Neural Network, 1994, Vol.5, No.1, pp.61-101.

[61] Fogel, D.B., Ghozeil, A., Using Fitness Distribution to Design More Effective Evolutionary Computation, The 1996 IEEE International Conference on Evolutionary Computation, 1996, pp.11-19.

[62] Manderick, B., Spiessens, P., How to Select Genetic Operators for Combinatorial Optimization Problems by Analyzing their Fitness Landscape, Computational Intelligence Imitating Life, IEEE Press, 1994, pp.170-181.

[63] Davidor Y., A Natural Occurring Niche & Species Phenomenon: The model and First Results, The Fourth International Conference on Genetic Algorithms, Virginia, 1991, pp.257-263.

[64] Cohon, J., Martin, W., and Richards, D., A Multi-Population Genetic Algorithm for Solving the K-Partition Problem on Hyper-Cubes, The Fourth International Conference on Genetic Algorithms, Virginia, 1991, pp.244-248.

[65] Collins, R., and Jefferson, D., Selection in Massively Parallel Genetic Algorithms, The Fourth International Conference on Genetic Algorithms, Virginia, 1991, pp.249-256.

[66] Muhlenbein, H. , Schomisch, M., and Born, J., The Parallel Genetic Algorithm as Function Optimizar, The Fourth International Conference on Genetic Algorithms, Virginia, 1991, pp.271-279.

[67] Manderick, B., and Spoessens, P., FINE- GRAINED PARALLEL GENETIC ALGORITHMS, The Fourth International Conference on Genetic Algorithms, Virginia, 1991, pp.428-433.

[68] Mahfoud, S., and Goldberg, D., A, genetic algorithm for parallel simulated annealing, Parallel Problem Solving from Nature 2, Elsevier Science Publishers B. V., 1992.

[69] Davis, L., Bit Climbing, Representational Bias, and Test Suite Design, The Fourth International Conference on Genetic Algorithms, Virginia, 1991, pp.18-23.

[70] Renders, J., and Bersini, H., Hybridizing Genetic Algorithms with Hill-Climbing Methods for Global Optimization: Two Possible Ways, The First IEEE Conference on Evolutionary Computing, Frorida, 1994, vol.1, pp.312-317.

[71] Syswerda, G., A Study Reproduction in Generational and Steady-State Genetic Algorithms, Foundations of Genetic Algorithms, San Mateo: Morgan Kaufmann Publishers, Inc., 1991, p94-101.

[72] Kursawa, F., Towards Self-Adaptive Evolution Strategies, The 1995 IEEE International Conference on Evolutionary Computation, 1995, pp.283-288.

[73] Fogel, L.J., Evolutionary Programming in Perspective, Computational Intelligence Imitating Life, New York: IEEE Press, 1994, pp.135-146.

[74] Fogel, D., A comparison of Evolutionary Programming and Genetic Algorithms on Selected Constrained Optimization Problems, SIMULATION, 1995, vol.64, no.6, pp.397-404.

[75] Koza, J., Genetic Programming, Massachusetts The MIT Press, 1992.

[76] Koza, J., Genetic Programming II, Massachusetts The MIT Press, 1994.

[77] Hinterding, R., Mapping, Order-independent Genes and the Knapsack Problem, Proceedings of The First IEEE Conference on Evolutionary Computing, 1994, Vol.1, pp.13-17.

[78] Graeme Faulkner: Genetic Operators using Viral models, The 1995 IEEE International Conference on Evolutionary Computation, vol.2, 1995, pp.652-656.

[79] Tamaki, H., Kita, H., et al, A Comparison Study of Genetic Codings for the Traveling Salesman Problem, The First IEEE Conference on Evolutionary Computing, Frorida, 1994, vol.1, pp.1-6.

[80] Bui, T.N., Moon, B., A New Genetic Approach for the Traveling Salesman Problem, Proceeding of The First IEEE Conference on Evolutionary Computing, 1994, Vol.1, pp.7-12.

[81] Homaifar, A., Guan, S., Liepins, G.E., A New Approach on the Traveling Salesman Problem by Genetic Algorithms, Proceedings. of the Fifth International Conference on Genetic Algorithms, 1993, pp.460-466.

[82] Jog, P., Suh, J.Y., Gacht, D.V., The Effects of Population Size, Heuristic Crossover, and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem, Proceeding of The Third International Conference of Genetic Algorithm, 1989, pp.98-103.

[83] Bramlette, M., Initialization, Mutation and Selection Methods in genetic Algorithms for Function Optimization, The Fourth International Conference on Genetic Algorithms, Virginia, 1991, pp.100-107.

[84] Lozano-Perez, T., Automatic Planning of Manipulator Transfer Movements, IEEE Transaction on SMC, 1981, Vol.11, pp.681-698.

[85] Brooks, R.A., Planning Collision Free Motions for Pick and Place Operation, Robotics Research, MIT Press, 1983, pp.5-38.

[86] Khatib, O., Real-Time Obstacle Avoidance for Manipulators and Mobile Robots, Robotics and Research, 1986, Vol.5, No.1, pp.90-98.

[87] Davidor, Y., A genetic Algorithm Applied to Robot Trajectory Generation, Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991, pp.144-165.

[88] Fukuda, T., Ueyama, T., Cellular Robotics and Micro Robotic Systems, World Scientific Series in Robotic and Automated Systems Vol.10, World Scientific, 1994.

[89] Rahmani, A., Ono, N., A Genetic Algorithm for Channel Routing Problem, The Fifth International Conference on Genetic Algorithms, Virginia, 1993, pp.494-498.

[90] Pu, P., Hughes, J., Integrating AGV Schedules in a Scheduling System for a Flexible Manufacturing Environment, IEEE International Conference on Robotics and Automation, California, 1994, pp.3149-3154.

[91] Macchiaroli, R., Riemma, S., Clustering Methods for Production Planning and Scheduling in a Flexible Manufacturing System, IEEE International Conference on Robotics and Automation, California, 1994, pp.3155-3160.

[92] Thamgiah, S.R., Vinayagamoorthy, R., Gubbi, A.V., Vehicle, Routing with Time Deadlines using Genetic and Local Algorithm, Proceeding of The Fifth International Conference on Genetic Algorithms, 1993, pp.506-513.

[93] Starke, J., Kubota, N., Fukuda, T., Combinatorial Optimization with Higher Order Neural Networks -Cost Oriented Competing Processes in Flexible Manufacturing System, Proceedings of The International Conference on Neural Network, 1995, pp.2658-2663.

[94] Yamada, T., Nakano, R., A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems, Parallel Problem Solving from Nature 2, Elsevier Science Publishers B. V., 1992, pp.281-290.

[95] Cleveland, G. A., Smith, S. F., Using Genetic Algorithms to Scheduling Flow Shop Releases, Proceeding of The Third International Conference on Genetic Algorithms, 1989, pp.160-169.

[96] Holsapple, C., Jacob, V., et al., A Genetics-Based Hybrid Scheduler for Generating Static Schedules in Flexible Manufacturing Contexts, The IEEE of Transaction on System, Man, and Cybernetics, 1993, Vol.23, No.4, pp.953-972.

[97] Kim, G.H., Lee, C.S.G., Genetic Reinforcement Learning Approach to the Machine Scheduling Problem, Proceeding of The International Conference on Robotic and Automation, Vol.1, 1995, pp.196-201.

[98] Wang, J., Luh, P.B., Optimization-Based Scheduling of a Machining Center, Proceeding of The International Conference on Robotic and Automation, 1995, Vol.1, pp.502-507.

List of Publications

Jorunal and Transaction

[99] Baba, N., Kubota, N., Path Planning and Collision Avoidance of a Robot Manipulator Using Genetic Algorithm, Journal of the Robotics Society of Japan, 1993, Vol.11, No.2, pp.299-302, in Japanese.

[100] Kubota, N., Date, T., Fukuda, T., Introduction of Age Structure to Genetic Algorithm and Its Convergence, The Transaction of the Society of Instrument and Control Engineers, 1995, Vol.31, No.5, pp.560-568, in Japanese.

[101] Kubota, N., Fukuda, T., A Study of Dynamically Reconfigurable Robotic System (the 23rd Report, Application of Genetic Algorithm to optimal Location Problem on Self-Organizing Manufacturing System), Transaction of the Japan Society of Mechanical Engineers, 1995, Vol.61, No.592C, pp.4660-4665, in Japanese.

[102] Kubota, N., Fukuda, T., Shimojima, K., Virus-Evolutionary Genetic Algorithm for A Self-Organizing Manufacturing System, Computer & Industrial Engineering Journal, 1996, Vol.30, No.4, pp.1015-1026.

[103] Kubota, N., Shimojima, K., Fukuda, T., The Role of Virus Infection in Virus-Evolutionary Genetic Algorithm, Journal of Applied Mathematics and Computer Science, 1996, Vol.6, No.3, pp.415-429.

[104] Kubota, N., Fukuda, T., Shimojima, K., Trajectory Planning of Cellular Manipulator System Using Virus-Evolutionary Genetic Algorithm, Robotics and Autonomous Systems 19, 1996, pp.85-94.

[105] Shimojima, K., Kubota, N., Fukuda, T., Self-Tuning Fuzzy Controller Based on Virus Theory of Evolution, Transaction of the Japan Society of Mechanical Engineers, 1997, to appear, in Japanese.

Books

[106] Shimojima, K., Kubota, N., Fukuda, T., Virus-Evolutionary Genetic Algorithm for Fuzzy Controller Optimization, Studies in Fuzziness, Vol.8, Genetic Algorithms and Soft Computing, Physica-Verlag, A Springer-Verlag Company, 1996, pp.369-388.

[107] Fukuda, T., Kubota, N., Shimojima, K., Virus-Evolutionary Genetic Algorithm for Traveling Salesman Problem, Evolutionary Computation: Theory and Applications, Xin Yao ed., Singapore: World Scientific Publ. Co., 1997, to appear.

International Conferences

[108] Baba, N., Kubota, N., Collision Avoidance Planning of a Robot Manipulator by Using Genetic Algorithm - A Consideration for the Problem in Which Moving Obstacles and/or Several Robots Are Included in the Workspace, Proceedings of the 1st International Conference on Evolutionary Computation (ICEC'94), 1994-6, pp.714-719.

[109] Kubota, N., Fukuda, T., Arai, F., Shimojima, K., Genetic Algorithm with Age Structure and Its Application to Self-Organizing Manufacturing System, Proceedings of 1994 IEEE Symposium on Emerging Technologies & Factory Automation (ETFA'94), 1994-11, pp.472-477.

[110] Kubota, N., Shimojima, K., Fukuda, T., The Role of Virus Infection in Virus-Evolutionary Genetic Algorithm, Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'96), 1996-5, pp.182-187.

[111] Shimojima, K., Kubota, N., Fukuda, T., RBF Fuzzy Controller with Virus-Evolutionary Genetic Algorithm, Proceedings of the International Conference on Neural Network (ICNN'96), 1996-6, p.1040-1043.

[112] Kubota, N., Shimojima, K., Fukuda, T., Virus-Evolutionary Genetic Algorithm - Ecological Model on Planar Grid, Proceedings of the Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS'96), 1996-6, pp.505-509.

[113] Kubota, N., Fukuda, T., Virus-Evolutionary Genetic Algorithm for Combinatorial Optimization Problem, Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems, 1996-6, pp.118-123.

[114] Kubota, N., Fukuda, T., Shimojima, K., Trajectory Planning of Redundant Manipulator Using Virus-Evolutionary Genetic Algorithm, Proceedings of the Symposium on Robotics and Cybernetics -Computational Engineering in Systems Applications (CESA'96), 1996-7, p.728-733.

[115] Kubota, N., Fukuda, T., Shimojima, K., Trajectory Planning of Reconfigurable Redundant Manipulator Using Virus-Evolutionary Genetic Algorithm, Proceedings of The 22nd International Conference on Industrial Electronics, Control, and Instrumentation, (IECON'96), 1996-8, pp. 836-841.

[116] Kubota, N., Shimojima, K., Fukuda, T., Virus-Evolutionary Genetic Algorithm - Ecological Model on Planar Grid, Proceedings of The Fifth IEEE International Conference on Fuzzy Systems (FuzzIEEE'96), 1996-9, pp.232-238.