HOKKAIDO UNIVERSITY

Title	A Study on Genetics-based Adaptive Problem Solver
Author(s)	Kawakami, Takashi; 川上, 敬
Degree Grantor	北海道大学
Degree Name	博士(工学)
Dissertation Number	甲第3849号
Issue Date	1996-03-25
DOI	https://doi.org/10.11501/3111974
Doc URL	https://hdl.handle.net/2115/51305
Type	doctoral thesis
File Information	000000297165.pdf



Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

A STUDY ON GENETICS-BASED ADAPTIVE PROBLEM SOLVER

Takashi KAWAKAMI

A Study On Genetics-Based Adaptive Problem Solver

by

Takashi KAWAKAMI

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Engineering

Hokkaido University

December 1995

Contents

Contents i

List of Figures vi

List of Tables ix

Chapter 1

Introduction 1

Reference of Chapter 1 7

Chapter 2

Definitions and Characteristics of Genetics-based Adaptive Problem Solvers 8

2.1 Introduction 8

2.2 What is Problem Solving? 9

2.3 Survey of Conventional Problem Solving Methods 10

2.3.1 Newell and Simon's Human Problem Solving and GPS 10

2.3.2 Traditional AI Planning Techniques 16

2.4 General Formalization of GAPS 20

2.4.1 Definitions of the Problem Space and the Problem Solver 20

2.4.2 Definitions of Adaptive Problem Solvers 24

2.4.3 Genetics-based Adaptive Problem Solvers 32

2.5 Genetic Algorithms as Adaptive Evolutionary Computation Scheme 36

2.5.1 An Overview of Genetic Algorithms 36

2.5.2 Definitions of Simple Genetic Algorithms 39

2.6 Classifier Systems as Adaptive Reinforcement Learning Scheme 43

2.6.1 An Overview of Classifier Systems 43

2.6.2 Formalization of Simple Classifier Systems 44

2.7 Required Robustness of GAPS in Uncertain and Non-Stationary Problem Domains 52

2.8 Conclusions 55

Reference of Chapter 2 56

Chapter 3

Extended Mechanisms for Developing the Advanced GAPS 59

3.1 Introduction 59

3.2 Some Difficulties on Implementation of Simple GA and CS 61

3.2.1 Focused Issues on Genetic Algorithms 61

3.2.2 Key Issues on Classifier Systems 63

3.3 Evolutionary Synthesis of CS Architectures by G A 67

3.3.1 Introduction 67

3.3.2 Related Works 69

3.3.3 Definitions of Simple Classifier Systems to be Evolutionary Synthesized 70

3.3.4 Evolutionary Architecture Synthesis Process 73

3.3.4.1 The Evolutionary Synthesis Process 74

3.3.4.2 Representation of a Chromosome 75

3.3.4.3 Evaluation of Simple Classifier System Architectures 79

3.3.5 Experimental Verification 80

3.3.5.1 Motion Planning Problems of a Robot Manipulator 80

3.3.5.2 Implementation of Simple Classifier Systems 82

3.3.5.3 Applying Genetic Algorithms to the Architecture Synthesis Process 84

3.3.5.4 Results and Analysis 86

3.3.6 Concluding Remarks 93

3.4 Extensive Rule Representation in CS by Masking Classifiers 95

3.4.1 Introduction 95

3.4.2 Representational Difficulties 96

3.4.3 Definitions of Masking Classifiers 99

3.4.3.1 Representations 100

3.4.3.2 The Masking Operation 101

3.4.3.3 The Conditional Matching and Biding Mechanisms 102

3.4.3.4 Revision of Strength Values 102

3.4.3.5 The Discovery of Masking Classifiers 103

3.4.4 Numerical Experiments 103

3.4.4.1 Block Stacking Problems 103

3.4.4.2 Implementation Methods 105
3.4.4.3 Experimental Results 107
3.4.5 Conclusions 112
Reference of Chapter 3 113

Chapter 4

Engineering Applications by the GAPS 118
4.1 Preliminary 118
4.2 The Development of Adaptive Tuning Systems of 3-Dimensional Packing Strategies 120
4.2.1 Introduction 120
4.2.2 Related Works 123
4.2.3 Description of the Three-Dimensional Packing Problem 124
4.2.4 Hybrid Adaptation Method in Single Container Environments 126
4.2.4.1 The 3-Dimensional Packing Strategy 126
4.2.4.2 Evaluation Function to Select an Allocation Position 127
4.2.4.3 Evaluation Function to Select a Box 128
4.2.4.4 Altering the Allocation Space 129
4.2.4.5 The Termination Conditions 130
4.2.4.6 Implementing the Genetic Algorithms 130
4.2.5 Development of a Three-Dimensional Packing Rule-base 132
4.2.5.1 Extraction of the Feature Information 133
4.2.5.2 The Conditional Matching 134
4.2.6 Hierarchical Adaptation of 3-D Packing Strategies in Multiagent environments 135
4.2.6.1 The Extended Problem Setting 135
4.2.6.2 An Outlines of Hierarchical Adaptation of 3-Dimensional Packing Strategies 137
4.2.6.3 The Agent Packing Strategy 138
4.2.6.4 The Supervisor Strategy 138
4.2.6.5 Implementation of Genetic Algorithms to the Hierarchical Adaptation 138
4.2.7 Numerical Experiments 142
4.2.7.1 Simulation Results in Single Container Environments 142

4.2.7.2 Experimental Results of the Constructed 3-Dimensional Packing Rule-base....145
4.2.7.3 Numerical Experiments in Multiagent Environments....147
4.2.8 Conclusions....150
References....152
4.3 The Autonomous Robot Navigator with Classifier Mechanisms....155
4.3.1 Introduction....155
4.3.2 The Autonomous Robot Navigation Problem....157
4.3.3 Implementing the Classifier System....159
4.3.4 Experimental Results....161
4.3.4.1 Experiments in Single Agent Domains....161
4.3.4.2 Experiments in Multiagent Domains....166
4.3.5 Discussions....168
4.3.6 Conclusions....169
References....169
4.4 An Approach to Sequencing Problems in Process Planning....171
4.4.1 Introduction....171
4.4.2 Problem Setting....173
4.4.3 An Approach with Genetic Algorithms....176
4.4.3.1 Representations....176
4.4.3.2 A Reproduction Operator....177
4.4.3.3 Further Genetic Operators....178
4.4.4 An Approach Using Classifier Systems....178
4.4.4.1 The Encoding Method of Processing States....179
4.4.4.2 The Decoding Mechanism of Classifiers....180
4.4.4.3 A Reinforcement Method of Classifiers....180
4.4.5 GA-Based Seeding Mechanism of Initial Rules in Classifier Systems....180
4.4.6 Experimental Results....182
4.4.6.1 Experimental Tasks....182
4.4.6.2 Experiment 1....183
4.4.6.3 Experiment 2....185
4.4.6.4 Experiment 3....186
4.4.7 Conclusion....188

References 189
4.5 Autonomous Acquisition of Cooperative Robot Task Plans in Multiagent Environments 190
4.5.1 Introduction 190
4.5.2 Robot Task Planning Problem in Multiagent Environments 191
4.5.3 Implementation Method 192
4.5.4 Numerical Experiments 195
4.5.5 Conclusions 197
References 198
4.6 An Approach to Reactive Motion Planning Problems of Robot Manipulators 200
4.6.1 Introduction 200
4.6.2 Motion Planning Tasks of Robot Manipulators 201
4.6.3 Implementing Learning Classifier Systems 204
4.6.4 Experimental Results 207
4.6.4.1 Simulation 1: a 2-link planar manipulator in the unknown workspace 208
4.6.4.2 Simulation 2: 4-link manipulator experiment 212
4.6.5 Conclusions 214
References 214

Chapter 5

Summary 215

Acknowledgments 221
List of Results 222

List of Figures

Figure 2.1 General structure of an information processing system.....12
Figure 2.2 A brief chronology of some planning systems.....18
Figure 2.3 The generated complete problem space in conventional problem solvers embedded in problem solving agents.....21
Figure 2.4 A problem space made from given well-defined problem. 22
Figure 2.5 Schematic view of the APS and its problem solving mechanisms.....30
Figure 2.6 The planning cycle of SMPA model.....31
Figure 2.7 A process of the reactive planning. 31
Figure 2.8 Evolving populations by genetic algorithms. 38
Figure 2.9 The executive cycle of genetic algorithms.....38
Figure 2.10 A schematic illustration of the original classifier system.....44
Figure 2.11 Functional organization of a simple classifier system. 45
Figure 2.12 A basic sense-act cycle in the performance system. 48
Figure 3.1 The evolution cycle of Simple classifier system architectures by GA..74
Figure 3.2 The encoding process by EF and the decoding process by DF.....77
Figure 3.3 A GA-encoded parameters. 79
Figure 3.4 Given experimental tasks of 2-link planar manipulator motion planning. 81
Figure 3.5 An example of encoding and decoding method. 83
Figure 3.6 Results of evolutionary architecture synthesis experiments by GA. ...87
Figure 3.7 Resultant motion plans for given experimental tasks.....87
Figure 3.8 Learning behaviors of classifier system architectures made from six typical combinations.....90
Figure 3.9 Applicabilities of learned classifier systems for a similar task.....93
Figure 3.10 A schematic view of the masking operation.....102
Figure 3.11 Given experimental tasks.....105
Figure 3.12 A direct representation method of a state of blocks.....106
Figure 3.13 Learning curves of the masking classifier system for given experimental tasks.....108
Figure 3.14 Strength values of classifiers, in the learned system for example 1 over 200 learning trials.....109

Figure 3.15 Revised connective strength values in the learned system for example 1 over 200 learning trials. 109
Figure 3.16 A relatively complex task. 111
Figure 3.17 The learning curve of the masking classifier system for the task example 3. 111
Figure 4.1 The 3-D Packing Strategy 127
Figure 4.2 Altering a set of allocable positions. 129
Figure 4.3 A conceptual flow of a 3-D packing rule-base 132
Figure 4.4 The 3-D packing problem in multiagent environment 136
Figure 4.5 The outline of a proposed approach 137
Figure 4.6 Graphical output of the resultant configuration of data 8 144
Figure 4.7 Growth of the packing performances for three typical input data 144
Figure 4.8 The behavior of the population under evolution for the data 1 145
Figure 4.9 The relationship between the feature distance and the packing performances 146
Figure 4.10 Packing performances of the constructed 3-D packing rule-base. 146
Figure 4.11 The volume distribution of given experimental boxes. 148
Figure 4.12 Simulation results of the supervisor strategy tuning by the GA search and a random search. 149
Figure 4.13 An autonomous robot searches the shortest path from starting position to destination position. 158
Figure 4.14 The credit distribution of environmental rewards 161
Figure 4.15 The given robot navigation tasks. 162
Figure 4.16 The learning curves of typical five navigation tasks 163
Figure 4.17 The learning results by different combinations of learning plans. 164
Figure 4.18 Learning results of task E with evolved classifiers that are previously learned by sub-problems. 165
Figure 4.19 Learning times to converge for randomly generated additional 50 tasks by using the learned classifier system. 165
Figure 4.20 Experimental navigation results in a multiagent environment 167
Figure 4.21 The varying behavior of strength values of two typical classifiers in the agent 2 during this learning process is achieving. 167
Figure 4.22 A part *P* and a decomposition of removal volume *Q*. 174

Figure 4.23 A proposed autonomous operation planning system integrated GA approach and CS approach. 182
Figure 4.24 Three-dimensional and two-dimensional views of the part 1 with decomposed elementary volumes 183
Figure 4.25 Search results of given three planning tasks by GA approach. 184
Figure 4.26 The behavior of the population while GA evolution. 184
Figure 4.27 The learning curves of three planning tasks by CS approach 186
Figure 4.28 Strength values of classifiers after learning the task of part 1. 186
Figure 4.29 Learning efficiencies for different learning types (the part 1). 186
Figure 4.30 Learning curves of Part 1 using some special classifiers generated from the tuned plan by GA approach. 187
Figure 4.31 Two-dimensional view of a part 4 that is similar as a part 1. 188
Figure 4.32 The improvement of learning performances through accumulation of knowledge of a similar problem. 188
Figure 4.33 A proposed representation method of the state of blocks. 194
Figure 4.34 State transitions of blocks by alternative operations of multiple robots. 194
Figure 4.35 The operation of the bucket brigade algorithm. 195
Figure 4.36 Given experimental tasks. 196
Figure 4.37 Learning curves of given experimental tasks. 196
Figure 4.38 Given additional experimental task 197
Figure 4.39 Learning curves of example 3. 197
Figure 4.40 A simplified model of robot manipulators. 203
Figure 4.41 The motion control of a manipulator. 203
Figure 4.42 An implementation method. 205
Figure 4.43 An encoding procedure and a decoding procedure. 206
Figure 4.44 A reinforcement strategy. 207
Figure 4.45 Experimental results for task 1. 209
Figure 4.46 Experimental results for task 2. 210
Figure 4.47 Experimental results for task 3. 211
Figure 4.48 Simulation results of 4-link manipulator experiment. 213

List of Tables

Table 2.1 Significant features of some domain-independent planning systems. 19

Table 3.1 Some extensive approaches on genetic algorithms 62

Table 3.2 Some extensive approaches on classifier systems 66

Table 3.3 Prepared candidate values for each parameter and the length of encoded binary sub-string in a chromosome. 85

Table 3.4 GA-parameters used in the evolutionary synthesis process 86

Table 3.5 Learning results of some typical combinations of parameters. 89

Table 3.6 Affections of parameter values of $P_{\#}$ for the learning performance of classifier systems. 91

Table 3.7 Experimental results about convergencies of classifier system architectures 92

Table 3.8 Experimental parameters 108

Table 3.9 Obtained simulation results of four types of classifier systems 110

Table 4.1 Experimental GA parameters 143

Table 4.2 Given experimental input data and those simulation results. 143

Table 4.3 Applied GA parameters 147

Table 4.4 The given experimental data. 147

Table 4.5 Dimensions of given containers and individual tuned results 148

Table 4.6 Experimental results of supervisor strategy tuning in multiagent environment. 150

Table 4.7 The influence of the number of profit shared rules at a time for the learning performance. 162

Table 4.8 Given experimental parts. 182

Table 4.9 Experimental GA parameters. 184

Table 4.10 Experimental parameters in simulation 1. 208

CHAPTER 1

Introduction

This thesis describes about Genetics-based Adaptive Problem Solvers. To realize more flexible and intelligent engineering systems, it is expected that a lot of difficult problems would be autonomously solved. However, the problem solving of many practical subjects is known as hard works. Until today, the Problem Solving has been treated as the one of most important subjects by many researchers in artificial intelligence research field. The problem solving behavior of human has also been tried to clarify in cognitive science research domain. Those facts mean that there are many complex problems should be automatically solved by the machine intelligence in several practical areas. To realize these automatic problem solving mechanisms, modeled human information processing procedures were utilized. As we know, however, a traditional definition of human problem solving, i.e., logical symbolic manipulation method, can be successfully applied only limited problem area. Especially, only the effectively structured problem, that is called well-defined problem, can be solved by that method. Therefore, it is difficult to solve the problems in uncertain and/or non-stationary environments using conventional approaches. To overcome these difficulties, many additional AI mechanisms, methodologies and techniques have been proposed. Nevertheless, these dependencies on the defined problem domain are not resoled by those approaches based on the logical symbolism.

The other hand, Evolutionary Computation is going to be the major research field in the Artificial intelligence area. Concepts of evolutionary computation, then, tend to appeal to not only computer scientists but also evolutionary biologists who are interested in natural evolutional process and biochemists who apply evolutional optimization methods. Evolutionary computation is a new approach of simulating evolution on a computer to realize a machine intelligence. The majority of research in artificial intelligence has simulated symptoms of intelligent behavior as observed in humans. In evolutionary computation, contrastingly, intelligence is defined as the capability of a system to adapt its behavior to achieve its goals in a range of environments. Rather than focus on human beings and attempt to model human behaviors and cognitive processes, it would be more important to recognize that we can create entities capable of generating intelligent behavior by modeling evolutionary processes. Evolutionary computation, therefore, may provide the basis for realizing a new viewpoint of machine intelligence.

Since intelligence is viewed in terms of adaptation, the functional behavior of systems should be evolved. To construct a useful model, an evolution process must be abstracted in any manner. The result of such modeling of evolutionary computation is a number of adaptive optimization algorithms that rely on very simple rules. This adaptive optimization process iteratively improves the quality of these solutions. Generally, the procedures converge to near-optimal solutions despite of the existence of topological path in search space. These methods offer potential for addressing engineering problems that have resisted solution by classic techniques. Moreover, more important fact is that, these methods may be used to address a broad range of problems, rather than any one specific problem. They have proven themselves to be robust and may be applied toward general problem solving. This latter attribute represents the greatest potential for evolutionary computation. The real promise of evolutionary computation, however, remains mostly unfulfilled.

Also, Machine Learning is well known as the effective technique to realize intelligent systems and to improve the performance of those systems. Many AI researchers have been interested in various machine learning

paradigms from 1950s. A major objective of machine learning is the knowledge refinement and adjustment in a set of primarily acquired data, rules, and knowledge. For this purpose, lots of approaches, e.g., form logic-based theoretical algorithms to explanation-based or case-based reasoning, were applied. In this framework, some limitations are assumed. Thus, treated problem domains are static and all of required knowledge must be given. However considering certain robotics problems, all of state variables in a problem environment cannot be acquired. Capabilities of uncertainty management and dealing with dynamic environments must be embraced. Reinforcement Learning, that is one of machine learning paradigms, is recently tend to be focused to solve the uncertainty management problems. Therefore, the reinforcement learning scheme is very useful tool to construct an adaptive problem solving system.

So, this thesis is an attempt to integrate the potentials of evolutionary computation, reinforcement learning, and the engineering problem solving. This integration is realized as the **Genetics-based Adaptive Problem Solver** (**GAPS**). As the useful tools, especially, Genetic Algorithms as adaptive evolutionary computation scheme and Classifier Systems as adaptive machine learning mechanism are employed to construct adaptive problem solving systems. These implements have great advantages for adaptive problem solving. There, however, are some potential problems on implementation of simple GA and CS. Consequently, extended functions of GAPS are also discussed for developing the advanced GAPS.

Recently, the subjective problem class of problem solving systems in the artificial intelligence research field is changing from a class type of classifying problems such as diagnosis problems to a class type of synthesizing problems such as planning problems and designing problems. In other words, research subjects are transformed from the relatively easy problem class to the more difficult problem class. Therefore, the subjective problem class in this thesis is synthesizing type problems, especially, planning problems.

The contents of remaining chapters are summarized as followed.

I start in Chapter 2 with definitions and characteristics of Genetics-

based Adaptive Problem Solvers(GAPS). Some conventional problem solving methods are reviewed to clarify research activities in this field: from cognitive scientific approaches such as Newell and Simon's Human Problem Solving, to knowledge-based expert systems and AI planning systems, those are called domain-dependent approaches. I then general formalization of GAPS is defined. Required robustness of GAPS in uncertain and non-stationary problem domains is also discussed. Mechanisms of Genetic Algorithms as evolutionary computation scheme and Classifier Systems as adaptive reinforcement learning scheme are summarized.

In Chapter 3, I propose extended mechanisms in GA and CS to develop the advanced GAPS, because simple GA and CS have some latent difficulties on implementation. Proposed extending mechanisms are as followed;

1. Synthesis of CS architectures by GA.
2. Extension of rule representations in CS by masking classifiers.

Engineering applications of GAPS are presented in Chapter 4. In this chapter, I have chosen some variety of problem domains to examine the performances of GAPS, those are optimization problems, planning problems, and scheduling problems. Definitely, five practical problems, i.e. 3-Dimensional Packing Problems, Autonomous Mobile Robot Navigation Problems, Process Sequencing Problems, Robot Task Planning Problems, and Manipulator Motion Planning Problems, are solved by GAPS in following manners;

- Firstly, 3-D packing strategies are evolved by adaptive tuning systems in which the hybrid tuning method of GA and heuristics is applied. In multiple containers environments, hierarchical tuning mechanisms are also proposed. Based on evolved packing strategies, the possibility of development of 3-D packing rule-base systems is discussed.
- Autonomous Robot Navigator with the Classifier Mechanism is also proposed. In this application, I attempt to flexibly solve the mobile robot navigation problem in an arbitrary environment. In order to realize an autonomous navigation system, I adopt a classifier mechanism. The autonomous robot that individually has a classifier system as the

navigation system, is called an Agent. Thus, multiagent environments, in which multiple mobile robots exist in one navigation area, are also treated. Each agent induces an individual optimal solution for a given navigation task. The given navigation area is defined as a two-dimensional maze space that is suitable for representation of factories or offices.

- An approach to Sequencing Problem in Process Planning is described. In general process planning for a designed machine part involves generating a set of plans that outline operations, machine tools, fixtures, and tools required to produce that part. Based on the design specifications provided, a process planner has to determine the process plan under consideration of minimizing a production cost, and at the same time maximizing rate of production and quality of a part. Generally speaking, the task of the process planning is decomposed into several phases. It is very difficult to solve the sequencing phase of machinable volumes since the optimal sequence of operations must be selected from many combinations of operations that satisfying the process constraints. Furthermore, this phase largely affects the processing time. Hence, solving this phase is one of the most important problem for developing the autonomous process planning system. In order to realize an autonomous planning mechanism, the two approaches are performed. First, the Genetic Algorithms are applied directly. As the second approach, the Classifier System is applied to this problem to obtain the further advantages.
- Then, Robot Task Planning Problems are solved by GAPS approaches. As the one of robot task planning problems, the block stacking problem is treated. To realize an autonomous planning mechanism, classifier systems are applied to this problem. In particular, the difficulties of this problem are increased in multiagent environment where multiple robots exist in a problem domain. In these multiagent systems, multiple autonomous robots achieve given tasks and cooperate with each other to effectively solve the task that is difficult or is never performed by single

robot. Research interests on such autonomous distributed robot systems are related to interactions among agents. In other words, the important problem is the trade off between the autonomy of each agent and the adjustment of whole system. In this problem domain, I attempt to realize a mechanism by which cooperative strategies of agents can be acquired.

- As a final application, an approach to Reactive Motion Planning Problems of robot manipulators is shown. The motion planning problem of a robot manipulator is well known that how to get the solution of this problem is said to be the one of the most difficult problem in robotics and artificial intelligence fields. In this application, I treat this problem under uncertain constraints where unknown obstacles exist in a problem domain and reactive plans have to make based on sensed environmental information. This reactive planning mechanism is realized by trial-and-error process based on classifier systems. Required resolutions of sensors by which environmental information and problem state are obtained, are also discussed.

Finally, Chapter 5 describes concluding remarks of contributions of this thesis.

References of Chapter 1

[Adeli and Hung 95] Adeli, H. and Hung, S.: "Introduction", Machine learning: neural networks, genetic algorithms, and fuzzy systems, John Wiley & Sons, pp.1-4, 1995.

[Beer 90] Beer, R.D.: "Foundations", Intelligence as adaptive behavior, Academic-Press, pp.1-20, 1990.

[Fogarty 94] Fogarty, T.C.(ed.): "Evolutionary computing", Springer- Verlag, 1994.

[Fogel 95] Fogel, D. B.: "Defining artificial intelligence", Evolutionary computation: Toward a new philosophy of machine intelligence", IEEE Press, pp.1-36, 1995.

[Goldberg 89] Goldberg, D.E. :"Genetic Algorithms in Search, Optimization & Machine Learning", Addison-Wesley, pp.1-25, 1989

[Holland 75] Holland, J.H. :"Adaptation in Natural and Artificial Systems", University of Michigan Press, pp.20-120, 1975.

[Holland et al. 86] Holland, J.H., Holyoak, K.J., Nisbett, R.E. and Thagard, P.R. :"Induction", The MIT Press, pp.102-150, 1986.

[Kodratfoff 88] Kodratfoff, Y.: "Introduction of machine learning", Pitman, pp.1-41, 1988.

[Michalewicz 92] Michalewicz, Z.: " Introduction", Genetic algorithms + Data structures = Evolution programs, Springer-Verlag, pp.1-10, 1992.

[Newell and Simon 72] Newell, A. and Simon, H.A.: "Human problem solving", Prentice-Hall, pp.1-140, 1972.

[Sebald and Fogel 94] Sebald, A.V. and Fogel, L.J. (ed.): "Proceedings of the third annual conference on Evolutionary Programming", World Scientific, 1994.

[Uehara et al. 95] Uehara, K., Tsuji, H., and Ajisaka, T. (ed.): "Special issue on recent trends in knowledge acquisition and machine learning", Journal of Systems, Control and Information, Vol.39, No.4, pp.159-197, 1995. (in Japanese)

[Veloso 94] Veloso, M.M. :"Introduction", Planning and learning by analogical reasoning, Springer-Verlag, pp.1-14, 1994.

CHAPTER 2

Definitions and Characteristics of Genetics-based Adaptive Problem Solvers

2.1 Introduction

In the decade of the 1980s, artificial intelligence created a lot of success and excitement. The most significant and widespread outcome of artificial intelligence research was the development of knowledge-based expert systems. However, many hard practical problems called ill-defined problems or ill-structured problems cannot be solved by those knowledge-based expert systems. Generally, an ill-defined problem means a problem in which it is difficult to previously specify the problem domain. Another hand, an ill-structured problem implies that the effective problem solving method for that problem cannot be determined beforehand. As a result, research interests about intelligent systems are transferred from analytic problem domain such as diagnosis and classification, to synthetic problem domain such as planning and design in the 1990's. Almost of problems in synthetic problem domain are ill-structured and NP-complete. Therefore, it is expected to appear the new problem solving paradigm. To make an answer of this expectation, some paradigms are gaining momentum. So, I focus on adaptation and learning, and try to provide the adaptive problem solving system on the basis of genetics.

In this Chapter, I define the Genetics-based Adaptive Problem

Solvers(GAPS) and clarify characteristics of GAPS. For this purpose, an outline of problem solving is described in section 2.2, and some conventional problem solving methods are reviewed in section 2.3. Section 2.4 formalizes the adaptive problem solving and defines the GAPS. Mechanisms of Genetic Algorithms as evolutionary computation scheme and Classifier Systems as adaptive reinforcement learning scheme are summarized in section 2.5 and section 2.6 respectively. Required robustness of GAPS in uncertain and non-stationary problem domains is discussed in section 2.7. Section 2.8 concludes this chapter.

2.2 What is Problem Solving?

The phenomenon of problem solving is a main research subject in the artificial intelligence field. This is considered as processes of discovering and generating solutions. The problem solver manages these processes and performs procedures of problem solving. The main object of artificial intelligence research is development of the problem solver that has good computational performance and is valid for a broader area of problems. To assign a problem to a problem solving system, usually, the following general problem statement is transmitted to the problem solving system.

> "Given a problem domain description *D*, find the solution *x* that is a element of a set of feasible solutions *X*, and satisfies problem conditions *C*."

In this problem statement, feasible solution set *X* and problem conditions *C* are represented by concepts of description *D*. If a problem solving system can explain information of *D*, *X* and *C*, and can effectively exploit these information, then the system can understand given problem statement and search procedures of solutions are able to be achieved.

In general, a feasible solution set *X* is not given explicitly, but *X* is formed by generating solution elements *x* individually. The problem condition *C* provides restrictions of a searched solution *x* to be satisfied. The problem domain description *D* is an aggregate of basic knowledge about the problem domain. The domain description *D* contains a class of object type, forms of predicates and a nature of a task environment about a problem specification.

For a problem solving system, if a judgment processing procedure by which feasibility of a searched solution is estimated, is included in the proposed problem statement, then this problem statement represents a well-defined problem. In a well-defined problem, the problem solving system has to know complete information of X and C in a given problem statement.

Generally speaking, most of past AI researches treated the problem solving processes of well-defined problems. However, it is often happened that the complete information is not given to systems. In that case, appropriate D and X are not exhibited, or only an incomplete information of those is presented at first stage of problem solving process. Therefore, the problem solving system must have any mechanism that makes a solvable problem from the incomplete statement.

General concepts of problem solving theories are firstly defined by Newell and Simon[Newell and Simon 72]. I, therefore, describe more detailed explanations of general terms of problem solving in following sections of historical reviews.

2.3 Survey of Conventional Problem Solving Methods

2.3.1 Newell and Simon's Human Problem Solving and GPS

At the earlier period of the problem solving research, some important researches were achieved in a cognitive science field. These researches greatly affected to artificial intelligence researches. Especially, Newell and Simon's work[Newell and Simon 72] is full of suggestions even now. The definitions of many general and common concepts about the problem solving are also converged on their study. Therefore, I review details of the work in this section.

The aim of their research is to clarify how humans think. In their study, a phenomenon of problem solving is considered as relationships between task environments and information processing systems.

The Information Processing System

Their theory proclaims man to be an information processing system, when he is solving problems. The theory posits a set of processes or mechanisms that produce the behavior of the thinking human. That is, the theory is reductionistic, and it does not simply provide a set relations or laws about behavior from which one can often conclude what behavior must be. For the purpose of their work, first, an information processing system is defined, and the organization and function of the information processing system are described. Figure 2.1 shows the general characteristics of their system. The system consists of a memory containing symbol structures , a processor, effectors, and receptors. They, then, described a number of interrelated definitions and postulations as follows;

1. There is a set of elements, called symbols.
2. A symbol structure consists of a set of instances of symbols connected by a set of relations.
3. A memory is a component of an information processing system capable of storing and retaining symbol structures.
4. An information process is a process that has symbol structures for its inputs or outputs.
5. A processor is a component of an information processing system consisting of:
   (a) a fixed set of elementary information processes;
   (b) a short term memory that holds the input and output symbol structures of the elementary information processes;
   (c) an interpreter that determines the sequence of elementary information processes to be executed by the information processing system as a function of the symbol structures in a short term memory.
6. A symbol structure designates an object if there exist information

processes that admit the symbol structure as input and either:

(a) affect the object; or

(b) produce, as output, symbol structures that depend on the object.

7. A symbol structure is a program if:

(a) the object it designates is an information process; and

(b) the interpreter, if given the program, can execute the designated process.

8. A symbol is primitive if its designation is fixed by the elementary information processes or by the external environment of the information processing system.

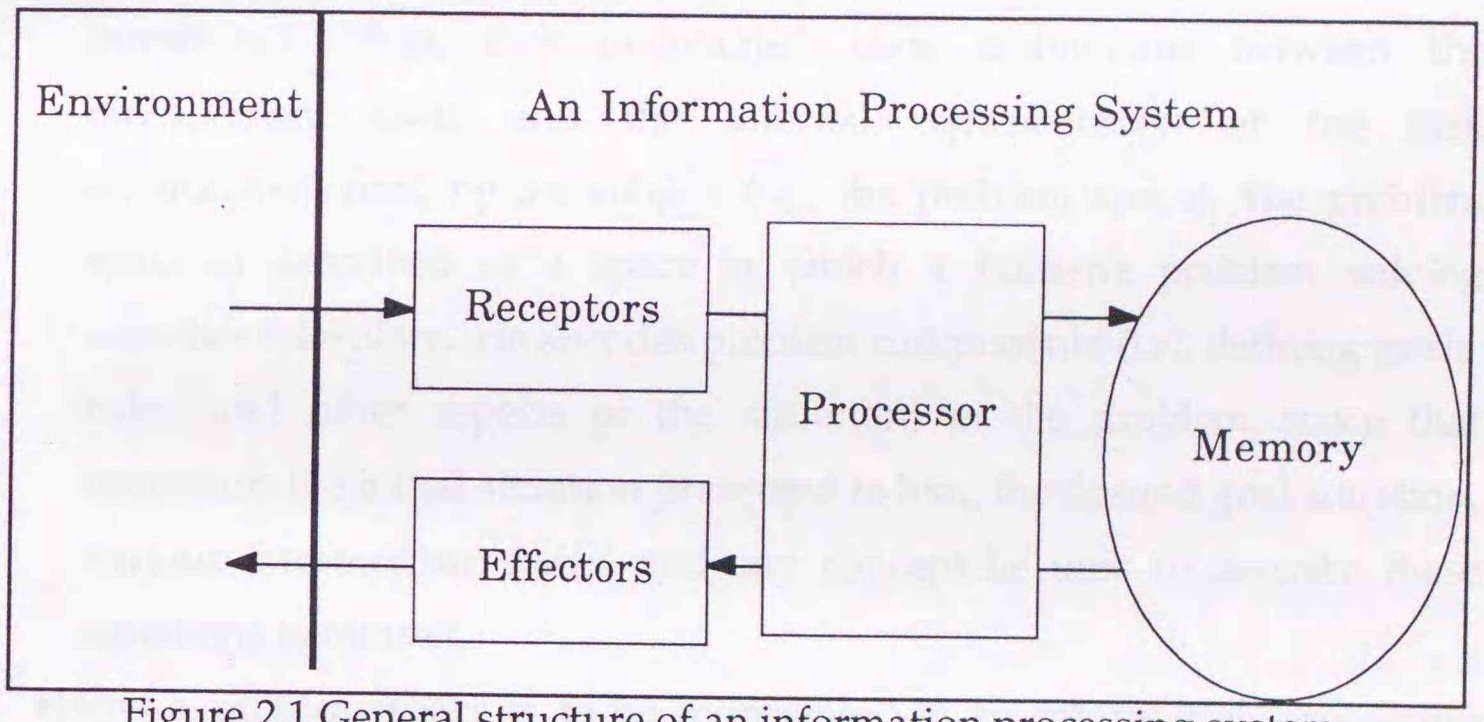


Figure 2.1 General structure of an information processing system.

(from [Newell and Simon 72])

In above definitions, the term *object* is used to encompass following sort of things;

1. Symbol structures stored in one or another memory, which are often usefully classified into (a) data structures, and (b) programs;
2. Processes that the information processing system is capable of executing;
3. An external environment of sensible stimuli.

Based on the above definitions, more detailed characteristics of symbols, symbol structures, designation, primitive symbols, programs, and several processes are defined. They, also described the production system to illustrate the notation of symbol structures.

The Task Environment

Then, the nature of the task environment and its role in a psychological theory is discussed. To define the task environment, the following things are discussed.

- How the external environment is to be represented?
- How the subject represents a problem internally?

  As a solution to answer the above questions, a concept of problem space is introduced. Thus, they maintained clear distinctions between the environment itself, and the internal representation of the task environment used by the subject (i.e., the problem space). The problem space is described as a space in which a human's problem solving activities take place. He encodes problem components (i.e., defining goals, rules, and other aspects of the situation) in the problem space that represents the initial situation presented to him, the desired goal situation, various intermediate states, and any concept he uses to describe these situations to himself.

- How a problem space is to be represented in an information processing system?
- How the processes of the information processing system operate on such a representation to solve problems?

To clarify a more general characterization of problem space, two broad classes of problem representations: i.e., set representations and search representations. By the set representation method, a problem is characterized as follows:

> *Given a set* **U**, *to find a member of a subset of* **U** *having specified properties (called the goal-set,* **G**),

where, **U** is a set of all solutions. This representation method is useful

when a well-defined problem is given, and when all possible symbolic expressions of solutions are possible to define. In some case where a problem is characterized set-theoretically, the problem can be decomposed into a number of subproblems, and each of which can be characterized in the same way. The set representation of a problem, while it is relatively general, is not always the most convenient characterization. To construct an alternative in the set-theoretic framework, the search representation of a problem is also proposed.

When a complete set U is not given, but an initial solution u_0 and any operator Q_i by which another solution u_i can be generated based on u_0, the search representation method is useful. In this representation method, generated solutions are defined as follows:

$$u_t = Q_t(Q_{t-1}(.....Q_1(u_0)).....) \qquad (2\text{-}3\text{-}1)$$

where, $u_i = Q_i(u_{i-1})$ means the solution obtained from u_{i-1} by applying to it the operator Q_i, and where u_0 is the solution given initially.

Problem Solving Procedure as Interactions Between Task Environments and Information Processing Systems

In their theory, then, the problem solving procedure is illustrated as interactions between task environments and information processing systems mentioned above. Thus, the problem solving process is performed as follows:

1. An initial process, called the input translation, produces inside the problem solver an internal representation of the external environment, at the same time selecting a problem space. The problem solving then proceeds in the framework of the internal representation.
2. Once a problem is represented internally, the system responds by selecting a particular problem solving method. A method is a process that bears some rational relation to attaining a problem solution, as formulated and seen in terms of the internal representation.
3. The selected method is applied. It comes to control the behavior, both internal and external, of the problem solver. At any moment, as the outcome either of processes incorporated in the method itself or of more general processes that monitor its application, the execution of

the method may be halted.

4. When a method is terminated, three options are open to the problem solver: (a) another method may be attempted, (b) a different internal representation may be selected and the problem reformulated, or (c) the attempt to solve the problem may be abandoned.
5. During its operation, a method may produce new problems, i.e., subgoals, and the problem solver may elect to attempt one of these. The problem solver may also have the option of setting aside a new subgoal, continuing instead with another branch of the original method.

General Problem Solver (GPS)

On the basis of their problem solving theory defined above, General Problem Solver (GPS) is developed as the first problem solving program using the problem reduction schema. GPS obtained its name because it was the first problem solving program to separate in a clean way a task independent part of the system containing general problem solving mechanisms from a part of the system containing knowledge of the task environment. GPS operates on problems that can be formulated in terms of objects and operators. An operator is something that can be applied to certain objects to produce different objects. The objects can be characterized by the features they possess, and by the differences that can be observed between pairs of objects. Operators for a given task are restricted to apply only to certain kinds of objects; and there exist operators that apply to several objects as inputs, producing one or more objects as outputs.

By GPS, various kinds of problems can be formulated in a task environment containing objects and operators, by which a given object is transformed into another; an object possessing a given feature is found; an object is modified so that a given operator may be applied to it; and so on. To specify problems and subproblems, GPS has a discrete set of goal types. The main methods of GPS are the means-ends analysis and the reduction method of differences between objects. These methods seem to be a hierarchical planning strategy that generates a sequence of operations to achieve a given goal from an initial state. That, then, becomes the general approach used many other

succeeding planning programs

2.3.2 Traditional AI Planning Techniques

Until today, many problem solving systems are developed based on the Newell and Simon's "Human Problem Solving Theory" mentioned before section. In these problem solving systems, lots of problem domains are treated. However, as I described in chapter 1, the subjective problem class of problem solving systems is changing from a class type of classifying problems such as diagnosis problems to a class type of synthesizing problems such as planning problems and designing problems. Also, my subjective problem class is synthesizing type problems, especially, planning problems. In this section, therefore, some traditional AI planning techniques are reviewed.

Planning problems have been attacked in two major ways: approaches which try to understand and solve the general problem without use of domain specific knowledge, and approaches which use domain heuristics directly. In planning, these are often referred to as domain-dependent approaches which use domain-specific heuristics to control the planner's operation, and domain-independent approaches in which the planning knowledge representation and algorithms are expected to work for a reasonably large variety of application domains. The issues involved in the design of domain-dependent planners are those generally found in applied approaches to AI; i.e., the need to justify solutions, the difficulty of knowledge acquisition, and the fact that the design principles may not map well from one application domain to another. Work in domain-independent planning has formed the bulk of the Al research in the area of planning. Figure 2.2 illustrates a brief chronology of some well-known planning systems. In this figure, lines represent influences to develop planning systems. The long history of these efforts has led to the discovery of many recurring problems, as well as to certain standard solutions. In addition, there have been a number of attempts to combine the planning techniques available at a given time into prototypes able to cope with increasingly more realistic

application domains. Table 2.1 shows an example of some of these efforts and the domains to which they were applied.

As I reviewed here, planning systems have been an active research topic within Artificial Intelligence for nearly thirty years. There have been a number of techniques developed during that period which still form an essential part of many of today's AI planning systems. However even now, there exist many un-solved problems in the planning field. These problems are not enough analyzed, or are going to be solved by advanced techniques. Types of these problems may be classified as follows;

1. A problem in which a planner knows only partial information of environment. Therefore, the planner must make plans by stochastic or un-deterministic strategies. In this type of problems, the important issue is how required information should be collected.
2. The development of a planner with learning mechanisms. Planners reviewed above can not learn solved planning tasks, and can not exploit obtained knowledge to a new planning task. In this type of problems, the interesting issue is effective memorization and searching and filtering techniques.
3. A planning task in multiagent environments in which multiple agents exist in the planning domain. These agents interact each other. Thus, cooperation and negotiations and conflicts among agents occur in the problem domain. This problem is researched as a Distributed Problem Solving.

To solve such problems, a development of the new problem solving paradigm is expected.

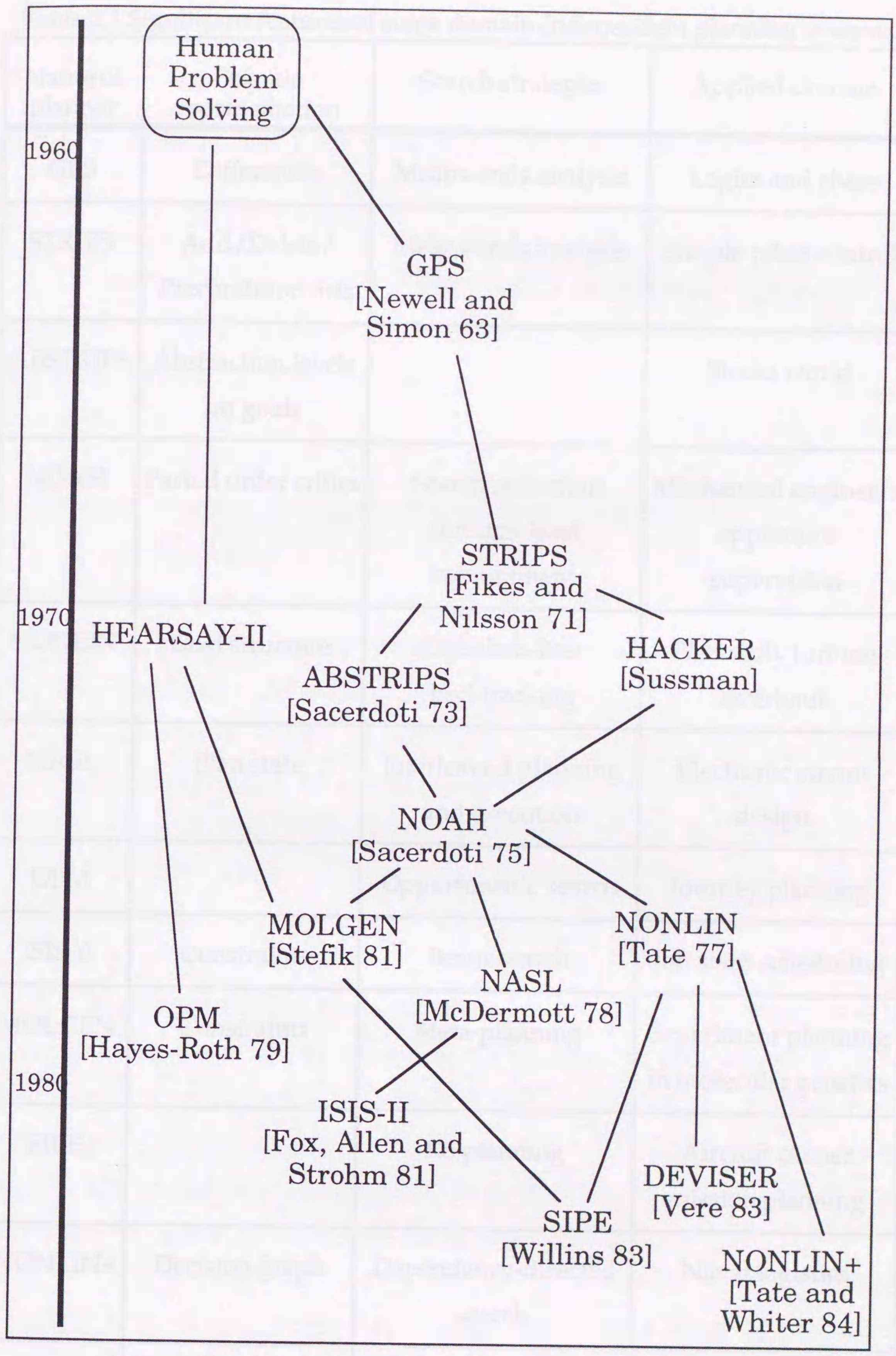


Figure 2.2 A brief chronology of some planning systems.

(from [Tate, Hendller, and Drummond 90])

Table 2.1 Significant features of some domain-independent planning systems.

Name of planner	Domain representation	Search strategies	Applied domain
GPS	Differences	Means-ends analysis	Logics and chess
STRIPS	Add/Delete/ Precondition lists	Means-ends analysis	Simple robot control
ABSTRIPS	Abstraction levels on goals		Blocks world
NOAH	Partial order critics	Search reduction through least commitment	Mechanical engineers apprentice supervision
NONLIN	Goal structure	One-then-Best backtracking	Electricity turbine overhaul
NASL	Plan state	Interleaved planning and execution	Electronic circuit design
OPM		Opportunistic search	Journey planning
ISIS-II	Constraints	Beam search	Job shop scheduling
MOLGEN	Constraints	Meta-planning	Experiment planning in molecular genetics
SIPE		Re-planning	Aircraft carrier mission planning
NONLIN+	Decision graph	Dependency-directed search	Naval logistics
DEVISER	Time windows and events		Voyager spacecraft mission sequencing

2.4 General Formalization of GAPS

2.4.1 Definitions of the Problem Space and the Problem Solver

As I mentioned before, I mainly focus on synthesizing type problems, especially, planning problems as the subjective problem class in this thesis. Moreover, I attempt to solve the following un-solved problems in the planning field by GAPS.

1. A problem in which a planner knows only partial information of environment.
2. The development of a planner with learning mechanisms.
3. A planning task in multiagent environments in which multiple agents exist in the planning domain.

So in section 2.4, I formalize and define the problem solvers such as conventional problem solvers , adaptive problem solvers , and genetics-based adaptive problem solvers. Let us start with the definition of conventional problem solvers.

In conventional problem solvers, such as GPS, the problem solving takes place by search in a problem space. Therefore, the problem solver must generate the complete problem space by recognizing the external environments including a problem statement that is given to the problem solving agent (figure 2.3). A problem space made from given well-defined problem is illustrated as Figure 2.4. The problem space P is defined as;

$$P =< X, Y, X_I, Y_G, F, O > \qquad (2\text{-}4\text{-}1)$$

where X is a set of problem states, and Y namely is a set of properties corresponding to each problem state, and X_I is a set of initial states, and Y_G is a set of goal properties, and F is a surjective mapping function by which set X is mapped onto set Y, and O is a set of operations by which a problem state $x \in X$

is transformed to a new problem state $x' \in X$. Each symbol of P is satisfied following expressions.

$$X_I \subset X \tag{2-4-2}$$

$$Y_G \subset Y \tag{2-4-3}$$

$$F : X \rightarrow Y \tag{2-4-4}$$

$$O : X \rightarrow X \tag{2-4-5}$$

$$\text{for } \forall y \in Y, \ |F^{-1}(y)| \geq 1 \tag{2-4-6}$$

Formula (2-4-6) means that F is a surjection.

Based on the above definitions of the problem space P, I assume the problems solving of search problems and planning problems, as follows.

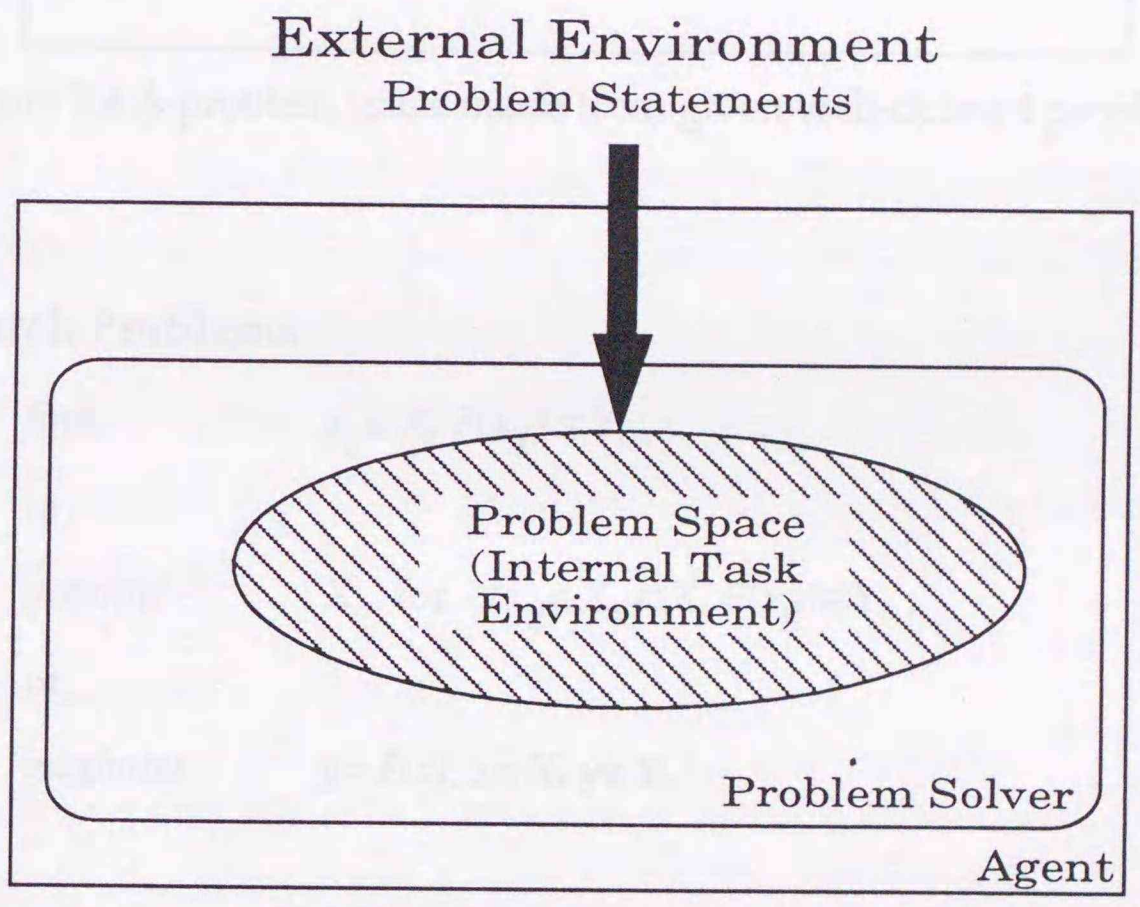


Figure 2.3 The generated complete problem space by recognizing the external environments including a problem statement in conventional problem solvers embedded in problem solving agents.

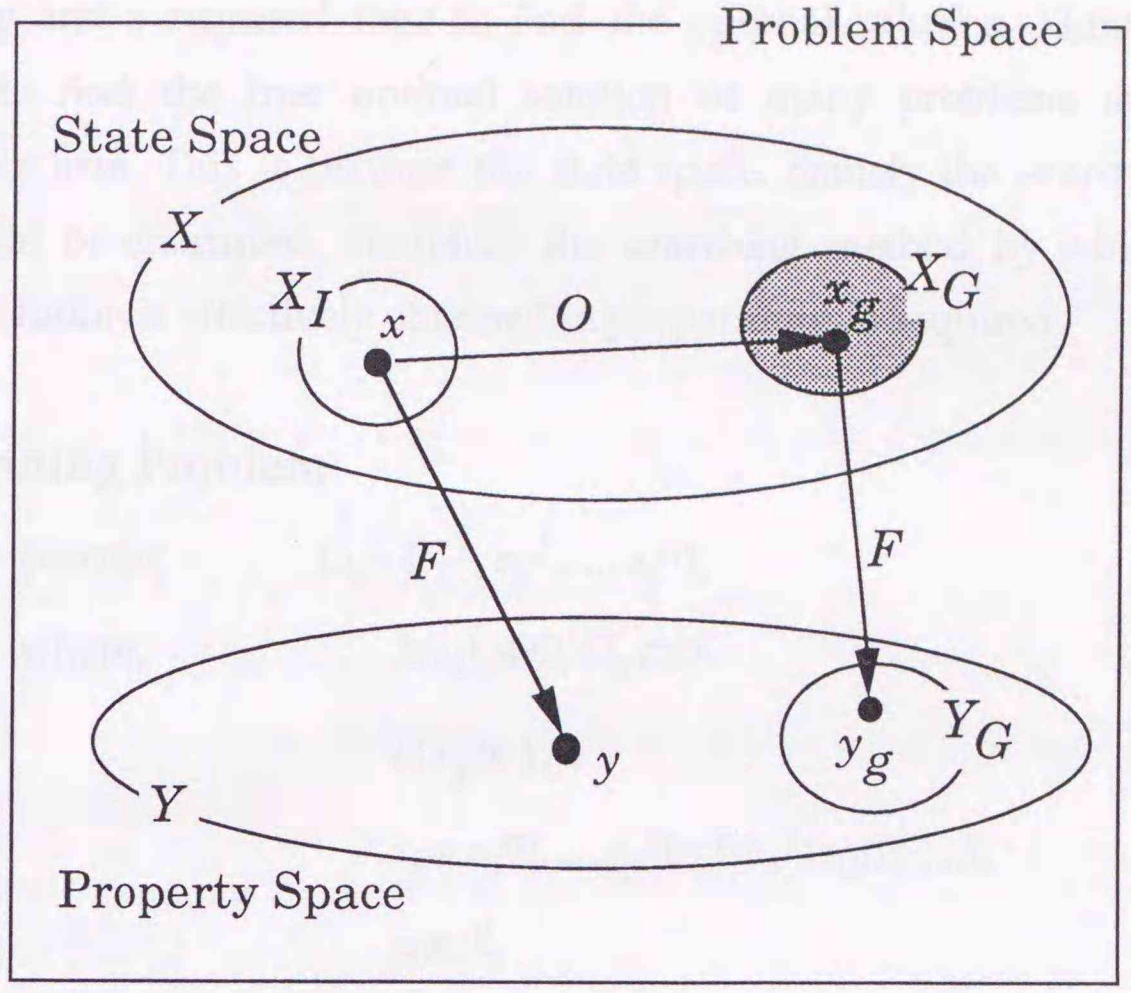


Figure 2.4 A problem space made from given well-defined problem.

The Search Problems

$$\textit{find} \quad x_g \in X,\ F(x_g) \in Y_G, \tag{2-4-7}$$

or,

$$\textit{generate} \quad X_G,\ \text{for}\ \ \forall x_g \in X_G \subset X,\ F(x_g) \in Y_G, \tag{2-4-8}$$

or,

$$\textit{maximize} \quad y = F(x),\ x \in X,\ y \in Y, \tag{2-4-9}$$

The expression (2-4-7) represents a general search problem in which it is necessary to find at least one solution x_g that has specified property $F(x_g) \in Y_G$. Y_G is a given set of objective properties. The other hand, a problem represented by the expression (2-4-8) requires to generate a set of all feasible solutions. Every solution in the set X_G must satisfy the given objective properties.

The expression (2-4-9), in general, is called an optimization problem. This kind of problems are treated in the Operations Research area. The mainly

focused research issues in such optimization problems are an examination of optimality and a required time to find the optimal solution. However, it is difficult to find the true optimal solution of many problems in artificial intelligence area. This is because the state space, namely the search space, is multimodal or enormous. Therefore the searching method by which a near optimal solution is effectively obtained in proper time, is required.

The Planning Problem

$$\textit{generate} \qquad O_S = (o_S{}^1, o_S{}^2, \ldots, o_S{}^n), \tag{2-4-10}$$

$$\text{where,} \qquad \forall o_S{}^i, \in O, O_S \subset O, \tag{2-4-11}$$

$$F(x_g) \in Y_G, \tag{2-4-12}$$

$$x_g = o_S{}^n(\ldots\ldots o_S{}^3(o_S{}^2(o_S{}^1(x_0)))\ldots\ldots), \tag{2-4-13}$$

$$x_0 \in X_I \tag{2-4-14}$$

This problem is called a simple planning problem. The object of this problem is to generate an operation path that can transform the problem state from a given initial state $x_0 \in X_I$ to a goal state x_g which satisfies given goal properties Y_G. Moreover, when following statement is added to the above problem, that means the optimization of plans. This type of problems is called combinatorial optimization problems.

$$\textit{minimize} \qquad \textit{cost} = \sum_{i=1}^{n} C(o_S{}^i) \tag{2-4-15}$$

where C is a cost function which calculates a cost of state transformation from x_i to x_{i+1} by an operator $o_S{}^i$.

2.4.2 Definitions of Adaptive Problem Solvers

As I already described, complete description of problems is necessary in traditional problem solving systems. However most of practical problems are ill-defined problems. Even the well-structured problems such as blocks world problems, when complex tasks are given, the complete problem space cannot be described in a proper memory area. In general, traditional problem solvers cannot solve the following type of problems.

1. A problem in which a problem solver knows or memorizes only partial information of task environment.
2. A problem that should be effectively solved by using learned knowledge of past solved problems.
3. A problem in which the task environment is dynamically changed.
4. A problem in multiagent environments in which multiple agents exist in the problem domain.

Many practical engineering problems are included in the class of above problems. Therefore, to solve the practical engineering problems, I define the Adaptive Problem Solvers (APS) that solve a problem adaptively.

Here I consider characteristics of the problem should be solved adaptively. Using the problem space P defined by the expression (2-4-1), following cases are considered.

- The state space X is too huge to completely memorize in a computational work area.
- The function F that calculates properties of problem states, is unknown or dynamically changed, so the property space Y is initially unknown and gradually changed.
- The effective transformation operation set O is unknown, because the problem space is multimodal.

Definition of Adaptive Problem Solvers

The adaptive problem solver *APS* is defined as;

$$APS=< X_{APS}, Y_{APS}, A_{APS}, X_S, A_S, X_I, Y_G, ,MF, SF, LF, EF, DF>, \quad (2\text{-}4\text{-}16)$$

where, X_{APS} is a set of states, Y_{APS} is a set of properties, A_{APS} is a set of outputs corresponding to states, X_S is a selected set of states, A_S is an output set corresponding to X_S, X_I is an initial state set to be examined, Y_G is a goal property set, *MF* is a mapping function, *SF* is a searching function, *LF* is a learning function, *EF* is an encoding function, and *DF* is a decoding function. Figure 2.5 illustrates a schematic view of the adaptive problem solver APS and its problem solving mechanisms.

Adaptive Search

The adaptive search procedure is performed by *APS* as follows.

In *APS* a subset X_S of problem states is examined because X_{APS} is too large. That is,

$$X_S \subseteq X_{APS} \quad (2\text{-}4\text{-}17)$$

Since *APS* don't know any property evaluation function *F* and hence cannot decide the property of certain problem state $x \in X_S$ by itself, a property is acquired from the external task environment by broadcasting a corresponding output $a \in A_S$ to the environment. This mapping procedure from state space to output space, is achieved by a mapping function *MF*. That is represented as,

$$MF : X_{APS} \rightarrow A_{APS}, \quad (2\text{-}4\text{-}18)$$

is equal as,

$$a = MF(x) \quad (2\text{-}4\text{-}19)$$

Generally in search problems, the mapping function *MF* is fixed. Particularly in a simple problem, the most primary function is used. Thus, for $\forall x \in X_{APS}$,

$$a = MF(x) = x \quad (2\text{-}4\text{-}20)$$

A generated output *a* by the function *MF* is decoded by function *DF* for external environments. Thus,

$$DF : A_{APS} \rightarrow A_E, \quad (2\text{-}4\text{-}21)$$

where A_E is a set of external actions. Then, the environment returns a property y of the output a. Based on the acquired properties set Y_{APS} including the given goal properties Y_G, a new subset $X_S' \subseteq X_{APS}$ of states to be examined at next step, is constructed by searching function SF. That is expressed as,

$$SF : X_{APS} \times X_S \times Y_{APS} \rightarrow X_S \tag{2-4-22}$$

This function SF is controlled by the adaptive searcher AS embedded in APS.

The APS repeatedly executes these adaptive search procedures, and so an acquired property y is gradually approaching to the goal properties Y_G. Therefore the executive cycle of APS is terminated when the following condition is satisfied.

$$y \in Y_G, \tag{2-4-23}$$

The search performance of the APS is mainly affected by the SF. So, it is necessary to construct or select the good adaptive searcher that controls adaptive search strategies.

Reactive Planning

I then change the view point to planning problems. To solve difficult situated problems that I described above, the reactive planning is often said to be a useful approach. These difficult situations mean that, for example, a problem domain is uncertain or non-stationary. In particular, considering robotics application, many difficult issues arise. In any given task, we can usually distinguish between information that we can easily hardwire into the robot from that which would involve a lot of human effort. Sometimes the information necessary to program the robot is simply not readily available. For example, we might want to have a robot explore an unknown terrain. Clearly, in such situations it is imperative that the robot be able to learn a map of the environment by exploring it. Moreover, the real world is a dynamically changing place. Objects move around from place to place, or appear and disappear. Even if we had a complete model of the environment to begin with, this knowledge could quickly become obsolete in a very dynamic environment. There are also slower changes, such as in the calibration of the robot's own

sensors and effectors. Thus, it would be beneficial if a robot could constantly update its knowledge of both its internal and external environments. In such a difficult situation, the planner must have following features [Connell and Mahadevan 93].

- Non-deterministic selection of actions: Since the planner has an incomplete model of its environment, actions will not always have similar effects.
- Reactivity: *A* robot must respond to unforeseen circumstances in real time. In terms of learning, any reasonable algorithm must be tractable in that every step of the algorithm must terminate quickly.
- Incrementality: *A* robot has to collect the experience from which it is to learn the task. The need for efficient exploration dictates that any reasonable learning algorithm must be incremental. Such algorithms should allow the robot to become better at deciding which part of the environment it needs to explore next.

Conventional planning models reviewed in section 2.3, is usually called Sense-Model-Plan-Action model (SMPA model) [Brooks 91]. The planning cycle of SMPA model is illustrated in figure 2.6. It is pointed out that the SMPA model cannot have above required features. On the other hand, the reactive planning method is one of methods that can deal with such the difficult situations [Agre and Chapman 90] [Connel and Mahadevan 93] [Kaelbling and Rosenschein 90] [Maes 90] [Naruse 94]. A reactive planning system flexibly decides an action called behavior based on a state of the world model. On the basis of the having knowledge at a certain time, namely reactive rules, a seemingly appropriate action is selected. Therefore a selected action is not always absolutely correct. Although it can not generate a sequence of behaviors, which leads to the final state, such a sequence is accomplished by deciding the behaviors repeatedly. Figure 2.7 illustrates a process of the reactive planning.

The concept of the reactive planning is realized in the *APS*. A reactive planning procedure is executed as follows.

1. *Encoding process of problem states*: The external problem space X_E is encoded into a set of internal states X_{APS} by the encoding function *EF*, that is represented as,

$$EF : X_E \rightarrow X_{APS}, \tag{2-4-24}$$

is equal to,

$$EF(x_E) = x, \tag{2-4-25}$$

where, $x_E \in X_E$, $x \in X_{APS}$

2. *The mapping process*: The encoded problem state x is mapped over the output space A_{APS} (may be called the action space) by *MF*, thus,

$$MF : X_{APS} \rightarrow A_{APS} . \tag{2-4-26}$$

In reactive planning problems however, *MF* is not fixed. Because the *APS* has to select an action non-deterministically. That is represented as,

$$MF^t (x^t) = a^t \in A_{APS} , \tag{2-4-27}$$

where, MF^t is a mapping function at time t, x^t is an encoded state at time t, a^t is a selected action for x^t at time t, and a^t is an element of A_{APS}. This mapping function *MF* corresponds to the reactive rule, therefore plays main role in reactive planning systems.

3. *The decoding process of an action*: The selected action a^t is decoded by the decoding function *DF*. That is,

$$DF : A_{APS} \rightarrow A_E, \tag{2-4-28}$$

and is equal to,

$$DF(a^t) = a_E{}^t \in A_E, \tag{2-4-29}$$

4. *The learning process*: According to the acquired property y^t for an external action $a_E{}^t$, the learner adjusts the mapping function MF^t. This process is also represented as following expressions.

$$LF : X_{APS} \times A_{APS} \times Y_{APS} \times MF \rightarrow MF, \tag{2-4-30}$$

is function,

$$LF(x^t, a^t, y^t, MF^t) = MF^{t+1}, \tag{2-4-31}$$

By this learning mechanism, *MF* is modified so that an acquired property y^t is approaching to the goal properties Y_G. This type of learning scheme is generally realized by the reinforcement learning scheme.

The reactive planning performance of the *APS* is mainly affected by the *MF*. However, since the *APS* doesn't have enough information of the external task environment, the appropriate mapping function cannot be initially defined. So, it is necessary to construct or select the reliable adaptive learner that adjusts the mapping function.

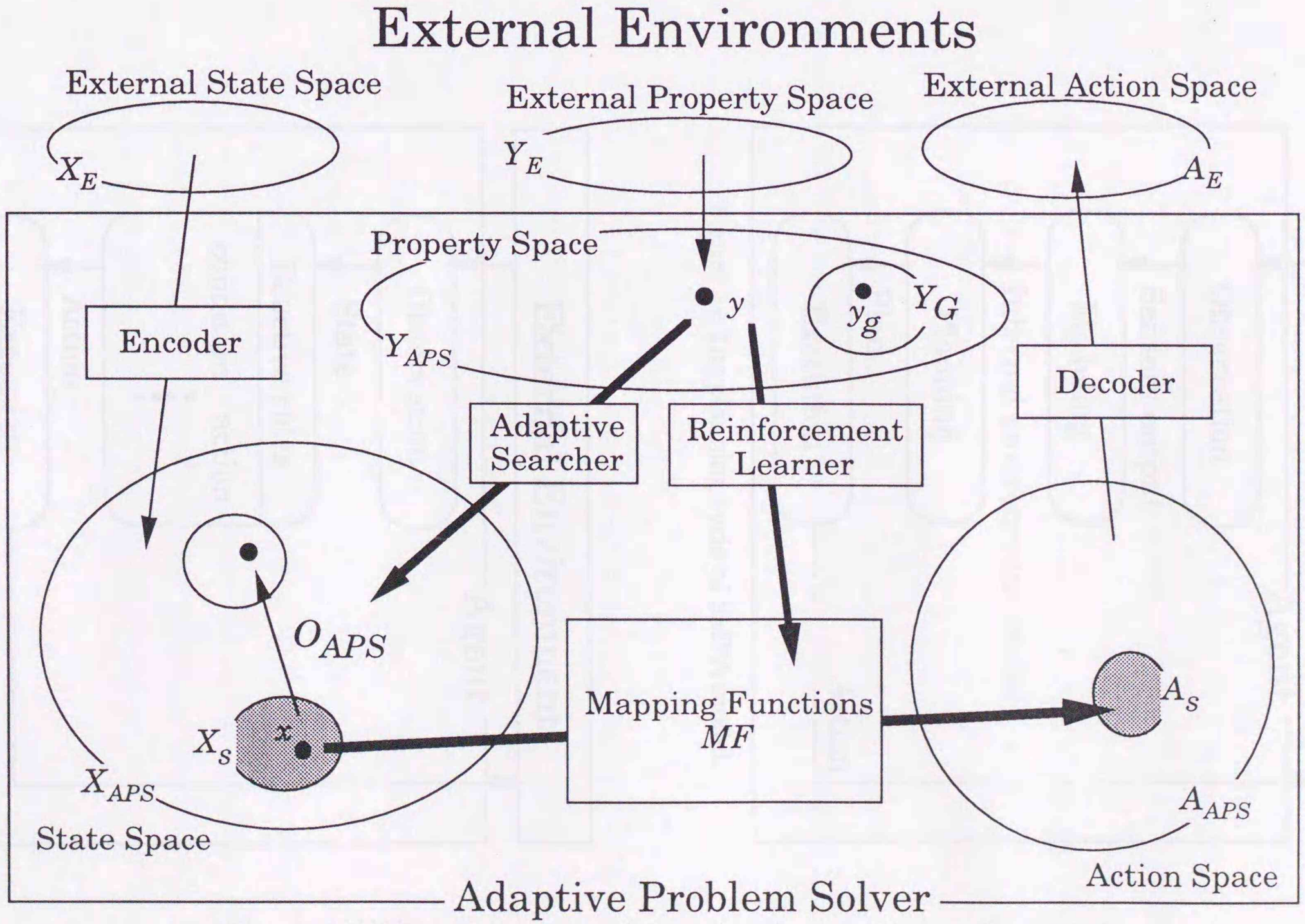


Figure 2.5 Schematic view of the adaptive problem solver *APS* and its problem solving mechanisms.

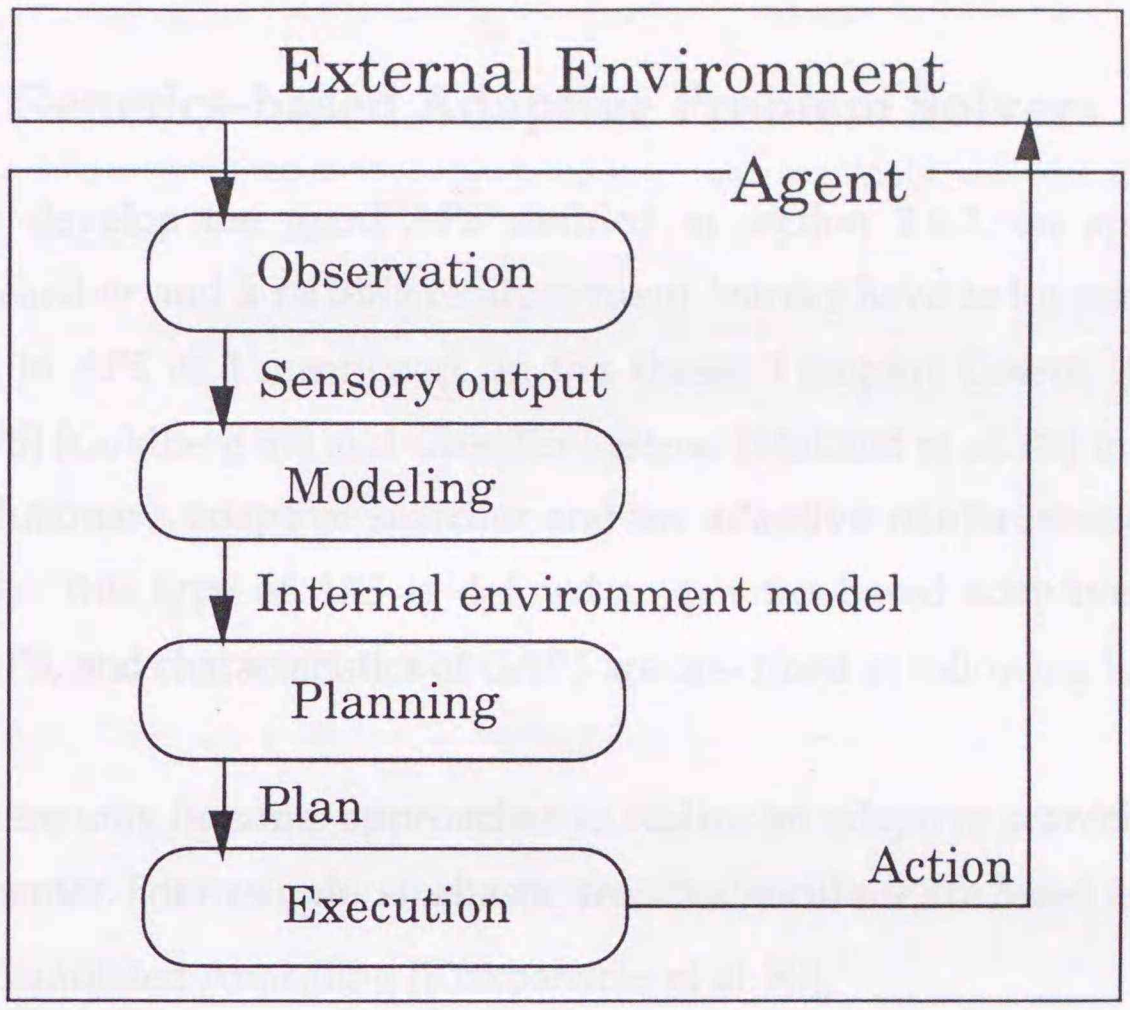


Figure 2.6 The planning cycle of SMPA model.

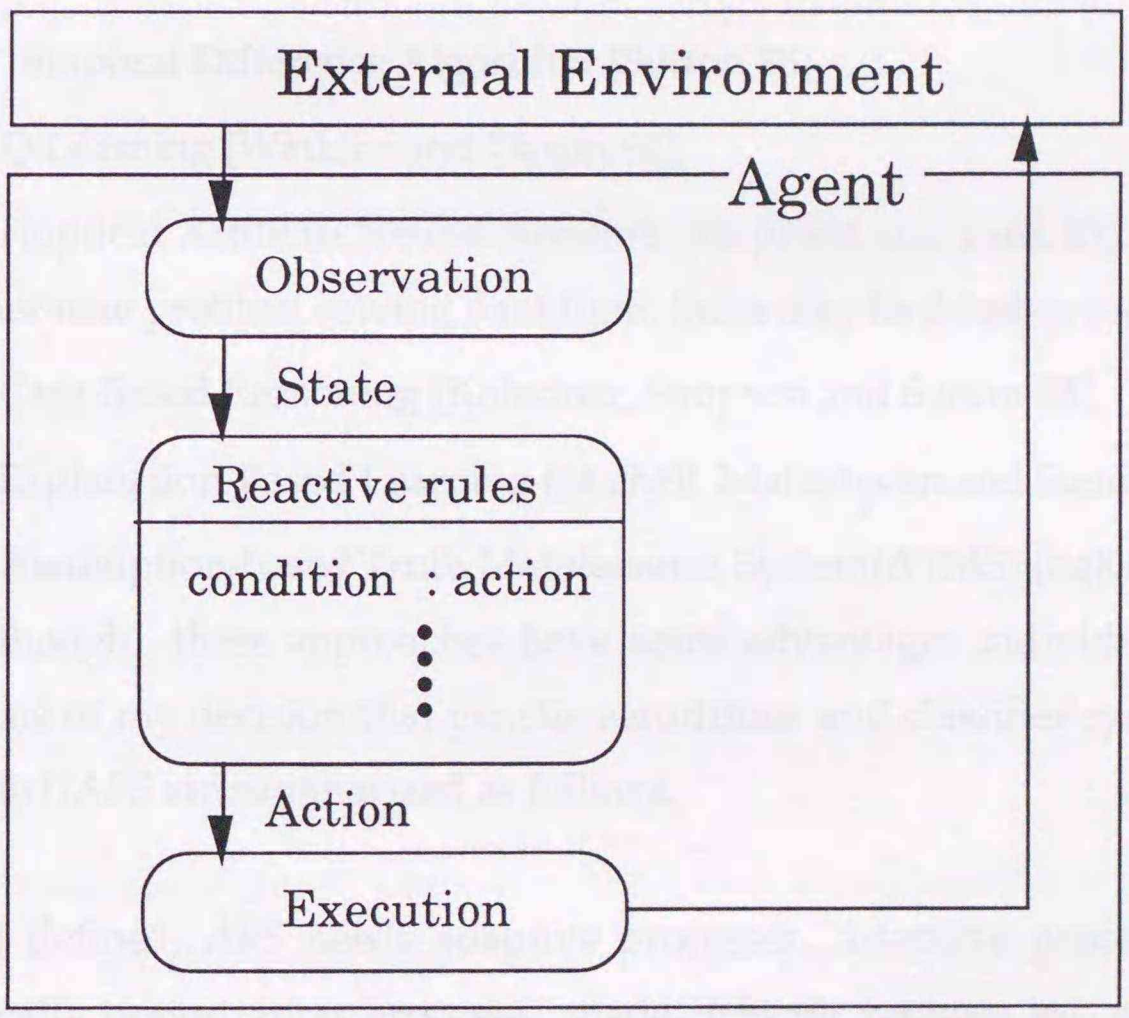


Figure 2.7 A process of the reactive planning

2.4.3 Genetics-based Adaptive Problem Solvers

To develop the good *APS* defined in section 2.4.2, an appropriate adaptive searcher and a reliable reinforcement learner have to be selected and embedded in *APS* as I mentioned. In this thesis, I employ *Genetic Algorithms* [Holland 75] [Goldberg 89] and *Classifier Systems* [Holland et al. 86] in the *APS*, as an evolutionary adaptive searcher and an adaptive reinforcement learner respectively. This type of *APS* is defined as genetics-based adaptive problem solvers *GAPS*, and characteristics of *GAPS* are described as following lines.

There may be some approaches to realize an adaptive searcher and an adaptive learner. For example, stochastic search algorithms are listed as,

- Simulated Annealing [Kirkpatricks et al. 83],
- TABU Search [Kido 93].

Also, as reinforcement learning scheme, following algorithms are listed.

- Stochastic Learning Automaton [Narendra and Thathachar 91],
- Temporal Difference Algorithm [Sutton 88],
- Q-Learning [Watkins and Dayan 92],
- Hopfield Artificial Neural Network [Hopfield and Tank 85].

And as other new problem solving paradigm, there may be listed as follows.

- Case-Based Reasoning [Kolodner, Simpson and Sycara 85],
- Explanation-Based Learning [Michell, Mahadevan and Steinberg 85],
- Assumption-based Truth Maintenance System(ATMS) [deKleer 86].

Although these approaches have some advantages individually, the main reasons of my decision that genetic algorithms and classifier systems are employed in *GAPS* are summarized as follows.

1. As I defined, *APS* needs adaptive processes. Adaptive processes. are basically optimization processes made difficult because the structures being modified are complex and their performance is uncertain. Defined more generally, adaptation designates any process whereby a structure is

progressively modified to give better performance in its particular environment. The other hand, in biology, the adaptation is the basic process underlying evolution, acting at all levels of organization from enzymes to ecosystems. Genetic algorithms are adaptive search algorithms based on the mechanics of natural evolution, natural selection and natural genetics. Therefore I choose to employ genetic algorithms.

2. A classifier system is a genetics-based machine learning system, and is suitable for complex planning tasks such as reactive planning problems. The term "genetics-based" means that a structure of reactive rules in general classifier systems is represented by very simple symbols. Thus the reactive rule space to be searched and modified, is the binary alphabetic space that is the same as the search space used by genetic algorithms. Combining genetic algorithms as an adaptive searcher and classifier systems as an adaptive learner makes the appropriate association in GAPS, because one search space is commonly exploited and explored by the searcher and the learner.

3. The credit assignment system in classifier systems plays role of effective reinforcement learning systems. Therefore, by collecting the learning experience, GAPS incrementally becomes better at deciding which part of the environment it needs to explore next.

4. If we want to use more complex structures in the search space——for example, decision trees for induction, formal logical expressions, such as traditional AI problems, production rules by natural languages, and so on——, then the genetic programming technique [Koza 92] can be applied by representing structures undergoing adaptation in LISP S-expressions. The genetic programming paradigm is an evolutionary synthesis algorithm of computer programs based on the concepts of original genetic algorithms. In particular, the structures are general, hierarchical computer programs of dynamically varying size and shape. In this thesis, however, the extension of search space structures by the

genetic programming is never focused on.

In GAPS , the defined symbols in *APS* are corresponding to structures of genetic algorithms and classifier systems as follows.

The Search Problems by the Adaptive Searcher Based on Genetic Algorithms

x : I_i, a binary string that has l length, thus,

$$I_i \in \{0, 1\}^l \tag{2-4-32}$$

X_S : I^{λ}, the set of binary strings being tested, is namely a population. λ is a population size.

$$I^{\lambda} = I_1 \times I_2 \times \ldots\ldots\ldots \times I_{\lambda} = \prod_{i=1}^{\lambda} I_i \tag{2-4-33}$$

X_{APS} : $\boldsymbol{I}$, the set of all feasible space of l-bit binary strings.

$$\boldsymbol{I} = \{0, 1\}^l \tag{2-4-34}$$

X_I : $I^{\lambda}_{(0)}$, initial set of strings.

Y_{APS} : **Reals**, fitness values.

SF : Ω, a set of adaptive plans, that is called genetic operators, e.g., selection, crossover and mutation.

DF : the decoding function by which a binary string is decoded to an output for the external environment.

The Reactive Planning Problems by the Reinforcement Learner Based on Classifier Systems

x : *EM*, an environmental message that then matched with conditional parts of reactive rules that is called classifiers. *EM* is represented as,

$$EM \in \{0, 1\}^{l_{EM}} \tag{2-4-35}$$

X_{APS} : $\{0, 1\}^{l_{EM}}$, the set of all feasible states that are represented as fixed lengthen bit patterns.

a : cf^a_i, a decided action that is also represented as a fixed lengthen bit string, that is called an action part of classifiers, thus,

$$cf^a_i \in \{ 0, 1 \}^{la} \tag{2-4-36}$$

A_{APS} : $\{ 0, 1 \}^{la}$, the set of all feasible actions.

Y_{APS} : **Reals**, reinforcement signals.

MF : CF, the set of classifiers by which an environmental message is mapped into candidate action strings. CF is expressed as,

$$CF = \{cf_i ; i=1,2,..,N\} \tag{2-4-37}$$

Each classifier cf_i maintains a numerical parameter Scf_i called strength that represents past usefulness of cf_i.

LF : the learning function by which classifier rules CF are modified based on reinforcement signals. In classifier systems, learning function LF is realized by credit assignment algorithms such as profit sharing and bucket brigade algorithm, and by new rule discovery algorithms.

EF : the encoding function by which states of external environments are encoded into fixed lengthen binary messages.

DF : the decoding function by which a binary action string is decoded to an actual action for the external environment.

2.5 Genetic Algorithms as Adaptive Evolutionary Computation Scheme

2.5.1 An Overview of Genetic Algorithms

Genetic Algorithms are search algorithms based on natural selection and natural genetics by organized evolution pressure of biological system. In genetic algorithms, an object to be evolved is encoded to a string using alphabets and numbers, and a population is generated as a set of strings. Each string is determined to survive or die depending on its fitness value. The values are calculated based on each search result. Only survived strings are copied to the next generation. This process is called reproduction. To acquire the further adapted string, genetic operations such as the cross-over operation and the mutation operation are applied. A new population is generated as a set of operated strings by these genetic operators. The population is evolved and eliminated repeatedly by each generation (figure 2.8). For solution of search problems, genetic algorithms have been investigated recently and shown to be effective at exploring a large and complex space in an adaptive way, guided by the equivalent biological evolution mechanisms of reproduction, crossover, and mutation. For solution to a search problem, five components are required in genetic algorithms[Adeli and Hung 95]:

1. Representation: This is a way of encoding the structures of the search problem in a string of binary digits (1's and 0's) called a chromosome. If there are *m* structures in a search problem and each structure is encoded as an n-digit binary number, then a chromosome is a string of n x *m* binary digits.
2. Evaluation Function: This function is used to evaluate the given structures and return a value. The value of a chromosome's objective function is a fitness of that chromosome. The fitness is used to determine the probability that this chromosome will be selected as a parent chromosome to generate new chromosomes.

3. Initialization of the Population: A method of initializing the population of chromosomes is needed. In general, the population of chromosomes is initialized in random.
4. A Set of Operators to Perform Evolution between Two Consecutive Chromosome Populations: Genetic algorithms use parent selection techniques that mimic the process of natural selection for selecting chromosomes to create a new generation, where the fittest members reproduce most often. After the parent selection, the process of genetic operation is applied. In general, crossover and mutation are employed as genetic operations. The crossover is applied to recombine two chromosomes and generate two new chromosomes when a random value associated to this pair is greater than a pre-defined crossover rate. The operation of one-point mutation simply alters one bit in the string when a random value, between 0 to 1, associated to that bit is greater than a pre-defined mutation rate.
5. Working Parameters. A set of parameters is pre-defined to guide the genetic algorithm, such as the length of each structure encoded as a binary string, the number of chromosomes to be generated and operated in each generation, the crossover rate, the mutation rate, and the stopping criterion. The crossover and mutation rates are used as thresholds to determine whether the operators have to be applied to a pair of parent chromosomes or not. In general, the value of crossover and mutation rates are assigned as real numbers, between 0 and 1. The stopping criterion is pre-defined as the number of iterations or a tolerance value for the objective function.

The executive cycle of genetic algorithms is shows in figure 2.9.

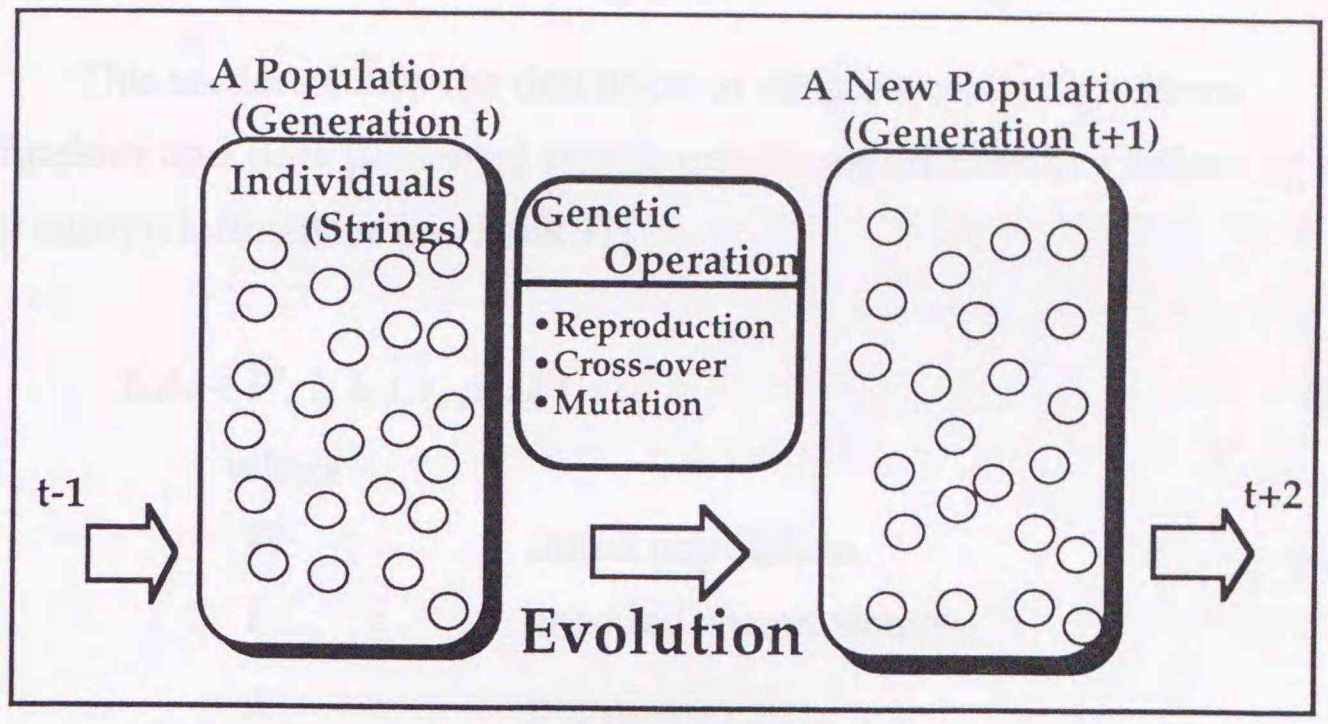


Figure 2.8 Evolving populations by genetic algorithms.

step 1	Encode the structures being adapted and evolved as chromosomes.
step 2	Initialize a population of chromosomes.
step 3	Evaluate each chromosome in the current population.
step 4	Create new chromosomes by mating current chromosomes; apply mutation and recombination as the parent chromosomes mate.
step 5	Delete members of the population to make room for the new chromosomes.
step 6	Evaluate the new chromosomes and insert them into the population.
step 7	If time is up, then stop and return the best chromosome; if not, then go to step 4.

Figure 2.9 The executive cycle of genetic algorithms(from [Davis 91]).

2.5.2 Definitions of Simple Genetic Algorithms

This section shows the definition of simple genetic algorithms. Hoffmeister and Back presented simple genetic algorithms as a following eight-tuple entity[Hoffmeister and Back 91].

$$GA=< P^0, I, \lambda, l, s, \rho, \Omega, f, t >, \tag{2-5-1}$$

where

P^0	:	initial population,
I	:	encoded chromosomes,
λ	:	population size,
l	:	length of chromosome,
s	:	selection operator,
ρ	:	operator determination,
Ω	:	operator set,
f	:	fitness function,
t	:	termination criterion.

These entities are formalized as following expressions.

$$P^0 = (a_1^0, \ldots, a_\lambda^0) = \left(\begin{bmatrix} a_{1,1} \\ \cdot \\ \cdot \\ \cdot \\ a_{1,l} \end{bmatrix} \cdots \begin{bmatrix} a_{\lambda,1} \\ \cdot \\ \cdot \\ \cdot \\ a_{\lambda,l} \end{bmatrix} \right) \in I^\lambda, \tag{2-5-2}$$

$$I = \{0, 1\}^l, \tag{2-5-3}$$

$$\lambda \in N, \tag{2-5-4}$$

$$l \in N, \tag{2-5-5}$$

$$s : I^\lambda \rightarrow I^\lambda, \tag{2-5-6}$$

$$\rho : I \rightarrow \Omega, \tag{2-5-7}$$

$$\Omega \subseteq \{\omega : I \times I^\lambda \rightarrow I\}, \tag{2-5-8}$$

$$f: I \rightarrow \textbf{Reals}, \tag{2-5-9}$$

$$t: I^{\lambda} \rightarrow \{0, 1\} \tag{2-5-10}$$

As general genetic operator of Ω, crossover and mutation are also formalized as follows.

c : crossover operator,

m : mutation operator,

$$c: I^2 \rightarrow I^2, \tag{2-5-11}$$

$$m: I \rightarrow I, \tag{2-5-12}$$

There are λ chromosomes in each population. The initial population of chromosomes P^0, is generated randomly. The entity a_k^t denotes the k-th chromosome in the t-th generation of population, P^t. A chromosome I, is encoded as a string of binary digits. The variable N is a set of integer. The evolution process of genetic algorithm is continued (t=0) until one of the termination criteria is met (t=1).

The selection operator s, produces an intermediate population $P^{t'}$ from the population P^t in the t-th generation by generating copies of elements from P^t.

$$s(P^t) = P^{t'} = (a_1^{t'}, \cdots, a_\lambda^{t'}) \tag{2-5-13}$$

In $P^{t'}$, any $a_i^{t'} = a_j^t$ is selected according to the following selection probability function p_S.

$$p_S: I \rightarrow [0, 1], \tag{2-5-14}$$

$$p_S(a_j^t) = \frac{f(a_j^t)}{\sum_{k=1}^{\lambda} f(a_k^t)} \tag{2-5-15}$$

Genetic operators are applied to the population P^t after the intermediate population $P^{t'}$ is produced. For each a_i^t, the applied operator is determined by

ρ, that is,

$$\rho(a_i^t) = \omega_i^t, \tag{2-5-16}$$

Two-Point Crossover Operator

As the one of $\omega_i^t \in \Omega$, the two-point crossover operator c_{tp} is defined as follows. For any pair of selected chromosomes in a population P^t, an associated real value r=[0, 1], is generated randomly. If r is greater than the predefined crossover threshold r_c, the crossover operator is applied to this pair of chromosomes. This pair of chromosomes is assumed as $\{a_i^t,\ a_j^t\}$. The two-point crossover c_{tp} produces a new pair of chromosomes $\{a_i^{t'},\ a_j^{t'}\}$ that is added to the intermediate population $P^{t'}$. The operation of c_{tp} is represented as:

$$\begin{Bmatrix} a_i^{t'} \\ a_j^{t'} \end{Bmatrix} = C_{tp}\left(\begin{Bmatrix} a_i^t \\ a_j^t \end{Bmatrix}\right)$$

$$= \begin{Bmatrix} \left[a_{j,1},\cdots,a_{j,\rho 1},a_{i,(\rho 1+1)},\cdots,a_{i,\rho 2},a_{j,(\rho 2+1)},\cdots,a_{j,l}\right]^T \\ \left[a_{i,1},\cdots,a_{i,\rho 1},a_{j,(\rho 1+1)},\cdots,a_{j,\rho 2},a_{i,(\rho 2+1)},\cdots,a_{i,l}\right]^T \end{Bmatrix} \tag{2-5-17}$$

where $1 < \rho 1 < \rho 2 < l$. In this crossover strategy, two positions in a pair of chromosomes are selected. The pair of chromosomes is divided into three sub-chromosomes by these two points and crossed over to each other by swapping the first and third sub-chromosomes.

Mutation Operator

As another general operator, the mutation operator is defined as follows. For any chromosome in a population P^t, an associated real value r=[0, 1], is generated randomly. If r is less than the predefined mutation threshold r_m, the mutation operator m is applied to this chromosome. The mutation operator simply alters one bit from 0 to 1 , or 1 to 0 in a chromosome. For randomly selected chromosomes a_i^t from P^t, the operator of mutation m, produces a new

chromosome $a_i^{t'}$, that is added to the intermediate population $P^{t'}$. The mutation operator m is represented as:

$$m(a_i^t)=a_i^{t'}=\left[a_{i,1}^{'},\cdots,a_{i,k}^{'},\cdots,a_{i,l}^{'}\right] \tag{2-5-18}$$

$$a_{i,k}^{'}=\begin{cases} a_{i,k} & for\ k\in\{1,2,\ldots,\rho_m-1,\rho_m+1,\ldots,l\} \\ \overline{a_{i,k}} & for\ k=\rho_m \end{cases} \tag{2-5-19}$$

where $1<\rho_m<l$.

After these operations, all members of the intermediate population $P^{t'}$ are copied into a population of next generation P^{t+1}.

2.6 Classifier Systems as Adaptive Reinforcement Learning Scheme

2.6.1 An Overview of Classifier Systems

The classifier system[Holland et al. 86] is a machine learning system that has the strong learning performances and very simple mechanisms. The classifier system also has many similarities to the production rule based expert systems. In the classifier systems and the production rule-based systems, a strategy is identically performed by the activated rules that satisfy the environmental conditions. But the rule syntax of the classifier systems is very different from that of the production rule-based systems. Thus, many production systems permit to involve complex grammatical constructions for the condition and action parts of a rule. Therefore, there are many difficulties of employing production systems in the learning situations. The other hand, classifier systems depart from these difficulties by restricting a rule to fixed length representation using alphabets or numbers. By this restriction, string rules can be operated easily. In this classifier system, an appropriate strategy for an arbitrary environment is represented as a syntactically simple string rule that is called classifier, and all the strategies of the system are controlled by a set of the finite number of classifiers. Therefore, in the machine learning with classifier system, a set of the classifiers evolves for the given problem from an initial state to an adapted state by autonomous evolutional learning mechanisms over learning generations. This learning mechanism is realized by the simple credit distribution algorithm for all classifiers and the new rule discovery algorithms.

In general, a framework of original classifier system consists of several components that are schematically shown by figure 2.10. The details of these components are defined in the following sections.

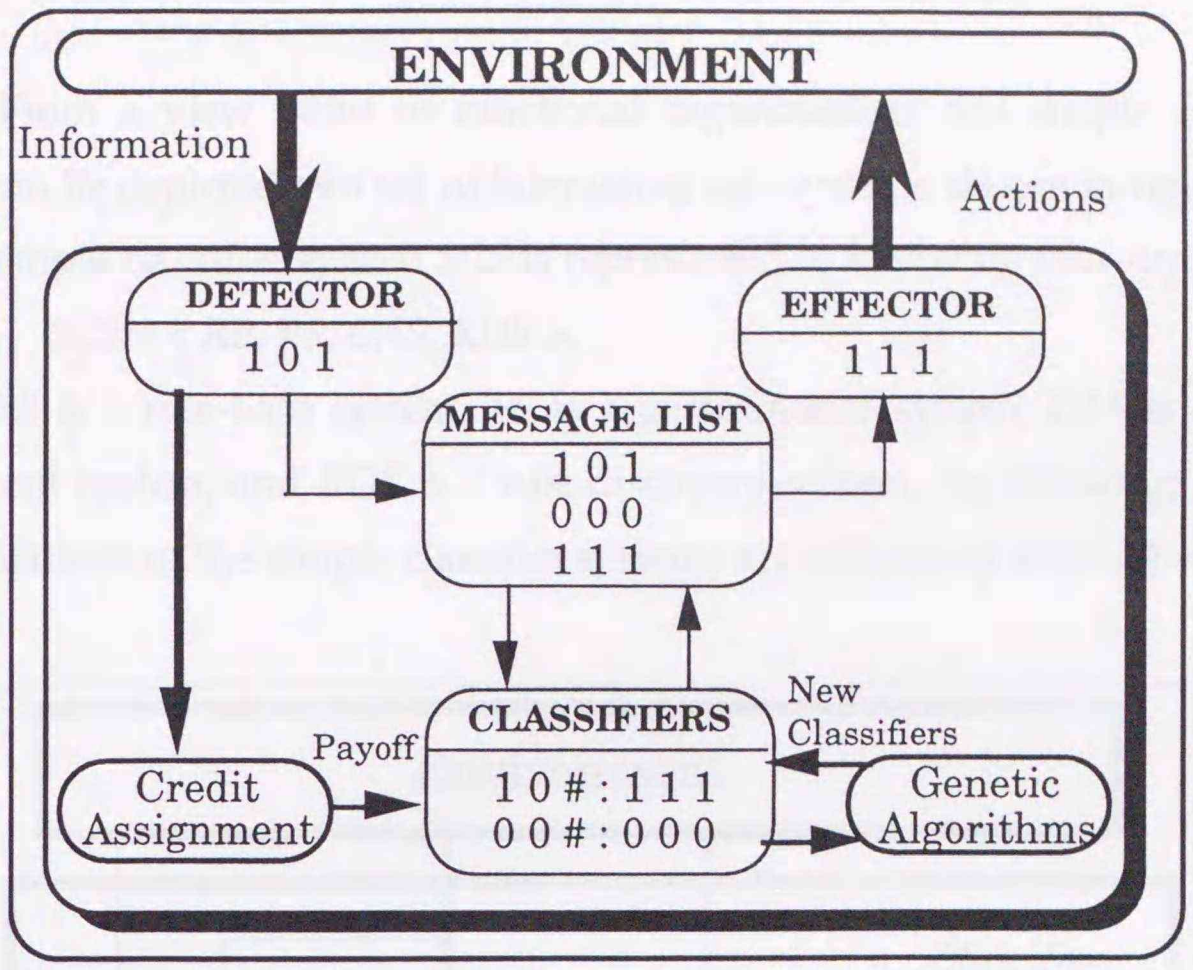


Figure 2.10 A schematic illustration of the original classifier system.

2.6.2 Formalization of Simple Classifier Systems

Though a number of researchers were inspired by Holland's framework to investigate classifier systems [Wilson and Goldberg 89] and it had some influence on the related field of reinforcement learning, efforts to realize the framework's potential have met with mixed success, primarily due to difficulty understanding the many interactions of the classifier system mechanisms that Holland outlined. In [Wilson 94], Wilson pointed that the most successful studies tended, in fact, to simplify and reduce the canonical framework, permitting better understanding of the mechanisms that remained.

In this section, I define the architecture of simplified classifier systems, named simple classifier systems. A simple classifier system has no internal message list. That is unlike to Holland's original classifier system concepts[Holland et al. 86]. Considering the implementation to reactive planning on GAPS, it is not necessary to have the internal message list. Note that the problem solving system in which a simple classifier system is implemented, is called an *Agent*.

From a view point of functional organizations, the simple classifier system can be depicted as a set of interacting sub-systems shown in figure 2.11. So, the simple classifier system *SCS* is represented as following four-tuple:

$$SCS = < RB, PS, CAS, RDS >, \tag{2-6-1}$$

where, *RB* is a rule-base system, *PS* is a performance system, *CAS* is a credit assignment system, and *RDS* is a rule discovery system. In followings, formal representations of the simple classifier systems are defined by each sub-system.

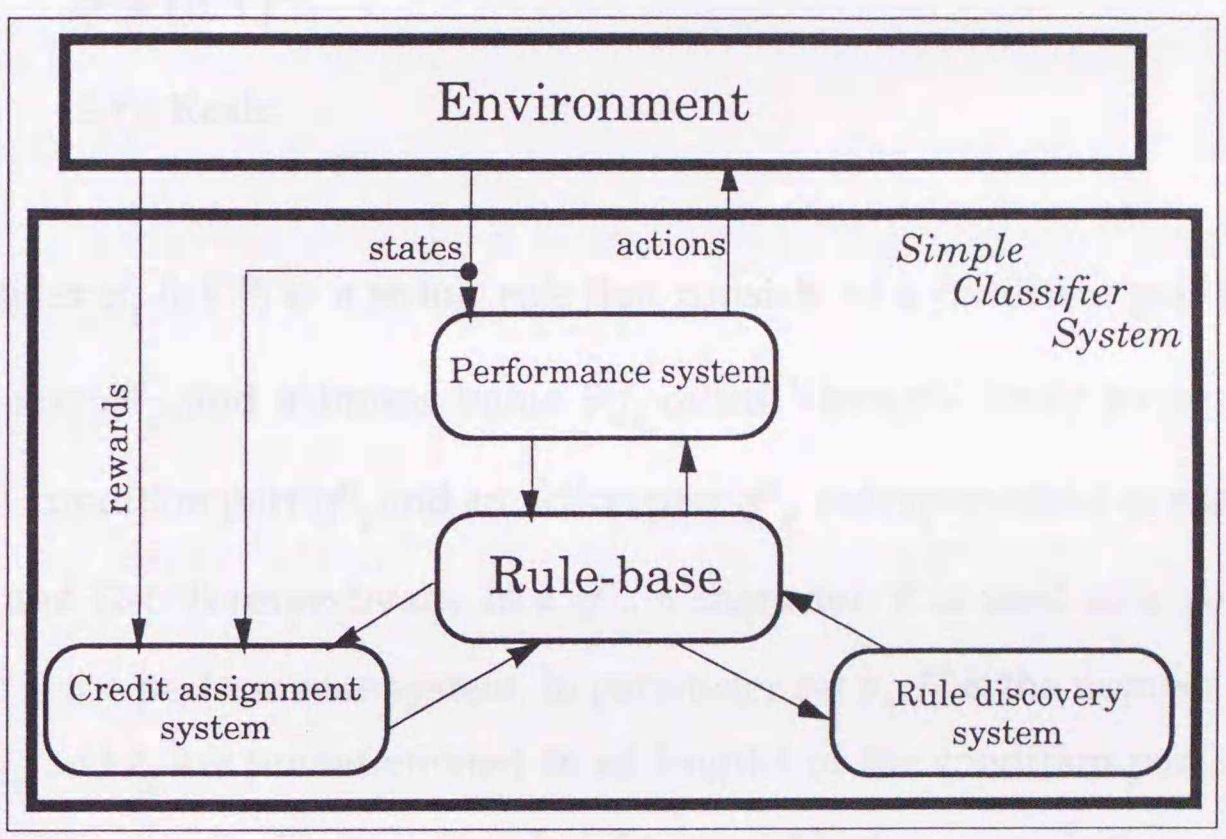


Figure 2.11 Functional organization of a simple classifier system.

RB: The Rule-base System

In the rule-base system, a set of classifiers exists. The classifiers play as reactive rules in a reactive planning process. The rule-base system *RB* is defined as following four-tuple,

$$RB = < CF, I, RGF, p_R >, \tag{2-6-2}$$

where, *CF* is a set of classifiers, *I* is a set of characters used in a classifier, *RGF* is a rule generating function that generates an initial *CF* , p_R is a set of internal parameters about classifiers. The initial set of classifiers $CF_{(0)}$ is generated by *RGF* as follows.

$$RGF : I \times p_R \rightarrow CF \tag{2-6-3}$$

where,

$$I = \{ 0, 1, \# \}, \tag{2-6-4}$$

$$p_R = \{ N, l_C, l_a, P_\#, S_0 \}, \tag{2-6-5}$$

$$CF = \{ cf_i ; i = 1,2,...,N \} \tag{2-6-6}$$

$$cf_i = \{ cf^c{}_i , cf^a{}_i , Scf_i \}, \tag{2-6-7}$$

$$cf^c{}_i \in \{ 0, 1, \# \}^{lc}, \tag{2-6-8}$$

$$cf^a{}_i \in \{ 0, 1 \}^{la}, \tag{2-6-9}$$

$$Scf_i \in \textbf{Reals}. \tag{2-6-10}$$

A classifier cf_i ($\in$ CF) is a string rule that consists of a condition part $cf^c{}_i$, an action part $cf^a{}_i$, and a fitness value Scf_i called 'strength' likely to production rules. A condition part $cf^c{}_i$ and an action part $cf^a{}_i$, are represented as expression (2-6-8) and (2-6-9) respectively. In a $cf^c{}_i$, a character '#' is used as a *'don't care'* symbol in the performance system. In parameter set p_R, N is the number of rules in CF, l_c and l_a are predetermined fixed lengths of the condition part and the action part respectively, $P_\#$ is a probability variable that controls the generated number of symbol "#" in a conditional part. The expected number of the symbol "#" in a $cf^c{}_i$, is $l_c * P_\#$. The initial strength values of all classifiers are equivalently settled by the parameter S_0.

PS: The Performance System

In the performance system *PS*, main processes of a stimulus-response cycle are executed. Figure 2.12 illustrates these processes. As shown in figure 2.12, following four processes in the basic executive cycle are performed.

1. Encoding process of environmental information: Through the detector as a perceptive instrument, an environmental information is encoded into internal message that has an appropriate form for processing in the *SCS*.

Let E be a set of external states of environmental information, and EM be a set of feasible internal messages, and EF be an encoding function, then this process is represented as following expression.

$$EF: E \rightarrow EM, \tag{2-6-11}$$

where,

$$EM \in \{0, 1\}^{l_{EM}}. \tag{2-6-12}$$

2. Conditional matching process: The most important property of the classifier system is a Parallelism. That is, many classifiers simultaneously be candidates of active classifier. The conditional matching between an encoded message EM and all conditional parts cf^c_i of all classifiers cf_i ($\in CF$), is achieved to extract a set of matched classifiers $CF_M(\subseteq CF)$. Let MF be a matching function, then this process is expressed as,

$$MF: \mathrm{CF} \times EM \rightarrow CF_M, \tag{2-6-13}$$

3. Rule competition process: To select an appropriate action, rule competition among matched classifiers CF_M is performed, and active set of classifiers $CF_A(\subseteq CF_M)$ is generated. Let RCF be a rule competition function, then this process is described as following expression.

$$RCF: CF_M \rightarrow CF_A, \tag{2-6-14}$$

In this competition, the usefulness of candidates is evaluated according to those strength values Scf_i (i = 1,2,...,N). As a result of the competition, a classifier having the higher strength value is selected with high selection probability. In general, this selection process is called a roulette wheel selection procedure. Thus, a selection probability $P_sel(cf_i)$ of a classifier cf_i is calculated by following equation.

$$P_sel(cf_i) = \frac{f(cf_i) * Scf_i}{\sum_{j=1}^{N} \{f(cf_j) * Scf_j\}} \tag{2-6-15}$$

where,

$$f(cf_i)=\begin{cases}1: if\ cf_i \in CF_M \\ 0: if\ cf_i \notin CF_M\end{cases} \quad (2\text{-}6\text{-}16)$$

4. Decoding process of action code: A winner classifier CF_A in the rule competition instructs an action of the agent according to its action part. This determined strategy is performed through the effector where a binary code of an action part is transformed to an executable command formation. Let $A = \{ a_j ; j = 1,2,...,N_A \}$ be a set of executable actual actions of the agent, and DF be a decoding function, then this process is represented as,

$$DF: CF_A \rightarrow A \quad (2\text{-}6\text{-}17)$$

On the basis of above descriptions, the performance system PS is defined as following eight-tuple;

$$PS=(CF, E, EM, EF, MF, RCF, DF, A), \quad (2\text{-}6\text{-}18)$$

where CF involves sub-sets of classifiers, i.e., $CF_A \subseteq CF_M \subseteq CF$, and strength values Scf_i.

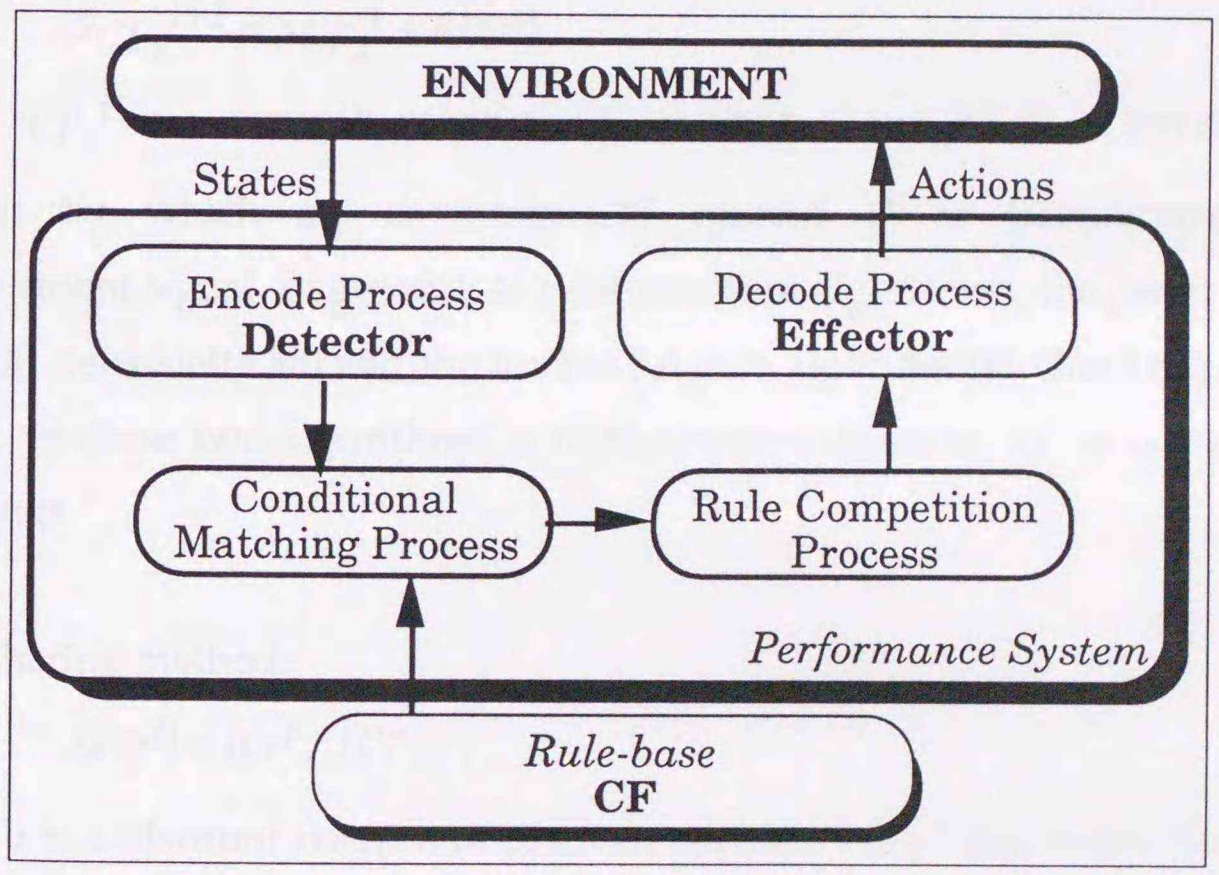


Figure 2.12 A basic sense-act cycle in the *performance system*.

CAS: The Credit Assignment System

Here, the reinforcement learning algorithm by which a set of classifiers are adapted and evolved is presented. The environmental rewards are assigned to the selected classifiers based on the results of the performed actions. By this algorithm, strength values of classifiers are revised, such that the useful classifiers for the given task tend to have higher strength values. Then the adaptive knowledge acquired from many instances.

Let r be an environmental reward value, and RF be a reinforcement function, then this learning procedure is represented as,

$$RF: CF \times r \rightarrow CF \qquad (2\text{-}6\text{-}19)$$

More precisely, this reinforcement procedure is applied every time step. That is, at time t, based on the achieved action a^t instructed by the selected classifier $CF_A{}^t$, the environmental state changes from E^t to E^{t+1}. As the result of changing the state, an environmental reward r^t is given. Then, the strength value of $CF_A{}^t$ revised from $S_{CF_A}{}^t$ to $S_{CF_A}{}^{t+1}$ according to the reward r^t. That is represented by following expression.

$$S_{CF_A}{}^{t+1} = S_{CF_A}{}^t + RF(r^t)\,, \qquad (2\text{-}6\text{-}20)$$

where $S_{CF_A}{}^t$ is a strength value of CF_A at time t, and RF is a transforming function by which an environmental reward r^t is transformed to a reinforcement signal. In general, as reinforcement algorithms, the profit sharing method[Grefenstette 88] and the bucket brigade algorithm[Holland 85] are well known. In these two algorithms, a reinforcement function RF is expressed as followings.

profit sharing method:

$$RF(r^t) = \alpha\, r^t / |CF_A|\,, \qquad (2\text{-}6\text{-}21)$$

where, α is a discount coefficient of given reward, $|CF_A|$ means the number of rules in CF_A. The profit sharing method is a bundle reinforcement scheme by which sequential classifiers that participate in the reward r^t, collectively receive the assigned equivalent reinforcement signal. So, participating sequential

classifiers are memorized in CF_A.

bucket brigade algorithm:

$$RF(r^t) = \alpha\, r^t - \beta\, S_{CF_A}{}^t + \gamma\, S_{CF'_A}{}^{t+1} \quad , \tag{2-6-22}$$

where, α is a discount coefficient of given reward, β and γ are learning rate for strength updates under bucket brigade, and CF'_A is an activated classifier at time *t+1*. Bucket brigade algorithm is a local technique for apportioning the strength. This mechanism is performed by continuous payment of fraction of classifier's strength.

On the basis of above descriptions, the credit assignment system *CAS* is defined as following triple;

$$CAS = < CF, R, RF >, \tag{2-6-23}$$

where R is a set of environmental rewards, thus, $r \in R$.

RDS: Rule Discovery System

Although classifier rules in the rule-base can be learned by the credit assignment system, a rule discovery mechanism is one of the important elements in machine learning system because all of the feasible rules cannot be searched. To search a more better rule, a new rule that is not in a current rule-base , must be generated and added in the rule-base. In the classifier system, since the representation of rules is very simple and fixed-length form, it is easy to produce a new rule by using the genetic algorithms [Holland 75] [Goldberg 89]. This rule discovery procedure is controlled by a parameter set $p_D = \{ \rho, \chi, \mu \}$. These are described as follows.

- ρ: Starting probability of rule discovery procedure by genetic algorithms per time-step of the performance cycle. Therefore, average starting times in N_L performance cycles, is $\rho^* N_L$.
- χ: Probability of crossover per invocation of the genetic algorithms. Thus, average number of new classifiers generated by a crossover operation in

one GA cycle, is χ^*N.

μ: Probability of mutation per invocation of the genetic algorithms. Thus, average number of new classifiers generated by a mutation operation in one GA cycle, is μ^*N.

Let *RDF* be a rule discovery function, the rule discovery system *RDS* is defined as following double;

$$RDS = (CF, RDF, p_D, GA), \tag{2-6-24}$$

where *GA* is simple genetic algorithms defined by expression (2-5-1). In *RDS*, then a set of classifiers *CF* is modified as following expression.

$$RDF: CF \times p_D \times GA \rightarrow CF \tag{2-6-25}$$

2.7 Required Robustness of GAPS in Uncertain and Non-Stationary Problem Domains

A realistic adaptive system has to have the robustness of the underlying adaptive plans or strategies. The robustness means its efficiency over the range of environments that it may encounter. Particularly, the robustness is needed in uncertain or non-stationary problem domains such as subjective problems of this thesis. By the robust adaptive mechanism, the system can effectively and flexibly search and get a new solution for a similar problem domain based on a past adapted internal structure. A similar problem domain means that the environment slightly changed. An adaptive system having robustness should exploit an unchanged or common part of adapted structures for a new problem domain to realize a speedup problem solving.

In general, a macro knowledge called the meta-knowledge of problem domains is usually utilized to realize a speedup problem solving or learning. The explanation-based learning[Michell, Mahadevan and Steinberg 85] technique is well known as a utilization method of the meta-knowledge. However an explanation-based learning system needs the perfect domain knowledge and the appropriate operational criteria to work effectively.

The other hand, since the reinforcement learning scheme implemented in GAPS is a behavior-based learning technique, re-adaptation can be carried out based on given environmental rewards form the changed environment. Especially, in classifier systems, multiple reactive rules are evaluated and reinforced for one environmental state in learning processes. Therefore, for one state, the system does not determine one fixed rule, but revises selection probabilities of feasible rules. As the result, re-learning process is flexibly achieved.

The robustness of the adaptive search based on genetic algorithms is explained by the schemata concept and the schema theorem[Holland 75] [Goldberg 89]. The schema is described as follows.

Schemata

Let $\delta=\{\delta_i \ ; \ i=1,...,l\}$ be a set of detectors, and each detector δ_i encodes the information of each other different environmental attribute, thus,

$$\delta_i: E \rightarrow V_i. \tag{2-7-1}$$

In term of the given detector, each environmental structure $e \in E$ will have an internal representation $(\delta_1(e), \ \delta_2(e),...., \ \delta_l(e))$; that is each structure e will be described by its particular ordered set of l attributes or detector values $\delta_i(e) \in V_i$. Thus, for an encoded chromosome of e, V_i can designate the set of alleles of locus i. Also V_i has different values $v_{i,j}$, $j=1,....$. For example, three detectors are given $(\delta_1, \ \delta_2, \ \delta_3)$, and three environmental structures are assumed $(e_1, \ e_2, \ e_3)$. If encoded chromosomes of three structures are as follows;

$$\delta\,(e_1) = (v_{1,1}, v_{2,1}, v_{3,2}),$$

$$\delta\,(e_2) = (v_{1,1}, v_{2,2}, v_{3,2}),$$

$$\delta\,(e_3) = (v_{1,1}, v_{2,2}, v_{3,1}),$$

then subsets of E which have attributes in common, are designated by using a new symbol '$*$" that indicates a wild-card character. Thus, $(v_{1,1}, \ *, \ *)$ designates the subset of all elements in E having the attribute $v_{1,1} \in V_1$. This ordered l-tuple that represents subsets of all elements, such as $(v_{1,1}, \ *, \ *)$, is named a *schema*. Hence, for the above example, three structures have a common schema $(v_{1,1}, \ *, \ *)$. Also, e_1 and e_2, are called instances of a schema $(v_{1,1}, \ *, \ v_{3,2})$. Schemata provide a basis for associating combinations of attributes with potential for improving current performance. Goldberg proofed this improvement mechanism in genetic algorithms by the schema theorem in [Goldberg 89].

Schema Theorem

The Goldberg's schema theorem of simple genetic algorithms is represented as

following expression.

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left[1-p_c\frac{\delta(H)}{l-1}-o(H)p_m\right], \qquad (2\text{-}7\text{-}2)$$

where,

H : a schema,

$m(H, t)$: the number of instances of the schema H in the population at time t,

$f(H)$: the average fitness of the instances of H at time t,

$\bar{f}$: the average fitness of all strings in the population at time t,

$\delta(H)$: the length of the schema H, that is the distance between the first and last specific position,

$o(H)$: the order of the schema H, that is simply the number of fixed positions (the number of 'not $*$') presented in the schema,

l : the length of strings,

p_c : crossover rate,

p_m : mutation rate,

$m(H, t+1)$: the expected number of instances of the schema H in the next generation,

This expression shows the lower bound of the expected number of a particular schema H in the next generation. This also implies that short, low-order schemata having above-average fitness values, are exponentially increasing in subsequent generations. For changing environments, more adapted schemata are increasing by genetic algorithms based on the schema theorem. Therefore , when the environment is changed from E to E', the GAPS can do effective re-adaptation using commonly adaptive schemata for the environment E and E'.

2.8 Conclusions

In this Chapter, the Genetics-based Adaptive Problem Solvers(GAPS) is defined. Moreover to clarify characteristics of GAPS, an outline of problem solving is described in section 2.2, and Newell and Simon's human problem solving in cognitive science field, and some conventional AI planning methods are reviewed in section 2.3. Section 2.4 discusses the problem being solved, and formalizes the adaptive problem solving and defines the GAPS. Mechanisms of genetic algorithms as evolutionary computation scheme is summarized in section 2.5. The simple classifier systems as adaptive reinforcement learning scheme is formalized in section 2.6. Required robustness of GAPS in uncertain and non-stationary problem domains is discussed in section 2.7.

References of Chapter 2

[Adeli and Hung 95] Adeli, H. and Hung, S.: "Machine learning: neural networks, genetic algorithms, and fuzzy systems", John Wiley & Sons, pp.128-135, 1995.

[Agre and Chapman 90] Agre, P.E. and Chapman, D.: "What are plans for?", in Designing Autonomous Agents, Bradford-MIT, pp.17-34, 1990.

[Amarel 87] S. Amarel : "Problem Solving", in Encyclopedia of Artificial Intelligence, Shapiro, Stuart C., (ed.), John Wiley & Son Inc., 1987.

[Brooks 91] Brooks, R. A: Intelligence Without Reason, International Joint conference of Artificial Intelligence, pp.569-595 1991.

[Connell and Mahadevan 93] Connell, J. H. and Mahadevan, S. : "Introduction to Robot Learning", in Robot Learning, Kluwer Academic Publishers, pp.1-4, 1993.

[deKleer 86] deKleer, J. : "An Assumption-based TMS", Artificial Intelligence, Vol.28, pp.127-162, 1986.

[Fikes and Nillson 71] Fikes and Nillson : "STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving", Artificial Intelligence, 2, pp.189-208.

[Goldberg 89] Goldberg, D.E. :"Genetic Algorithms in Search, Optimization & Machine Learning", Addison-Wesley, pp.1-25, 1989

[Grefenstette 88] J.J. Grefenstette : "Credit Assignment in Rule Discovery Systems based on Genetic Algorithms", Machine Learning, Vol.3, pp.225-245, 1988.

[Hayes-Roth and Hayes-Roth 79] Hayes-Roth, B. and Hayes-Roth, F., : "A Cognitive Model of Planning", Cognitive Science, pp.275-310, 1979.

[Hoffmeister and Back 91] Hoffmeister , F. and Back, T. : "Genetic Algorithms and Evolution Strategies—Similarlities and Differences", in Schwefel, H.P. and Manner, R. (ed.), Parallel Problem Solving from Nature, Springer-Verlag, pp.455-469, 1991.

[Holland 75] Holland, J.H. :"Adaptation in Natural and Artificial Systems", University of Michigan Press, pp.20-120, 1975.

[Holland 85] J.H. Holland : "Properties of the Bucket Brigade Algorithm", Proceedings of 1st ICGA, pp.1-7, 1985.

[Holland et al. 86] Holland, J.H., Holyoak, K.J., Nisbett, R.E. and Thagard, P.R. :"Induction", The MIT Press, pp.102-150, 1986.

[Hopfield and Tank 85] Hopfield, J.J. and Tank, P.W. : "Neural Computation of Decisions in Optimization Problems", Biological Cybernetics, Vol.81, pp.141-152, 1985.

[Ishida and Kuwabara 92] Ishida, T. and Kuwabara, K. : "Distributed Artificial Intelligence(1):Cooperative Problem Solving, Journal of Japanese Society for Artificial Intelligence, Vol.7, No.6, pp.945-954, 1992. (in

japanese)

[Kaelbling and Rosenschein 90] Kaelbling, L.P. and Rosenschein, S.: "Action and Planning in Embedded Agents", in Designing Autonomous Agents, MIT press, pp.35-48, 1991.

[Kido 93] Kido, T. : "Hybrid search by Genetic Algorithms", in Genetic Algorithms, Kitano, H. (ed.), Sngyo-Tosho, pp.61-88, 1993. (in Japanese)

[Kirkpatricks et al. 83] Kirkpatricks, S., et al. : "Optimization by Simulated Annealing", Science, Vol.220, pp.671-680, 1983.

[Kobayashi 92] Kobayashi, S. :"Planning Problems and Artificial Intelligence : An Overview", Journal of Japanese Society for Artificial Intelligence, Vol.7, No.6, pp.965-969, 1992. (in japanese)

[Kolodner, Simpson and Sycara 85] Kolodner, J.L., Simpson, R.L, and Sycara, K. : "A Process Model of Case-Based Reasoning in Problem Solving", IJCAI-85, pp.284-290, 1985.

[Koza 92] Koza, J.R. : "Genetic Programming: On the Programming of Computers by Means of Natural Selection", The MIT Press, pp.73-77, 1992.

[Maes 90] Maes, P.: "Situated Agents can have Goals", in Designing Autonomous Agents, MIT press, pp.49-70, 1990.

[Maes 95] Maes, P. : "Modeling Adaptive Autonomous Agents", in Artificial Life: An Overview, MIT press, pp.135-162, 1995.

[McDermott 78] McDermott, D.V., : "Planning and Acting", Cognitive Science, 2, 1978.

[Michell, Mahadevan and Steinberg 85] Michell, T.M., Mahadevan, S., and Steinberg, L.I. : "LEAP: A Learning Apprentice for VSLI Design", IJCAI-85, pp.573-580, 1985.

[Narendra and Thathachar 91] K.S. Narendra and Thathachar, M.A.L. : "Learning Automata :An Introduction", Prentice Hall, 1991.

[Naruse 94] Naruse, K. : "A Study on Reinforcement Learning for Multi-Agent Systems", Doctoral Thesis, Faculty of Engineering, Hokkaido Univ., pp.36-39, 1994 (unpublished).

[Newell and Simon 63] Newell, A. and Simon, H.A., : "GPS: a Program that Simulates Human Thought", in Feigenbaum and Feldman (ed.), Computers and Thought, pp.279-296, 1963.

[Newell and Simon 72] Newell, Allen and Simon, Herbert A.: "Human problem solving", Prentice-Hall, pp.1-140, 1972.

[Newell, shaw and Simon 63] Newell, A., Shaw,J.C., and Simon, H.A., : "Chess-Playing Programs and the Problem of Complexity", in Feigenbaum and Feldman (ed.), Computers and Thought, pp.39-70, 1963.

[Sacerdoti 73] Sacerdoti, E.D. : "Planning in a Hierarchy of Abstraction Spaces", IJCAI'73, 1973.

[Sacerdoti 77] Sacerdoti, E.D. : "A Structure for Plans and Behavior", Elsevier-North Holland, 1977.

[Samuel 63] Samuel, A.L., : "Some Studies in Machine Learning using the Game of Checkers", in Feigenbaum and Feldman (ed.), Computers and Thought, pp.71-108, 1963.

[Stefik 81] Stefik, M.J., : "Planning with Constraints ", Artificial Intelligence, 16, pp.111-140, 1981.

[Sussman 75] Sussman, G.J., : "A Computer Model of Skill Acquisition, New York, pp.105-109, 1975.

[Sutton 88] R.S. Sutton : "Learning to Predict by the Methods of Temporal Differences, Machine Learning, Vol.3, pp.9-44, 1988.

[Suzuki 92] Suzuki, K. : "Hypercube Learning of Autonomous Distributed Systems", Doctoral Thesis, Faculty of Engineering, Hokkaido Univ., pp.4-26, 1992 (unpublished).

[Tate 77] Tate, A. : "Generating Project Networks", IJCAI'77, 1977.

[Tate and Whiter 84] Tate, A. and Whiter, A.M., : "Planning with Multiple Resource Constraints and an Application to a Naval Planning Problem", 1st conf. on Applications of AI, 1984.

[Tate, Hendller, and Drummond 90] Austin Tate, James Hendller, and Mark Drummond : "A Review of AI Planning Techniques", in Readings in Planning, Morgan Kaufmann, pp.26-49, 1990.

[Vere 83] Vere, S. : "Planning in Time: Windows and Durations for Activities and Goals", IEEE Trans. on Pattern Analysis and Machine Intelligence, 3, pp.246-267, 1983.

[Vere 87] S. Vere : "Planning", in Encyclopedia of Artificial Intelligence, Shapiro, Stuart C., (ed.), John Wiley & Son Inc., 1987.

[Watkins and Dayan 92] C.J.C.H. Watkins and P. Dayan : "Technical Note :Q-Learning", Machine Learning, Vol.8, pp.55-68, 1992.

[Wilkins 83] Wilkins, D.E. : "Representation in a Domain-Independent Planner ", IJCAI'83, pp.733-740, 1983.

[Wilson 94] Wilson, Stewart W., : "ZCS: A Zeroth Level Classifier System", Evolutionary Computation, Vol. 2, No.1, pp. 1-18, 1994.

CHAPTER 3

Extending Mechanisms for Developing the Advanced GAPS

3.1 Introduction

As I defined in chapter 2, genetics-based adaptive problem solvers consist of genetic algorithms as an adaptive searcher and classifier systems as a reinforcement learner. The canonical genetic algorithms and simple classifier systems are very useful tools to construct the adaptive problem solving systems, because the mechanics of those are surprisingly simple and flexible. The other hand, however, simple genetic algorithms and simple classifier systems have some latent difficulties on implementation. To resolve these difficulties, a lot of researches have focused on the great variety of extending mechanisms of those.

In chapter 3, therefore, I propose extending mechanisms in frameworks of genetic algorithms and classifier systems to develop the advanced GAPS, In this thesis, I especially focus on the mechanisms of classifier systems. Let us start with describing some difficulties on their implementation in section 3.2. Section 3.3 shows the evolutionary synthesis mechanism of classifier systems architectures. By this synthesis mechanism, appropriate architectures of classifier systems for a give certain problem are synthesized. To realize the evolutionary mechanism, genetic algorithms are applied.

Frequently pointed issues of difficulties are something about the internal representation of chromosomes and classifier rules. In section 3.4, therefore, the extensive approach of rule representations in classifier systems

is provided. For this purpose, I propose and define a concept of new representational type of classifier, named masking classifiers.

3.2 Some Difficulties on Implementation of Simple Genetic Algorithms and Simple Classifier Systems

3.2.1 Focused Issues on Genetic Algorithms

Many researches have been carried out since Holland proposed a framework of genetic algorithms as an adaptive mechanism in artificial systems. These works can be roughly distinguished as following three approaches [Schaffer 89] [Belew and Booker 91] [Forrest 93] [Eshelman 95] [Rawlins 91] [Whitley 93];

- Theoretical analysis of the canonical genetic algorithms.
- Propositions of extensive mechanisms.
- Applications.

In these approaches, the extensive mechanisms of genetic algorithms were introduced to resolve some difficulties pointed out by the theoretical analysis, and to improve the abilities of genetic algorithms. Theses extensive approaches are related to following issues;

- *GA-hard, called deceptive, problems and the schema theorem:* The schema theorem provides a description for the growth rate of schemata that depends on the observed relative fitness of the schemata represented in the population. However, there are some problem conditions that might be difficult for genetic algorithms to solve. Thus, In such problem conditions, the global optimum cannot searched because of its searching direction based on the schema theorem, and solutions are converged to the deceptive optimum. This problem condition is called the deceptive problem.
- *Convergence and genetic diversity:* Genetic algorithms are randomized parallel search methods modeled on natural selection. Natural selection has to use diversity in a population to produce adaptation. If there is no diversity, then there is nothing for natural selection to work on. However

genetic algorithms work such that a population is converged to the fitted scheme. So, there is a tradeoff between convergence and diversity.

- *Representation and coding:* One of the main advantages of genetic algorithms approaches is the simple representational method that is separated from true representational structures called phenotypic representations. This internal representational method is called genotypical representations, and is generally using simple binary alphabets. However, for a particular problem class, more complex representation might improve the search performances of genetic algorithms.
- *Genetic operators*: In the canonical genetic algorithms, three genetic operators such as selection, crossover and mutation are applied. Since these operators have very simple mechanisms, the implementation of these is greatly easy. Instead of this simplicity, by employing more complex operators, the abilities of genetic algorithms might be improved. Moreover, the parameters of genetic operators also affect the search performances. So, the optimal parameter set must be determined.

Some of these researches are summarized in table 3.1.

Table 3.1 Some extensive approaches on genetic algorithms

Deceptive problems	Messy GA [Goldberg, Deb and Korb 91] Augmented GA [Grefenstette 93]
Convergence and genetic diversity	Population Sharing [Deb and Goldberg 89] Generation Gap [DeJong and Sarma 93] Population Sizing [Goldberg, Deb and Clark 93]
Representation and coding	Grammar-based GA [Antonisse 91] Real-Coded GA [Wright 91]
Genetic operators	Adaptive Mutation [Whitley and Hanson 89] Uniform Crossover [Syswerda 89] Multi-point Crossover [Spears and DeJong 91] CHC [Eshelman 91]

3.2.2 Key Issues on Classifier Systems

A classifier system is a learning system in which a set of condition-action rules called classifiers compete to control the system and gain credit based on the system's receipt of reinforcement from the environment. The original concept of classifier systems is due to Holland, who described it most completely in [Holland et al. 86]. Responding to perceived shortcomings of existing artificial intelligence systems, particularly with regard to adaptation and practical induction, Holland laid out a competitive/cooperative message-passing framework addressing desiderata including temporal credit assignment under conditions of sparse or delayed reinforcement, distributed and generalized representations of complex categories, default responses subject to exceptions, and graceful adaptation of system knowledge through gradual confirmation/disconfirmation of hypotheses. However, given the relative success of genetic algorithms in optimization, issues of classifier systems have often been treated with less interest.

Recently, researcher's interests about the abilities of classifier systems are increasing, and in 1992, *The First International Workshop on Learning Classifier Systems* was held [Smith 94]. Many growing research activities on classifier systems are achieved until today. The perspective of these researchers is that classifier systems are intended as a general approach to the induction of needs-serving behavior in uncertain or dynamic environments. To clarify these perspectives, lots of application to certain problems and theoretical works were done. As the common view, the classifier system is more of an approach than a method. That is, the classifier system is a set of conceptual details, rather than a set of algorithmic details. Clarification of these conceptual details, therefore, became a central focus of past researches. In these studies, the following key issues are treated.

- **Modeling and analysis of classifier system architectures:** The

Holland's original classifier system proposes a framework of genetics-based machine learning system, but doesn't indicate the "standard" classifier system. For all problems, the original classifier system does not consistently work well. Therefore, some extended or oppositely simplified classifier system architectures were defined, and features and characteristics of those are analyzed.

- **Representations:** There is a great deal of latitude in possible representations for the classifier system. Surprisingly, the typical {1, 0, #} representation is not felt to be essential to the classifier system paradigm. The consensus is that representation could be determined for the problem at hand. However, choice of representation in the classifier system remains a key issue because it affects cooperation, the discovery process by genetic algorithms, and overall system performance.
- **Decision Making**: The classifier system works as a reactive system that controls stimulus-response processes. So, the system has to decide an action for a certain state condition by any decision making mechanism. Generally, this mechanism is realized by a conditional matching process and biding process. Some extensive approaches for these processes are also proposed.
- **Credit Assignment:** Typically, classifier systems are associated with reinforcement learning problems. A set of classifier rules in the system is reinforced by any credit assignment mechanism. The implementation of applied credit assignment mechanisms largely affects its learning performance. The bucket brigade algorithm and the profit sharing plan are commonly associated with temporal credit assignment and reinforcement learning in classifier systems. The bucket brigade is simply seen as one of growing classes of temporal credit assignment schemes. Although credit assignment emerged as a central issue for classifier systems, no particular credit assignment scheme is defined as standard scheme in classifier systems.
- **Rule Discovery**: A rule discovery mechanism is one of the important elements in machine learning system because all of the feasible rules cannot be memorized in a set of classifier rules. To find a more better rule, a new

rule that is not in a current rule-base , must be generated and added in the rule-base. Genetic algorithms, typically, are used as a new rules discovery mechanism. As there are some variation of genetic operators in genetic algorithms, the learning performance is greatly influenced by applied rule discovery mechanisms.

- **Cooperation:** The cooperative aspect of a classifier system population was a defining characteristic of the classifier system approach. When genetic algorithms are used in optimization, population members are functionally independent. These population members only interact in a competitive fashion, through the genetic algorithms selective process. In a classifier system, population members are interdependent. They must cooperate to improve overall system performance, while also competing in the selective process. Therefore, the demands placed on genetic algorithms in a classifier system are quite different from those placed on genetic algorithms used in optimization. In a classifier system, genetic algorithms must maintain a functional, diverse population of individuals that are co-adapted to one another.

Some of these studies are summarized in table 3.2.

Since the original classifier systems provide the conceptual framework, domain-dependent implementations are required for particular problems. Therefore, using the simple classifier system in GAPS, some difficulties occur on its implementation. I then focus on the following two difficulties, and propose extensive mechanisms in following sections to resolve these difficulties.

1. How the appropriate classifier system structures having high learning performance is to be designed?
2. How the huge size of rule space can be represented in the restricted area of memory?

Table 3.2 Some extensive approaches on classifier systems

Modeling and analysis	GOFER [Booker 88]
	Rosetta [Smith and Wilson 89]
	Alecsys [Dorigo and Sirtori 91]
	ZCS [Wilson 94]
	COGIN [Greene and Smith 94]
Representations	VCS [Shu and Schaeffer 89]
	Attribute-based Representation [Booker 91]
	Relational Schemata [Collard and Escazut 95]
	Fuzzy Classifier [Rendon 91]
Decision Making	Noisy Auction [Goldberg 89]
	Variance-sensitive Biding [Wilson and Goldberg 89]
Credit Assignment	Back Propagation [Belew and Gherrity 89]
	Dyna-Q-CS [Roberts 93]
	Q-Credit Assignment [Giani, Baiardi and Starita 94]
Rule Discovery	Triggered Rule Discovery [Booker 89]
Cooperation	Coupled Sequences of Classifiers [Riolo 89]
	Cooperative Population [Horn, Goldberg and Deb 94]

3.3 Evolutionary Synthesis of Classifier Systems Architectures by Genetic Algorithms

3.3.1 Introduction

The machine learning is considered as one of most important paradigms to realize an intelligent and flexible system. In machine learning scheme, the main interest of researchers is how the autonomous learning mechanisms are realized. The autonomous learning system is the unsupervised learning system that learns given tasks based on the interactions between the system and the environment. Inductive learning, deductive learning, analogical reasoning, and heuristic learning are well known as machine learning algorithms. Recently, the reinforcement learning is attracting attention as a profitable machine learning algorithm. The reinforcement learning is typical autonomous learning scheme by which appropriate action patterns are learned. This reinforcement learning is performed based on the given environmental rewards for executed actions. Therefore, the complete world model is not necessary to construct, and the reinforcement learning system is expected to solve the difficult problems in which external environments cannot be significantly modeled in the system, or environments are dynamically changing. The reinforcement learning might be possible to construct robust learning systems, and be applied to many engineering areas. Especially, considering the autonomous robot agent, since all of state variables in the real environment cannot be perceived, the system must execute any process under uncertainty. As the other problem, there is the delayed reward problem. Thus, an immediate reward is not consistently given for an executed action. There might be a situation where a collective reward is given for a sequence of accomplished actions. The reinforcement learning scheme is said to resolve these uncertainty problem and delayed reward problem [Yamamura et al. 95].

To realize reinforcement learning scheme, some approaches such as

Samuel's checker program[Samuel 59], Stochastic Learning Automata (SLA) [Narendra and Thathachar 91], Temporal Difference Algorithm (TD(λ)) [Sutton 88] and Q-Learning [Watkins 92], were proposed. I focused on Classifier Systems proposed by Holland [Holland et al. 86], and examined learning performances of classifier systems by constructing some learning systems for engineering applications[Kawakami and Kakazu 93] [Kawakami and Kakazu95a]. Classifier systems are well known as a profitable machine learning scheme, and have been implemented in autonomous agents that solve complex problems or learn adaptive action patterns. A classifier system particularly useful in rule based problem solving tasks, and combined with an evolutionary approach to generate new rules it is a powerful search technique [Holland et al. 86] [Goldberg 89] [Wilson and Goldberg 89].

Since reinforcement learning algorithms containing classifier systems have flexible frameworks, these can be applied broader class of problems. The other hand, as I mentioned in section 3.2, domain-depended implementations of these frameworks are required. In section 3.3, I focus on how the appropriate classifier system structure having high learning performance is to be designed. Thus, good learning parameters such as a reward function or resolution of sensors, must be suitably determined for given tasks. Although the parameter setting dominates a system architecture and largely affects learning performances of systems, it is not at all clear how this might be done. That is very important problem, and is corresponding to the bias determination problem in inductive learning or the architecture determination problem in artificial neural networks. Therefore, the general construction method of reinforcement learning system architectures is needed to resolve these difficulties. However, the suitability of system architectures greatly depends upon given problems and complex interactions among many elements in the systems. Hence, there is no deterministic method to generate a good system architecture.

Therefore, I propose an extensive approach of evolutionary synthesis of simple classifier system architectures that is well known as genetics-based machine learning system. This synthesis mechanism is realized by using genetic

algorithms as an evolutionary search technique. To examine our proposed method, evolutionary synthesis technique is applied to classifier systems that learn motion planning tasks of robot manipulator. The obtained results of computer simulation indicate the usefulness of this approach. In this approach, simple classifier systems defined in chapter 2 are treated as a subjective reinforcement learning system.

3.3.2 Related Works

Generally speaking, the learning performance depends on the interaction between a given task and a system architecture. There have been many works that optimize the architecture of learning systems. Especially, several studies that tune architectures or parameters of artificial neural networks by using genetic algorithms, have been done. In artificial neural networks, the parameters are the number of units, the connection information among units, the threshold values of units, and initial weighted values of connections. For example, [Montana and Davis 89] and [Whitely and Hanson 89] are primary studies evolve the weights and threshold values by genetic algorithms. Network architectures involving the connection information among units, are encoded into genetic strings and are evolutionarily tuned in [Harp, et al. 89], [Miller, et al. 89], [Gruau 93] and [Kitano 93]. [Belew, et al. 92] is an approach that evolves learning parameters of the back-propagation algorithm[Rumelhart, Hinton and Williams 86]. Ackley and Littman proposed the Evolutionary Reinforcement Learning(ERL) concept by combining genetic algorithms and neural networks in integrated way [Ackley and Littman 92].

Unemi et al. attempted to evolve the learning parameters of Q-learning using look-up table by implementing genetic algorithms [Unemi, et al. 94].

In classifier systems, also, it was pointed out that some difficulties occur on implementations to certain problems, and appropriate system architectures must be designed to get high learning performances [Wilson and

Goldberg 89] [Schuurmans and Schaeffer 89]. As I reviewed in section 3.2, some extensive studies of components of classifier systems have been done to resolve above difficulties. Patel and Dorigo implemented learning classifier systems on a robot arm to learn simple light approaching tasks [Patel and Dorigo 94]. In their study, a treated robot arm is similar with this research's setting, but, a simulated robot arm learns behaviors by which an end-effector is getting closer to a light source by visual sensors. The aim of their research is to investigate the nature of some sensory information, so some classifier systems having different condition of sensors, are examined those performances. However, these different conditions of sensors are given by a designer. Though, the present study has a similar interest with their study, the approaches are quite different. The objective of the present study is evolutionary synthesis of architectures containing the sensor setting. Moreover, no work has been achieved to evolutionarily synthesize the whole of architectures of classifier systems. So, this approach seems to make a profitable view.

3.3.3 Definitions of Simple Classifier Systems to be Evolutionary Synthesized

Here, I define the system parameters that dominate an architecture of the simple classifier system to clarify the subjective to be evolved. For this purpose, let us start with reconfirming the defined simple classifier systems in chapter 2.

The simple classifier system *SCS* can be depicted as a set of interacting sub-systems, and be represented as following four-tuple:

$$SCS = < RB, PS, CAS, RDS >, \qquad (3\text{-}3\text{-}1)$$

where, *RB* is a rule-base system, *PS* is a performance system, *CAS* is a credit assignment system, and *RDS* is a rule discovery system.

RB: The Rule-base System

In the rule-base system, a set of classifiers exists. The classifiers play as

reactive rules in a reactive planning process. The rule-base system *RB* is defined as following four-tuple,

$$RB = < CF, I, RGF, p_R >, \quad (3\text{-}3\text{-}2)$$

where, $CF=\{cf_i, ; i = 1,2,...,N\}$ is a set of classifiers, $I=\{0,1,\#\}$ is a set of characters used in a classifier, $RGF: I \times p_R \rightarrow CF$ is a rule generating function that generates an initial *CF*, $p_R=\{ N, l_C, l_a, P_\# , S_0\}$ is a set of internal parameters about classifiers. A classifier cf_i ($\in$ CF) is a string rule that consists of a condition part $cf^c_i \in \{0, 1, \#\}^{lc}$, an action part $cf^a_i \in \{0, 1\}^{la}$, and a fitness value $Scf_i \in$ **Reals** called 'strength' likely to production rules, i.e., $cf_i = \{ cf^c_i , cf^a_i , Scf_i \}$. In a cf^c_i, '#' is used as a *'don't care'* symbol in the performance system. In parameter set p_R, N is the number of rules in *CF*, l_c and l_a are predetermined fixed lengths of the condition part and the action part respectively, $P_\#$ is a probability variable that controls the generated number of symbol "#" in a conditional part. The expected number of the symbol "#" in a cf^c_i , is $l_c * P_\#$. The initial strength values of all classifiers are equivalently settled by the parameter S_0.

PS: The Performance System

In the performance system *PS*, main processes of a stimulus-response cycle are executed. The performance system *PS* is defined as following eight-tuple;

$$PS=< CF, E, EM, EF, MF, RCF, DF, A >, \quad (3\text{-}3\text{-}3)$$

where *CF* is a set of classifiers defined in *RB*, *E* is a set of external environmental states, and $EM \in \{0, 1\}^{l_{EM}}$ is a set of feasible internal messages, and $EF: E \rightarrow EM$ is an encoding function, $MF: CF \times EM \rightarrow CF_M$ is a matching function by which a set of matched classifiers $CF_M \subseteq CF$ is extracted, $RCF: CF_M \rightarrow CF_A$ is a rule competition function by which an active set of classifiers $CF_A \subseteq CF_M$ is generated, $DF: CF_A \rightarrow A$ is a decoding function, and $A=\{a_j ; j = 1,2,...,N_A\}$ is a set of executable actual actions of the agent.

CAS: The Credit Assignment System

In *CAS*, strength values of classifiers are revised, such that the useful classifiers for the given task tend to have higher strength values. The credit assignment system *CAS* is defined as following triple;

$$CAS =< CF, R, RF >, \tag{3-3-4}$$

where CF is a set of classifiers, R is a set of environmental rewards, and RF:$CF \times R \rightarrow CF$ is a reinforcement function. Let $r^t \in R$ be a given environmental reward at time t, then the strength value of an active classifier ${CF_A}^t$ revised from ${S_{CF_A}}^t$ to ${S_{CF_A}}^{t+1}$ by following expression.

$${S_{CF_A}}^{t+1} = {S_{CF_A}}^t + RF(r^t), \tag{3-3-5}$$

where ${S_{CF_A}}^t$ is a strength value of CF_A at time t. In general, as reinforcement algorithms, the profit sharing method[Grefenstette 88] and the bucket brigade algorithm[Holland 85] are well known. In these two algorithms, a reinforcement function RF is expressed by following equations.

- profit sharing method:

$$RF(r^t) = \alpha\, r^t / |CF_A|, \tag{3-3-6}$$

where, α is a discount coefficient of given reward, $|CF_A|$ means the number of rules in CF_A.

- bucket brigade algorithm:

$$RF(r^t) = \alpha\, r^t - \beta\, {S_{CF_A}}^t + \gamma\, {S_{CF'_A}}^{t+1}, \tag{3-3-7}$$

where, α is a discount coefficient of given reward, β and γ are learning rate for strength updates under bucket brigade, and CF'_A is an activated classifier at time $t+1$.

RDS: Rule Discovery System

In *RDS*, new rules that are not in a current rule-base are generated and added to search more profitable rules. The rule discovery system *RDS* is defined as following four-tuple;

$$RDS =< CF, RDF, p_D, GA >, \tag{3-3-8}$$

where *CF* is a classifier set, *GA* is simple genetic algorithms defined in chapter 2, $RDF: CF \times p_D \times GA \rightarrow CF$ is a rule discovery function, and $p_D=\{\rho, \chi, \mu\}$ is a parameter set by which a rule discovery procedure is controlled. In p_D, ρ is a starting probability of rule discovery procedure, χ is a crossover rate, and μ is a mutation rate.

3.3.4 Evolutionary Architecture Synthesis Process

In this section, I describe an evolutionary method to synthesize good architectures of simple classifier systems by using genetic algorithms. Based on the above definitions of simple classifier systems, in this method, a system parameter set *P* that denotes an architecture, is defined as following expression.

$$\boldsymbol{P} = \{ p_R, EF, DF, RF, p_D \} \tag{3-3-9}$$

where, $p_R=\{ N, l_C, l_a, P\#, S_0\}$ is a parameter set about *RB*, *EF* and *DF* are an encoding function and a decoding function in *PS* respectively, *RF* is a reinforcement function in *CAS*, and $p_D=\{\rho, \chi, \mu\}$ is a parameter set of *RDS*.

Each different combination of these parameters constructs a system architecture that has different learning performance. Since there are tremendous number of combinations, it is very hard to find the best combination of parameters denotes a suitable system architecture having highest learning performance for a given task. This problem seems the function optimization problem in which each parameter value is an input variable and the system performance is the output of the function. Moreover, the system performance is greatly affected by interactions among parameters. Hence, it is difficult to apply the conventional optimization methods such as hill-climbing algorithm. Here, to synthesize a beneficial system architecture having high performance, a parameter set *P* is evolved by adopting genetic algorithms. Genetic algorithms are useful evolutionary multi-point search method, and the success for NP-complete optimization problems is reported [DeJong and Spears 89]. In genetic algorithms, however, the optimality of searched solutions cannot be proofed, and hence, the optimal solution might not be searched. It is not

always necessary that the optimum architecture is required. A near-optimal architecture would be sufficient for implementing in the agent. Under this required condition, genetic algorithms are effective tools.

3.3.4.1 The Evolutionary Synthesis Process

In this approach, one combination of parameters corresponds to a chromosome, and genetic operators are applied based on evaluated fitness values of decoded chromosomes in a population. The evolutionary synthesis process of classifier systems architectures is illustrated in figure 3.1.

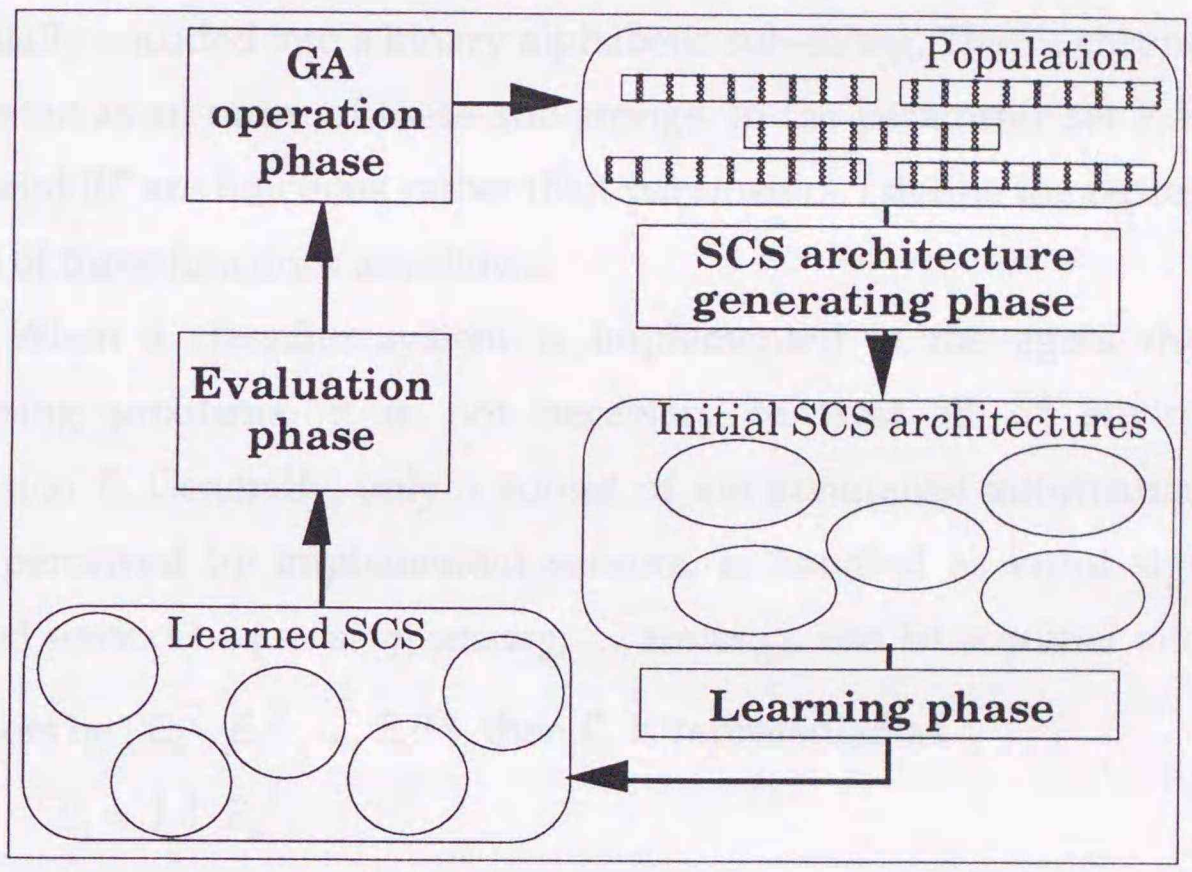


Figure 3.1 The evolution cycle of Simple classifier system architectures by GA. In this figure, the SCS means simple classifier systems.

That is, this evolutionary synthesis process is executed as following procedures.

Step 1) An initial population of chromosomes is randomly created.

Step 2) Each chromosome is decoded to generate an initial structure of each classifier system.

Step 3) Each unlearned classifier system learns the same task for the same learning trials.

Step 4) Each learned classifier system is evaluated according to its learning result.

Step 5) Genetic operators are applied to the current population based on each system's evaluation value, and then a new population is produced.

Step 6) *If* the termination condition is satisfied, then this evolutionary process terminates. *Otherwise*, go to step 2.

3.3.4.2 Representation of a Chromosome

The development of a representation method for classifier system architectures is an important problem. In this approach, each parameter in P is sequentially encoded into a binary alphabetic sub-string. Then a chromosome is represented as an order of these sub-strings. In the parameter set P, however, EF, DF and RF are functions rather than parameters. I define the representation method of three functions as follows.

When a classifier system is implemented in the agent that solves engineering problems, it is not necessary to treat all of environmental information E. Generally, only a subset of environmental information E_s that can be perceived by implemented sensors, is handled as input signals. Let equipped sensors be $\{ sensor_1, sensor_2, ..., sensor_n \}$, and let acquired information via sensors be $\{ E_s^1, E_s^2, ..., E_s^n \}$, then E_s is represented as,

$$E_s = \bigcup_j E_s^j . \tag{3-3-10}$$

In a simple classifier system, consequently, each acquired sensor value E_s^j is encoded into a part of an internal environmental message EM_j through a detector. An internal environmental message EM is generated as a sequence of EM_j, that is,

$$EM = (EM_1, EM_2, .., EM_j, .., EM_n) . \tag{3-3-11}$$

Here, I define the encoding function EF such that EF dominates the length of each binary string EM_j. Let n_s^j be the number of used bits in EM_j, then EF is represented as,

$$EF(E_s^j) = EM_j \in \{0,1\}^{n_s^j} . \tag{3-3-12}$$

Also, n_S^j implies the resolution of the $sensor_j$ in the simple classifier system. More precisely, let $[low_S^j, high_S{}^j]$ be a measurable range of the $sensor_j$, then *EF* divides this range into $2^{n_S^j}$ zones, and assigns the sensor value $E_s{}^j$ to a certain binary code having the n_S^j bit-length. In stead of encoding the *EF* itself, each n_S^j is encoded into a chromosome. For example, if n_S^j=3, then a measurable range$[low_S^j, high_S^j]$ is divided into eight areas, and each area is assigned to an alternative binary code having three bit-length. Even if sensed value $E_s{}^j$ takes discrete values, the same decoding processing is possible. For example, when $E_s{}^j$ takes two discrete values [*ON*, *OFF*] and n_S^j=2, then the signal 'ON' is assigned to two binary codes (00) and (01), and the signal 'OFF' is assigned to strings (10) and (11). Consequently, the fact n_S^j=0 implies that a sensed value $E_s{}^j$ of $sensor_j$ is ignored.

The decoding function *DF* is encoded on a chromosome in the same manner of *EF*. Let $\{actuator_1, actuator_2, ..., actuator_m\}$ be a set of implemented actuators in the agent, and let A_k be a set of executable actions by the $actuator_k$, then a set of feasible actions *A* of the agent is expressed as,

$$A = \bigcup_k A_k \tag{3-3-13}$$

The *DF* is defined to dominate the resolution of controller of each actuator. Let $[low_a^k, high_a^k]$ be a fixed range of control signal of $actuator_k$, and let $n_a{}^k$ be the number of bits used by a controller of $actuator_k$, then a controllable range $[low_a^k, high_a^k]$ is divided into $2^{n_a^k}$ areas. Therefore, each $n_a{}^k$ is encoded into a chromosome instead of the *DF* itself. Figure 3.2 illustrates the encoding process by *EF* and the decoding process by *DF*.

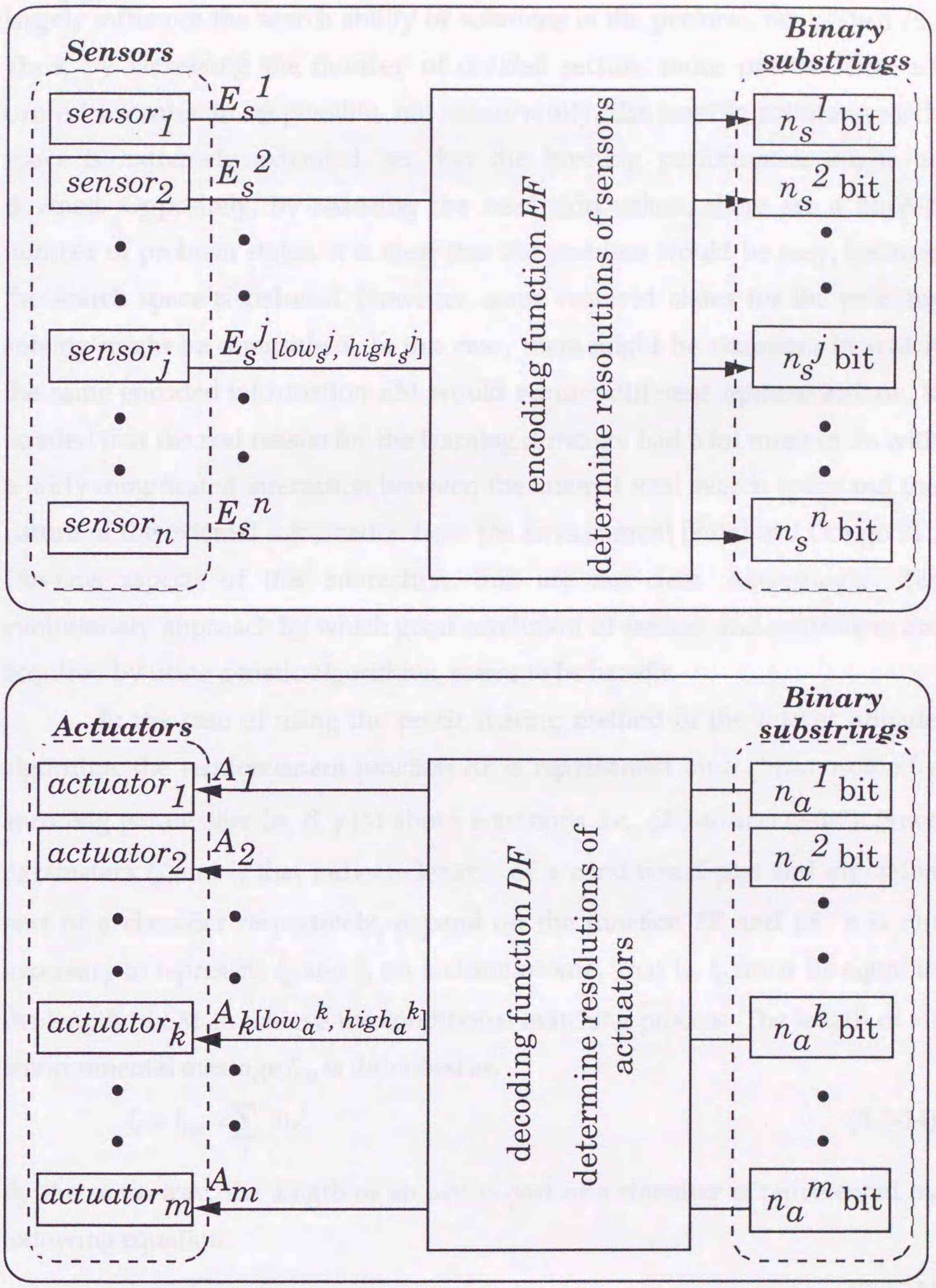


Figure 3.2 The encoding process by *EF* and the decoding process by *DF*.

In simple classifier systems, the resolutions of sensors and controllers largely influence the search ability of solutions in the performance system *PS*. Thus, by increasing the number of divided sectors, more precise sense-act control becomes to be possible, but concurrently, the feasible solution search space is extremely extended, so that the learning performance might be downed. Oppositely, by reducing the resolution values, there are a limited number of problem states. It is clear that the problem would be easy, because the search space is reduced. However, some required states for the problem solving might be diminished. In this case, there might be situations in which the same encoded information *EM* would require different optimal actions. It pointed that the real reason for the learning difficulty had a lot more to do with a fairly complicated interaction between the internal total search space and the nature of the external information from the environment [Patel and Dorigo 94]. Obvious aspects of this interaction, but, are not clear. Accordingly, this evolutionary approach by which good resolution of sensors and controllers are acquired by using genetic algorithms, seems to be benefit.

In the case of using the profit sharing method or the bucket brigade algorithm, the reinforcement function *RF* is represented on a chromosome by encoding parameters $\{\alpha, \beta, \gamma\}$of above equations, i.e., (3-3-6) and (3-3-7). Since parameters l_c and l_a that indicate lengths of a conditional part and an action part of a classifier respectively, depend on the function *EF* and *EF*, it is not necessary to represent l_c and l_a on a chromosome. That is, l_c must be equal to the length of *EM* to achieve the conditional matching process. The length of an environmental message l_{EM} is described as,

$$l_C = l_{EM} = \sum_j n_s^j \qquad (3\text{-}3\text{-}14)$$

By the same way, the length of an action part of a classifier is represented by following equation.

$$l_a = \sum_k n_a^k \qquad (3\text{-}3\text{-}15)$$

Based on the above setting, a chromosome by which a simple classifier system architecture is generated, is represented as figure 3.3.

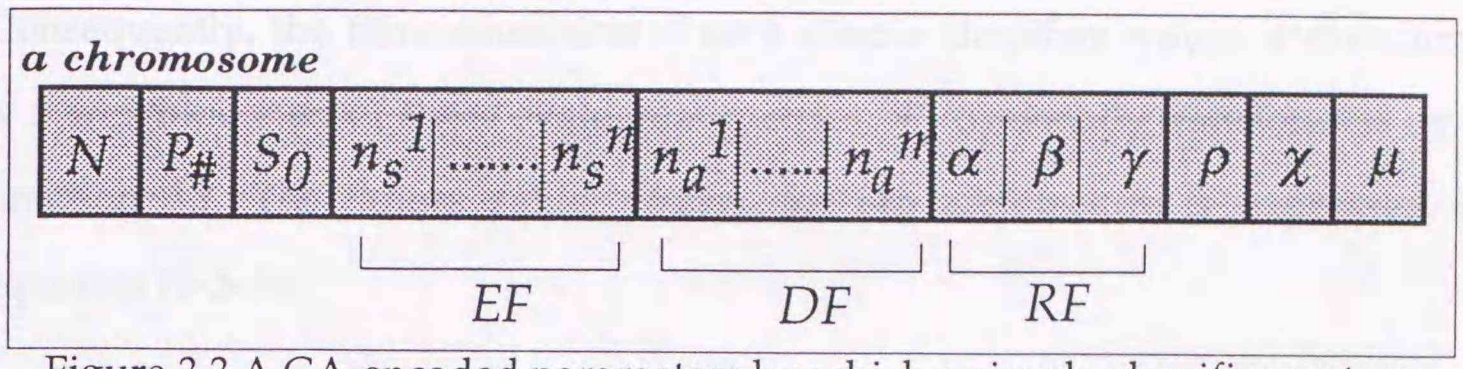


Figure 3.3 A GA-encoded parameters by which a simple classifier system architecture is determined.

3.3.4.3 Evaluation of Simple Classifier System Architectures

To apply genetic algorithms to the evolutionary synthesis process, the performance of each simple classifier system architecture has to be evaluated. There might be some evaluating methods. In this approach, I define evaluation items as follows;

1. ***Learnability***: The quality of a solution led by each learned simple classifier system, is evaluated.
2. ***Convergency***: The reinforcement learning is an on-line mapping scheme between a set of environmental conditions and a set of feasible appropriate actions of the agent. The task of reinforcement learning systems such as simple classifier systems is to learn a relevant stimulus-response pattern by which expected total reward for long term is maximized. In the learned system, a correct action is constantly selected for the same environmental state by the reinforced action pattern. The performance of a reinforcement learning system becomes higher, the faster the action pattern converges. So, the required learning trials to converge is evaluated as the convergency of each simple classifier system.
3. ***Robustness***: Another important feature of reinforcement learning systems is the robustness. The robustness means that a learned system can successfully manage a similar task or a partially changed environment. The quality of a gotten solution for a similar task to the learned task, is

evaluated as the robustness of each classifier system.

Consequently, the fitness function of each simple classifier system architecture is a weighted sum of these evaluation metrics, Ψ_j, optionally transformed by a function Ω_j. The fitness value, $fitness_i$, for i-th architecture is expressed in equation (3-3-16).

$$fitness_i = \sum_j^n e_j \, \Omega_j (\Psi_j (SCS_i)), \tag{3-3-16}$$

where, e_j is a weighted coefficient and SCS_i is i-th simple classifier system architecture.

3.3.5 Experimental Verification

3.3.5.1 Motion Planning Problems of a Robot Manipulator

To examine the proposed approach, simple classifier systems are applied to learn the motion planning task of robot manipulators, and the evolutionary synthesis experiments of architectures are carried out. In this experiment, two simple tasks of 2-link planar manipulator shown in figure 3.4, are learned by classifier systems. The motion planning is carried out in uncertain problem domain. That is, the planning agent does not initially know any information of a goal position nor obstacles. A configuration of the manipulator is represented by a vector of joint angles, q, that is,

$$q = (q_1, q_2)^T, \tag{3-3-17}$$

where q_i is a joint variable and T is a transposed operator. I assume that a measurable range of each joint variable, q_i, is fixed at $[0, 2\pi]$. Motions of a robot manipulator are controlled by a displacement vector of joint angles, Δq, at each time step.

$$\Delta q = (\Delta q_1, \Delta q_2)^T. \tag{3-3-18}$$

A controllable range of each Δq_i at unit time is also fixed at $[-\pi/6, \pi/6]$.

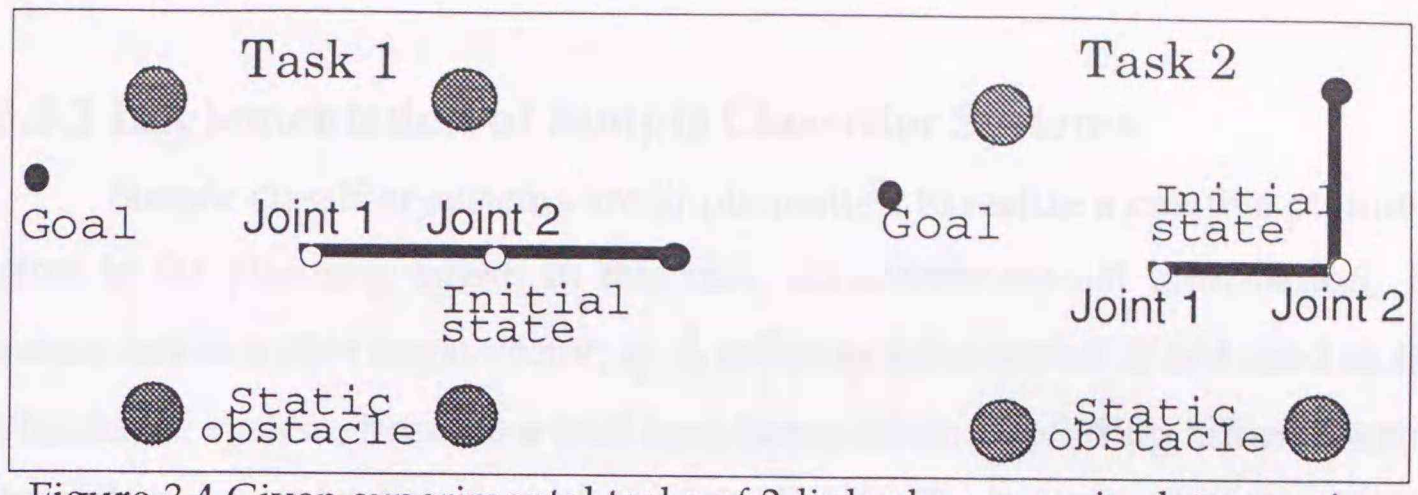


Figure 3.4 Given experimental tasks of 2-link planar manipulator motion planning.

To practice a reactive planning, some sensors are implemented on the agent. Each joint variable, q_i, is perceived by proprioceptive sensors implemented on each joint. The agent also has a collision detective sensor that perceives a collision information between robot arms and obstacles. This collision detective sensor can judge only information that indicates 'colliding' or 'not colliding'. The agent can receive no information about contacting positions on arms, nor information about distances between obstacles and the manipulator. Therefore, existing obstacles in the environment cannot be explicitly modeled in the planning agent. Since the agent initially has no information about the goal position, conventional planning methods cannot be applied. As reinforcement signals, the environment gives only distance information between the goal position and the end-effector of the manipulator. That, but, does not indicate direction information.

On the basis of this problem description, the agent executes one motion at every unit time. The agent makes motion plans from given initial state to an objective state in which the end-effector reaches the goal position. A solution of motion plans is represented as a sequence of angle displacement vectors, $\Delta q(t)$, from an initial state to an objective state. One planning cycle called a trial terminates when one of the following conditions is satisfied.

1. The end-effector reaches the goal position.
2. The manipulator collides with any obstacle.
3. The number of executed motion times in a trial reaches a predetermined upper limited value.

3.3.5.2 Implementation of Simple Classifier Systems

Simple classifier systems are implemented to realize a reactive planning system in the planning agent. In this case, an environmental information, E_S, corresponds to a joint angle vector, $\boldsymbol{q}$. A collision information is not used in the performance system, because a trial terminates when a colliding information is obtained, and next trial starts with the initial state. Through the detector, hence, a joint angle vector, $\boldsymbol{q}$, is encoded into an internal message *EM*, and then a conditional matching process is carried out to extract matched classifiers. An angle displacement vector, $\Delta\boldsymbol{q}$, is indicated by decoding the action part of selected classifier. As a result of these processes, the state of the manipulator is changed, and a new environmental information is observed.

The encoding process and the decoding process are accomplished based on the defined resolutions of sensors and controllers in section 3.3.4. That is, a joint variable, q_i, is encoded on a ${n_s}^i$ bit-lengthen sub-string. Similarly, an angle displacement value, Δq_i, is acquired by decoding a ${n_a}^i$ bit-lengthen sub-string. In this experiment, a gray coding method is applied rather than a usual binary coding method. Because in a gray coding method, a Hamming distance among neighboring regions is to be 1. For example, if ${n_s}^i = {n_a}^i = 2$, then an encoding and a decoding are carried out as shown in figure 3.5.

As a reinforcement strategy in the credit assignment system, a kind model of profit sharing method is applied to learn given tasks. In this reinforcement strategy, strength values of classifiers are revised by given environmental reward signals that are determined according to the moved result of the manipulator at every time step. Let $\Delta\boldsymbol{q}(t)$ be a displacement vector at time t, $\boldsymbol{q}(t)$ be a new state of manipulator observed as a result of a motion $\Delta\boldsymbol{q}(t)$, $EE(t)$ be a position of the end-effector in a state $\boldsymbol{q}(t)$, and p be a goal position. Then an environmental reward signal r^t is given according to the following conditions. That is, a strength value of the classifier that indicates a motion $\Delta\boldsymbol{q}(t)$, is changed by adding the αr^t.

1) *if* $d(p, EE(t)) < d(p, EE(t\text{-}1))$

then r^t = fixed small reward value (+10)

2) *if* $d(p, EE(t)) < \varepsilon$

then r^t = fixed large reward value (+30)

3) *if* $CC(\boldsymbol{q}(t), \boldsymbol{OB}) = 1$

then r^t = fixed large penalty value (-30)

where, d is a distance function among two points, ε is a small value to judge the accomplishment of a goal state, CC is a collision detective function that shows a value '1' when the manipulator collides with any obstacles in an existing obstacle set $\boldsymbol{OB}$, otherwise, the function produces a value '0'.

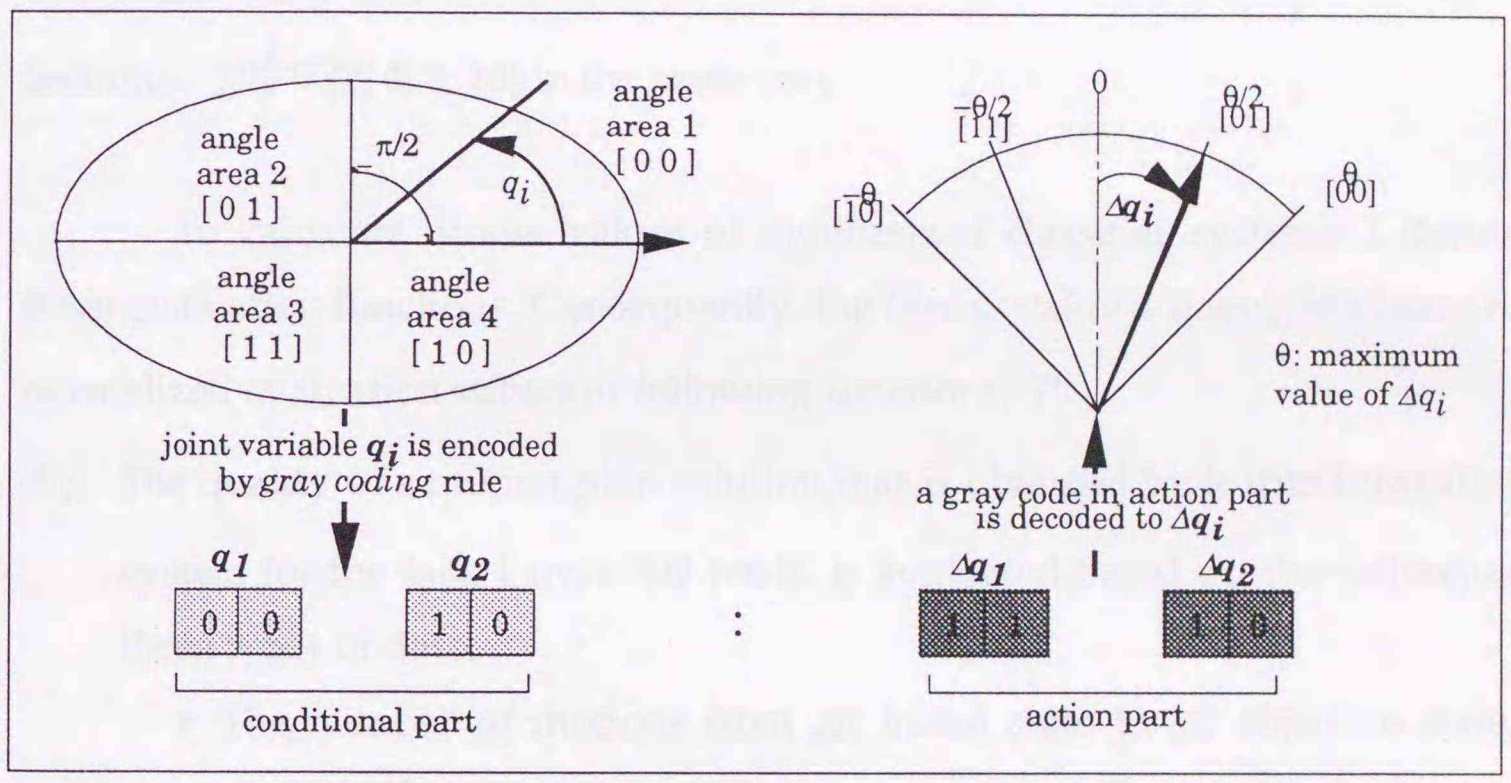


Figure 3.5 An example of encoding and decoding method.

$(n_s{}^i = n_a{}^i = 2)$

3.3.5.3 Applying Genetic Algorithms to the Architecture Synthesis Process

In this experiment, a designer prepares some candidate parameter values for each parameter in the defined parameter set P expressed by equation (3-3-9). Each candidate parameter value is assigned to a binary sub-string to represent a chromosome. Table 3.3 shows prepared candidate values for each parameter and the length of encoded binary sub-string in a chromosome. In this table, the number of divided regions of each joint variable q_i is equivalently settled, because the sensible range of each proprioceptive sensor is equally fixed ($[0, 2\pi]$). Since resolution of sensors, $n_s{}^i$, is assigned to one of prepared values $\{1, 2, 3, 4\}$, the number of divided regions of q_i becomes $2^{n_s^j} \in \{2, 4, 8, 16\}$. The number of divided regions of an angle displacement value, Δq_i, also becomes $2^{n_a^j} \in \{2, 4, 8, 16\}$ in the same way.

To calculate fitness values of synthesized classifier systems, I define three evaluation functions. Consequently, the fitness value is a weighted sum of normalized evaluation values of following functions, Ψ_j.

Ψ_1: The quality of a motion plan solution that is obtained by learned classifier system for the task 1 over 300 trials, is evaluated based on the following three types of data.

- The number of motions from an initial state to an objective state, namely, N_m.
- The proportion of appropriate motions by which the end-effector is getting closer to the goal position, namely, R_a. That is, by acting an appropriate motion, the system receives any positive reward value from the environment. R_a presents the local performance of a motion plan.
- The proportion of goal attaining times, namely, R_r.

Ψ_2: The required trials to converge the solution for the task 1.

Ψ_3: The motion times of a gotten solution for a task 2 over 20 trials, by using learned classifier system for a task 1 over 300 trials.

In the evolutionary synthesis process, simple genetic algorithms are applied to search an appropriate combination of parameters. Table 3.4 indicates GA-parameters used in the evolutionary synthesis process.

Table 3.3 Prepared candidate values for each parameter and the length of encoded binary sub-string in a chromosome.

Represented parameters in a chromosome	The length of binary sub-string	Assigned candidate values
The number of classifiers: N	2 bits	{500, 1000, 2000, 3500}
Generating probability of symbol '#': $P_{\#}$	1 bit	{0.1, 0.2}
Initial strength of classifiers: S_0	1 bit	{500, 1000}
Resolutions of proprioceptive sensors of each joint: $n_S = n_S{}^1 = n_S{}^2$	2 bits	{1, 2, 3, 4}
Resolutions of motion controllers of joints: $n_a = n_a{}^1 = n_a{}^2$	2 bits	{1, 2, 3, 4}
Discount coefficient in profit sharing method: α	1 bit	{0.1, 0.3}
Starting probability of rule discovery procedure: ρ	1 bit	{0.05, 0.1}
Crossover rate in a rule discovery procedure: χ	1 bit	{0.002, 0.005}
Mutation rate in a rule discovery procedure: μ	1 bit	{0.002, 0.005}

Table 3.4 GA-parameters used in the evolutionary synthesis process.

The length of a chromosome	12 bits
The number of chromosome in a population	10
Reproduction strategy	Elitist strategy
Crossover rate	0.5
Mutation rate	0.3

3.3.5.4 Results and Analysis

Figure 3.6 shows results of evolutionary architecture synthesis experiments by genetic algorithms. In this figure, the axis of abscissa indicates the genetic generation, and the axis of ordinate represents performances of synthesized classifier systems evaluated by defined fitness functions. The best, average and worst performance values in each population are plotted. These results indicate that chromosomes having higher performances are generated by this evolutionary search mechanism. Figure 3.7 illustrates resultant motion plans for task 1 and task 2. The motion plan of task 2 is shown by the learned classifier system for task 1 over 300 trials. These motion plans are generated by the best simple classifier system architecture, i.e., (N=3500, n_s=3, n_a=3, $P_\#$=0.2, α=0.1, S_0=1000, ρ=0.05, χ=0.002, μ=0.002), that is synthesized through the evolutionary process over 30 generations.

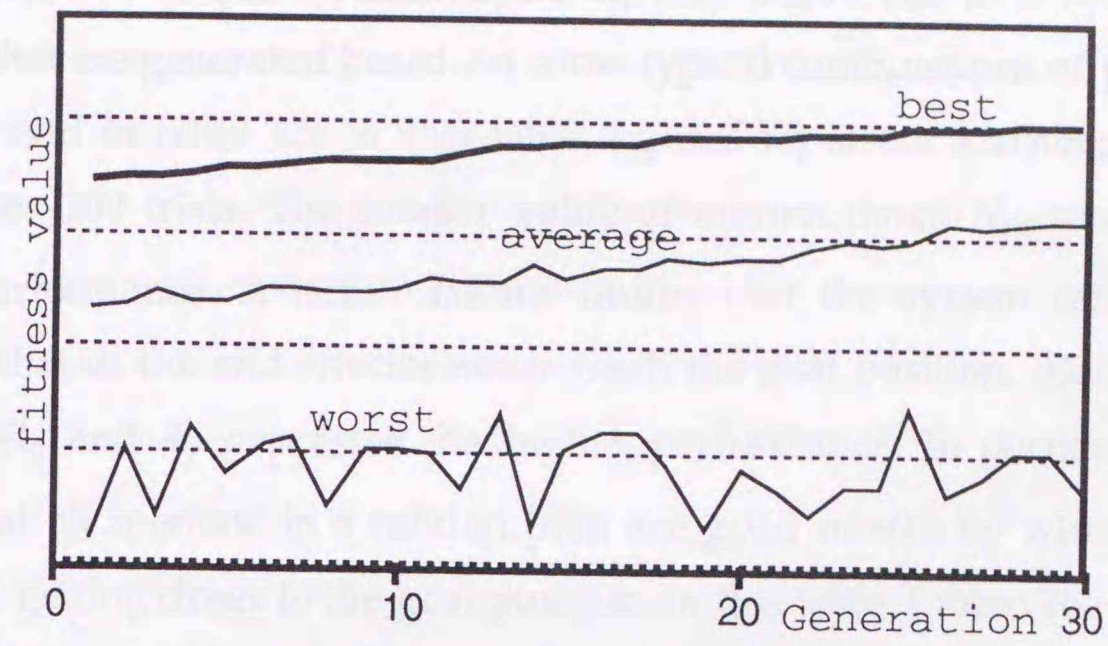


Figure 3.6 Results of evolutionary architecture synthesis experiments by genetic algorithms.

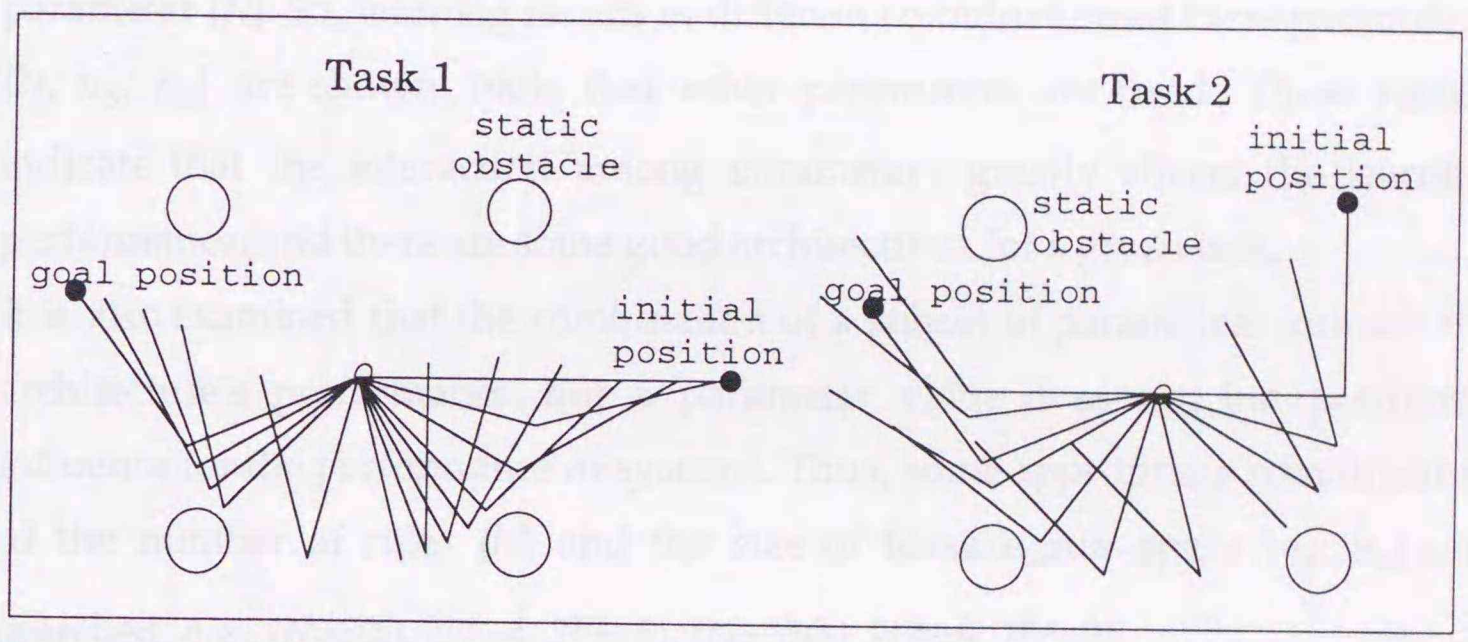


Figure 3.7 Resultant motion plans for given experimental tasks. The motion plan of task 2 is shown by the learned classifier system for task 1 over 300 trials. These motion plans are generated by the best simple classifier system architecture, i.e., (N=3500, n_s=3, n_a=3, $P\#$=0.2, α=0.1, S_0=1000, ρ=0.05, χ=0.002, μ=0.002)

To examine the influences of parameter values for learning performances of simple classifier systems, learning results of simple classifier systems that are generated based on some typical combinations of parameters, are expressed in table 3.5. In this table, figures represent learning results for task 1 over 300 trials. The smaller value of motion times N_m represents the higher performance. A term ' failure' means that the system cannot find a solution, that is, the end-effector never reach the goal position. Also, the larger value of R_a and R_r expresses the higher performance. In particular, R_a=1.0 means that all motions in a motion plan are good moves by which the end-effector is getting closer to the goal position. In this table, I focus the interaction between the size of potential rule space and the number of rules contained in the rule-base. The size of potential rule space depends on parameters about resolutions, that is, $\{n_s, n_a\}$, and the number of rules is determined by a parameter $\{N\}$. So, learning results of different combination of these parameters $\{N, n_s, n_a\}$ are shown. Note that other parameters are fixed. These results indicate that the interaction among parameters greatly affects the learning performance, and there are some good architectures for a given task.

It is also examined that the combination of a subset of parameters induces the architecture's performance, but a parameter value does not independently influence for the performance of systems. Thus, some appropriate combinations of the number of rules $\{N\}$ and the size of feasible rule space $\{n_s, n_a\}$, are searched, e.g., (N=500, 2^{ns}=4, 2na=8), (N=3500, 2^{ns}=8, 2na=8).

Learning behaviors of architectures made by six typical combinations in table 3.5 are shown in figure 3.8. In this figure, each parameter combination is represented by simplified notations, i.e., ($N/2^{ns}/2^{na}$). The vertical axis represents total motion times in a trial, and the horizontal axis indicates trials. I assume that upper limit of motion times, N_m, is fixed at 200. So, when a solution cannot be found, N_m is plotted on 200. This result indicates, there are some parameter combinations, e.g., (1000/8/8), that have higher convergency, but by which the best solution cannot be searched. Oppositely, there are some combinations, e.g., (500/4/8), (3500/8/8), or (3500/16/16), by which more

better solutions are searched through long learning phases.

It might be evident that induced motion plans are quickly converged in the system having relatively small rule size to the hole of potential search space. Because, rule conflicts among matched classifiers very rarely occur in such a system, and hence reinforcement signals are concentrated on primarily found solutions. Therefore, it is shown that appropriate frequencies of rule conflicts are required to do searching activity well. There also are bad parameter combinations, e.g., (3500/4/8) or (500/16/16). Classifier systems made from these parameter combinations, might have too small or too rich rule size for the size of potential search space.

Table 3.5 Learning results of some typical combinations of parameters.
(fixed other parameters:
$P_{\#}$=0.2, α=0.1, S_0=1000, ρ=0.05, χ=0.002, μ=0.002).

		Resolution of sensors and motion controllers					
		2^{ns}=4		2^{ns}=8		2^{ns}=16	
		2na=8	2na=16	2na=8	2na=16	2na=8	2na=16
N=500	N_m	12	failure	failure	failure	failure	failure
	R_a	1.0	0.89	0.76	0.71	0.64	0.60
	R_r	1.0	0.0	0.5	0.0	0.0	0.0
N=1000	N_m	19	14	34	28	34	43
	R_a	0.84	1.0	0.76	1.0	0.85	0.78
	R_r	1.0	1.0	1.0	1.0	0.6	1.0
N=2000	N_m	failure	failure	15	31	57	32
	R_a	0.67	0.93	0.87	0.85	0.70	0.84
	R_r	0.0	0.0	1.0	1.0	1.0	1.0
N=3500	N_m	failure	14	12	23	19	22
	R_a	0.67	1.0	1.0	0.80	0.89	0.86
	R_r	0.0	1.0	1.0	0.8	1.0	1.0

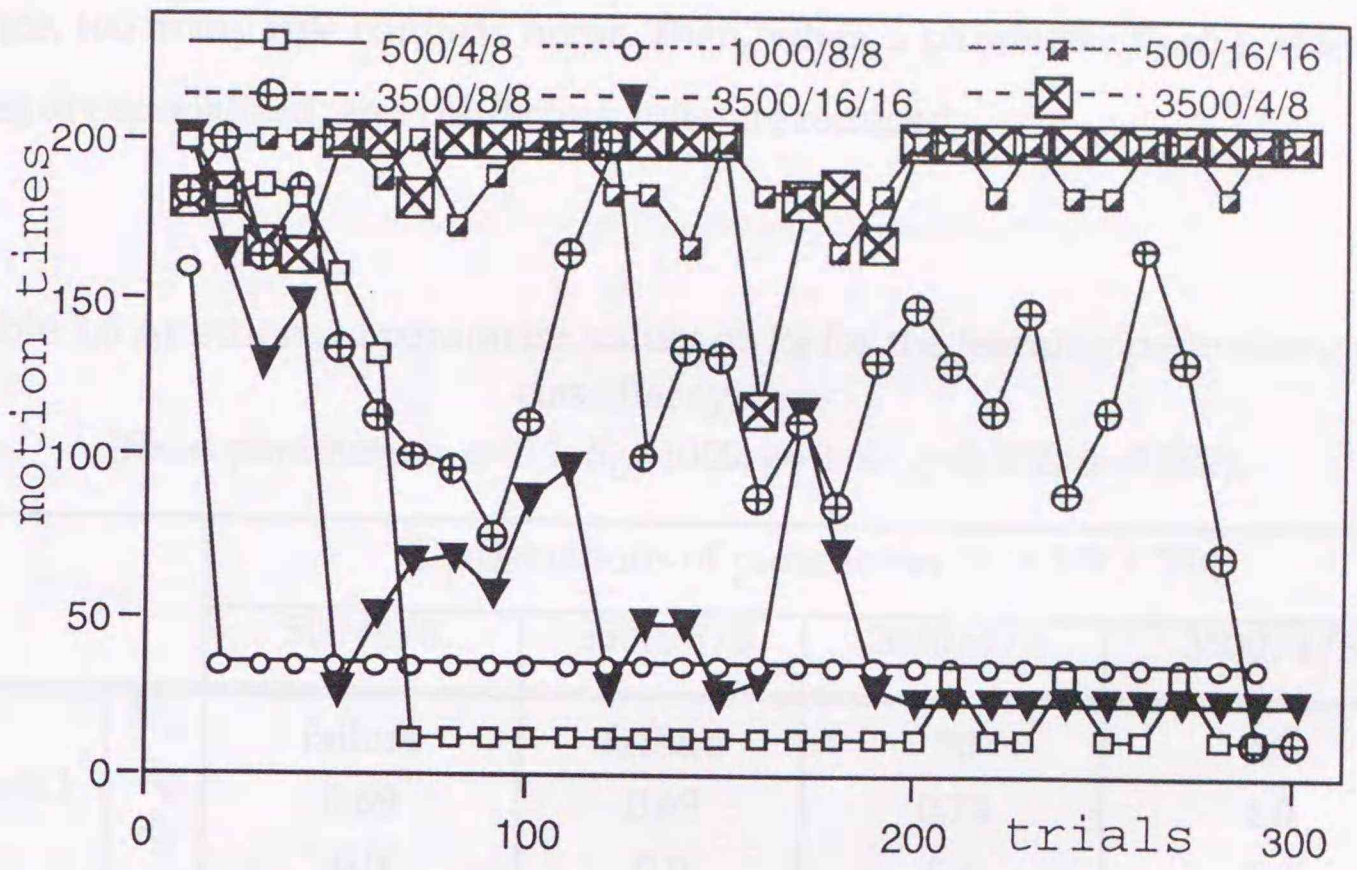


Figure 3.8 Learning behaviors of classifier system architectures made from six typical combinations. Each parameter combination is represented by simplified notations, i.e., ($N/2^{ns}/2^{na}$).

(fixed parameters: $P_{\#}$=0.2, α=0.1, S_0=1000, ρ=0.05, χ=0.002, μ=0.002)

Table 3.6 shows affections of parameter $P_{\#}$ for the learning performance of classifier systems. These results also indicate that the learning performance is influenced by interactions between a parameter $P_{\#}$ and other parameters. That is, parameter combinations ($P_{\#}$=0.2, N =500, 2^{ns}=4, 2na=8) and ($P_{\#}$=0.2, N =3500, 2^{ns}=8, 2na=8), make very good architectures of classifier systems. But, by changing the value of parameter $P_{\#}$ from 0.2 to 0.1 in these combinations, the systems become to find no solution. One reason for these results might be that useful generalized rules including symbol '#' are lost, and hence, there is no matched rule for certain problem states. The other hand, in this table there also be the system cannot search a solution well when a parameter $P_{\#}$=0.2, e.g., (2000/4/8) and (3500/4/8). For these systems, by altering the $P_{\#}$ to 0.1, good performances are obtained. I consider a reason of this as follows. When a parameter $P_{\#}$=0.2, there might be over generalized rules,

hence, too many rule conflicts occur. Then, when a parameter $P_{\#}$=0.1, classifier rules are specialized, and conflicting rules are reduced.

Table 3.6 Affections of parameter values of $P_{\#}$ for the learning performance of classifier systems.
(fixed parameters: α=0.1, S_0=1000, ρ=0.05, χ=0.002, μ=0.002).

		Combinations of parameters (N / 2^{n_s} / 2^{n_a})			
		500/4/8	3500/8/8	2000/4/8	3500/4/8
$P_{\#}$=0.1	N_m	failure	failure	50	16
	R_a	0.69	0.69	0.73	1.0
	R_r	0.0	0.0	0.4	1.0
$P_{\#}$=0.2	N_m	12	12	failure	failure
	R_a	1.0	1.0	0.67	0.67
	R_r	1.0	1.0	0.0	0.0

Experimental results about convergency of classifier system architectures are shown in table 3.7. In this table, the number of needed trials to converge, Ψ_2, is expressed. As this result shows, the interrelationship between initial strength values and amount of reinforcement rewards affects the convergency of systems. The relatively larger reward value for the initial strength value makes the faster convergence.

Table 3.7 Experimental results about convergency of classifier system architectures.
(fixed parameters: $P_{\#}$=0.2, ρ=0.05, χ=0.002, μ=0.002).

		Combinations of parameters ($N/2^{n_s}/2^{n_a}$)	
		3500/8/8	3500/16/16
S_0=1000, α=0.1	N_m	12	22
	Ψ_2	274	187
S_0=500, α=0.3	N_m	19	32
	Ψ_2	76	23

In these experiments, obvious influence of rule discovery parameters, i.e., { ρ, χ, μ }, is not recognized. One reason of this might be that the predetermined value of trials, 300, is not enough to exhibit the ability of the rule discovery mechanism.

Figure 3.9 illustrates applicabilities of learned classifier systems for a similar task. These results indicate the robustness of each system architecture. In this figure, learning results of five typical architectures made from successful parameter combinations in table 3.5. As this figure shows, all of successfully adapted system architectures for a task 1 does not always do well for a task 2. The successful system architectures for two tasks might learn the common problem concept of two tasks.

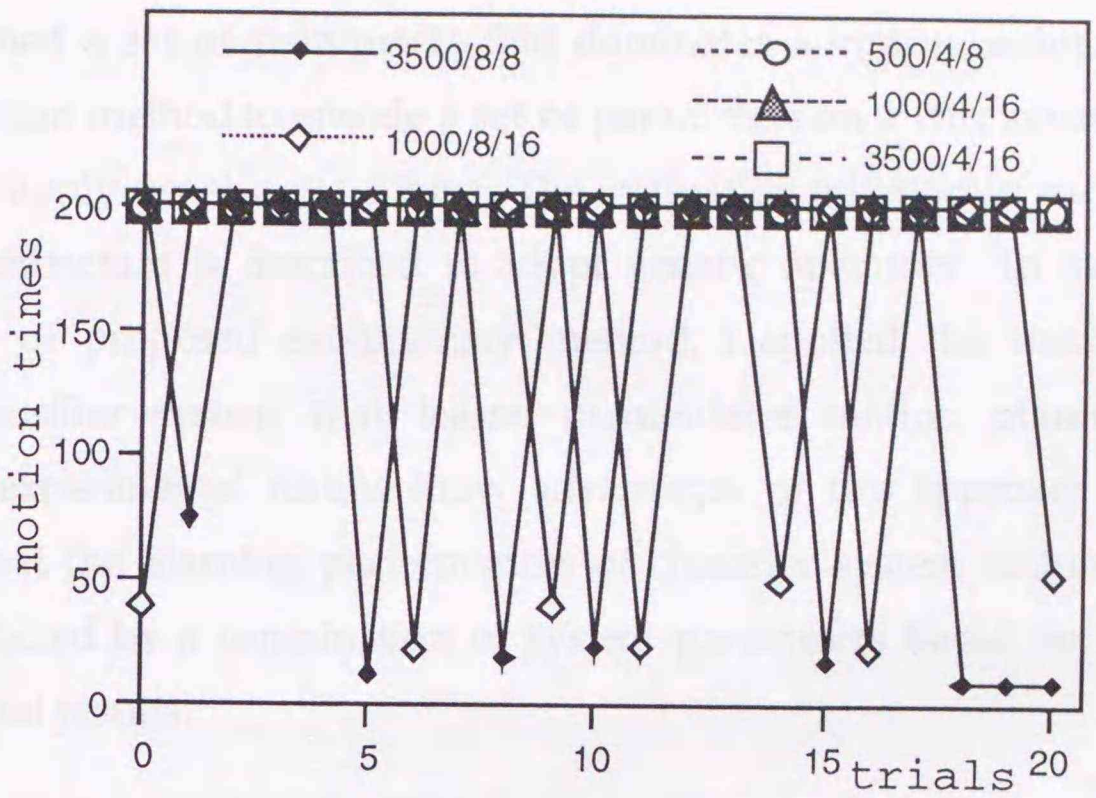


Figure 3.9 Applicabilities of learned classifier systems for a similar task.
(fixed parameters: $P_{\#}$=0.2, α=0.1, S_0=1000, ρ=0.05, χ=0.002, μ=0.002).

On the basis of above experimental results, it is examined that the learning performances of classifier system architectures are largely affected by a combination of system parameters, and it is difficult to find an appropriate system architecture from lots of feasible architectures by a human designer through trial-and-error processes. It is also clear that the system performance is greatly affected by interactions among parameters. It is hard to apply the conventional optimization methods such as hill-climbing algorithm. Therefore, this approach adopting genetic algorithms that are evolutionary multi-point search method, is very advantageous.

3.3.6 Concluding Remarks

In this approach, I proposed the evolutionary architecture synthesis method of simple classifier systems to generate an appropriate architecture for given tasks. To apply this method, I formalized the simple classifier systems,

and I defined a set of parameters that dominates a system architecture. The representation method to encode a set of parameters on a chromosome, is also defined to apply genetic algorithms. The evaluation criterion of each classifier system architecture is described to adopt genetic operators. To examine the usefulness of proposed evolutionary method, I applied this method to the simple classifier system that learns manipulator motion planning tasks. Obtained experimental results show advantages of this approach. Finally, I clarified that the learning performances of classifier system architectures are largely affected by a combination of system parameters based on the gotten experimental results.

3.4 Extensive Rule Representation in Classifier Systems by Masking Classifiers

3.4.1 Introduction

The classifier system is a machine learning system that learns production rules which induce appropriate actions for environmental conditions. As we already know, the original classifier system proposed by Holland is a simple representational and computational paradigm that utilizes genetic algorithms. A number of implementations of classifier systems have been built demonstrating the potential of this paradigm for machine learning. Advantages of using classifier systems are parallelism, convenient conditional matching and effective exploration in the search space. The simple representation of classifier rules, i.e., a conventional binary encoding method, produces these advantages. At a first glance, classifier representations appear to be a simple, effective method for implementing computational systems that are well suited to manipulation by genetic algorithms. However, upon implementing even a simple classifier-based learning system, many subtleties arise within the representation which can have a strong influence on the learning capabilities of the system [Shuurmans and Schaeffer 89]. Especially in section 3.4, I focus on the representational difficulty including encoding methods. That is, I propose an extensive rule representational method by which the learning can be achieved by relatively smaller size of internal classifier rules than the whole size of rule space. To realize this representational scheme, I define masking classifiers that play a role of filtering schemata for general classifiers. To examine representational performance of proposed represents, classifier systems with masking classifiers are applied to block stacking problems.

3.4.2 Representational Difficulties

Behind the simplicity of the classifier representation, there are a number of subtleties arising in the representation of domain knowledge that greatly affect the rate and capability of learning confirmed in classifier systems. The following problems have been pointed out in the classifier representation.

- ***Appearance of illegal string patterns***: We have necessarily encoded certain classifier patterns as being illegal. In the simplest case, classifiers are represented as fixed length bit strings. Consider an information field that can take on any one of n values. In a conventional classifier representation, we need the binary field having $k=\lceil \log_2 n \rceil$ bits to encode all of n values. A notation $\lceil \log_2 n \rceil$ means the least integer value in an integer set in which every integer value is greater than $\log_2 n$ or equal to $\log_2 n$. Using a k- bit length string, 2^k possible bit patterns can be produced. Generally, 2^k is greater than n. Therefore, $(2^k - n)$ bit patterns are not assigned any real value. These are called illegal patterns. For example, when a field can take on one of three values, 2-bit strings are necessary. In this case, there are four varieties of bit patterns, so one bit pattern dose not take on any value. Even if a designer creates an initial set of classifiers without any illegal pattern, the rule discovery mechanism based on the simple genetic algorithms would generate illegal strings. To resolve this illegal pattern problem, every pattern can be assigned to any value. However, a new problem might be happened by this solving technique, that is, the search bias problem.

- ***Search bias problems***: Shuurmans and Schaeffer pointed the search bias problem [Shuurmans and Schaeffer 89]. This problem arises when the probabilities of occurrence of field values are not equal. In the above example case, i.e., n=3, if I assign the pattern 00 to a first value, the pattern 01 to a second value, and the patterns 10 and 11 to a third value, then any illegal pattern is not produced. In classifier systems, however, the searching probability of third value is double of probabilities of other values. So, there are differences of reinforcement opportunities among three values. The

search bias problem might influence the learning performance of the system.

- ***Bit assignment problem***: The presence of the "don't care" symbol, #, in classifier representations is important, because it allows to generalize classifiers by expressing schemata in the conditional parts of classifiers. This conditional schema allows conditions to match any one of a set of possible values, that is, reducing the number of classifiers required to implement desired behaviors. To realize this reduction, effective bit pattern assignments are required. For example, consider a conditional information that can take on any one of three values, {X, Y, Z}, and an action field that takes on either of two values, {ON, OFF}. In this assumption, a classifier consists of the 2-bit long conditional part and the 1-bit length action part. As a bit pattern assignment, I fix an encoding method by which X=00, Y=10, and Z=11 in conditional strings, and in action parts ON=1 and OFF=0. So, a classifier 00/0 means that if a conditional value is an 'X', then achieve an action 'OFF'. Here, we simply want to express that whenever an input value is either an 'X' or an 'Z', but it does not contain an 'Y', then the action 'ON' should be responded. In this representation, however, it is impossible to express this schema, (X or Z, not Y), using '#' symbol in a classifier. Only conditional pattern ## can match an 'X' and an 'Z', but this pattern necessarily matches an 'Y' as well. Consequently, the solution to this sub-problem requires two distinct classifiers, that is, 00/1 and 11/1. Next, let us change the representation method. As another bit pattern assignment, I alter the assignment of a value 'Z' from 11 to 01. Then I can express the schema, (X or Z, not Y), by the pattern 0#, and yield a single classifier that solves the problem, that is, 0#/1. Obviously, by changing the bit pattern assignment implemented on the problem solving, the size of the solution set that the system must discover might be heavily altered.

- ***Grammatical representation:*** One of the main features of the classifier representation is very simplified encoding method that is separated from true representational structures. However, for a particular problem class, more complex representation such as natural language coding might

improve the learning performances of classifier systems. Furthermore, the problem of positional semantics is pointed out in [Shuurmans and Schaeffer 89]. That is, in simple classifier representation for many problems, we can see that the semantics of the representation are position dependent. However in the basic classifier systems/genetic algorithms framework, there is no effective mechanism by which important information can be exchanged between classifier positions. There is no way that important generalizations can be made across positions in a classifier string. If an important pattern has been discovered for one particular position in a classifier string, this same pattern will have to be independently discovered for each separate position where that pattern may be important.

- ***Combinatorial explosion of search space***: Goldberg said that classifier systems are computationally universal [Goldberg 89]. Given this, one cannot say that there are solutions which exist for some problems that a classifier system cannot express. However, we must keep in mind that solutions are automatically discovered with some form of mechanized search over the space of possible rules. By utilizing simplified representations we do not inherently exclude any problem solutions from being expressible, but we do expose ourselves to the possibility of a combinatorial explosion in the number of rules required to express these solutions. Such an explosion in the solution size necessarily manifests itself as an explosion in the search effort required to discover that entire solution. At present, there exists no detailed analysis from the machine learning perspective of this trade-off between solution set size and search space size. A better understanding of this issue would contribute a great deal towards constructing effective, domain independent learning machines.

Some extensive encoding methods have been proposed to overcome representational difficulties. For example, Grefenstette described a flexible and natural language expression method for rule representations in his genetic learning system named SAMUEL [Grefenstette 91]. Shu and Schaeffer proposed the variable classifier system to embedded variables into conditional parts of classifiers. Those variables are used to describe abstract relations in a succinct

manner, reducing the size of the solution set for many problems [Shu and Schaeffer 89]. Also some proposed extensive representation for genetic algorithms might be usefully applied to the classifier representation.

I would focus on the problem of the combinatorial explosion of search space. To overcome this representational difficulty inherent in classifier systems, the masking classifier is defined in the next section.

3.4.3 Definitions of Masking Classifiers

As we know, when a problem has the large size of potential state space, it is difficult to prepare all feasible rules in an initial set of classifiers. As a result of this, there might be situations where the appropriate solution or a solution cannot be induced by only stored classifiers.

Here, let us see again the classifier representation in simple classifier systems. A set of classifiers, *CF*, is represented as follows;

$$CF=\{cf_i, ; i = 1,2,...,N\}, \tag{3-4-1}$$

$$cf_i =(cf^c{}_i , cf^a{}_i , Scf_i), \tag{3-4-2}$$

$$cf^c{}_i \in \{0, 1, \#\}^{lc}, \tag{3-4-3}$$

$$cf^a{}_i \in \{0, 1\}^{la}, \tag{3-4-4}$$

$$Scf_i \in \textbf{Reals} \tag{3-4-5}$$

where, N is the rule size, a classifier cf_i is a string rule that consists of a condition part $cf^c{}_i$, an action part $cf^a{}_i$ and a strength value Scf_i. In a conditional part $cf^c{}_i$, the character '#' is used as a *'don't care'* symbol. In simple classifier systems, the conditional matching between the conditional part of each classifier and an encoded problem state, is performed to decide an appropriate action for the current state. Therefore, it is necessary that a set of memorized classifiers in the system covers whole of potential rule space. Moreover, classifier rules should be generated by a common encoding protocol by which

the environmental state is encoded into an internal string message. The encoding protocol is important and should be carefully determined, because all environmental information in a real current problem state cannot be represented in a finite length of string. Even if a problem state is encoded into a finite length of string by a certain encoding method, it is exhausting that a set of memorized classifiers in the system covers whole of external problem space. Generally, so, a designer decides the domain-dependent encoding protocol by which only required environmental information to solve the problem is extracted and encoded. By this domain-dependent encoding method, a huge size of state space is shrunken into a suitable size of rule space that is easy to deal with. However, it is very difficult to determine a good encoding protocol, because that greatly depend upon the problem domain. Moreover, unsuitable encoding methods enlarge the search space and bring upon that the learning performance becomes worse.

3.4.3.1 Representations

So, to overcome above difficulties, I attempt to develop a new representational scheme of classifiers, that is named the masking classifiers. By this representational scheme, the profitable schema of environmental information is automatically extracted and reinforced. To realize the representational scheme, another rule set is defined in the rule-base system. That is a set of masking classifiers, *MCF*. A masking classifier set, *MCF*, is represented as,

$$MCF=\{mcf_j\ ;\ j=1,2,...,M\}, \tag{3-4-6}$$

$$mcf_j \in \{*, \#\}^{lc}, \tag{3-4-7}$$

where, M is the number of masking classifiers, and a masking classifier mcf_j consists of symbol '*' and '#'. In this notation, a symbol '*' is the "pass through" character, and a symbol '#' is the "don't care" character. The string length of a masking classifier is equal to the length of a conditional part of a classifier, *lc*. This masking classifier is not directly compared with the encoded environmental message.

3.4.3.2 The Masking Operation

In the system, a reactive rule to be compared is generated by combining the original classifier, cf_i, and the masking classifier, mcf_j. Let RR be a set of reactive rules, and rcf be a rule combining function. Then RR is represented as follows;

$$rcf : CF \times MCF \rightarrow RR \tag{3-4-8}$$

where,

$$RR=\{rr_{ij} ; i=1,..,N ; j=1,..,M\}, \tag{3-4-9}$$

$$rr_{ij}=(rr^{c}_{ij}, rr^{a}_{ij}). \tag{3-4-10}$$

A generated reactive rule, rr_{ij}, consists of a conditional part, rr^{c}_{ij}, and an action part, rr^{a}_{ij}, that is similar with original classifiers. An action part of a reactive rule, rr^{a}_{ij}, is quite equal to the action part of the classifier, thus, $rr^{a}_{ij} = cf^{a}_{i}$. The other hand, a conditional part is generated by the masking effects. Let $rr^{c}_{ij}{}^{k}$ be a value on an k-th position in the conditional string of the reactive rule, then each value is created by the following equation.

$$rr^{c^{k}_{ij}} = \begin{cases} cf^{c^{k}}_{i}, & if\ mcf^{k}_{j} = * \\ \#, & otherwise \end{cases} \tag{3-4-11}$$

where, $cf^{c}_{i}{}^{k}$ and mcf^{k}_{j} are a value on an k-th position in the conditional part of the i-th classifier and a value on an k-th position of the j-th masking classifier respectively. For example, when a classifier cf_i=0101/00 and a masking classifier mcf_j=**## ,then the masked reactive rule becomes as rr_{ij}=01##/00. Figure 3.10 illustrates a schematic view of the masking operation. Similarly with simple classifier systems, after generating reactive rules by the masking operation, the conditional matching between an encoded environmental message and each reactive rule is carried out to extract a matched set of reactive rules. Then, by the rule competition among matched rules, a winner rule that indicates an action is selected.

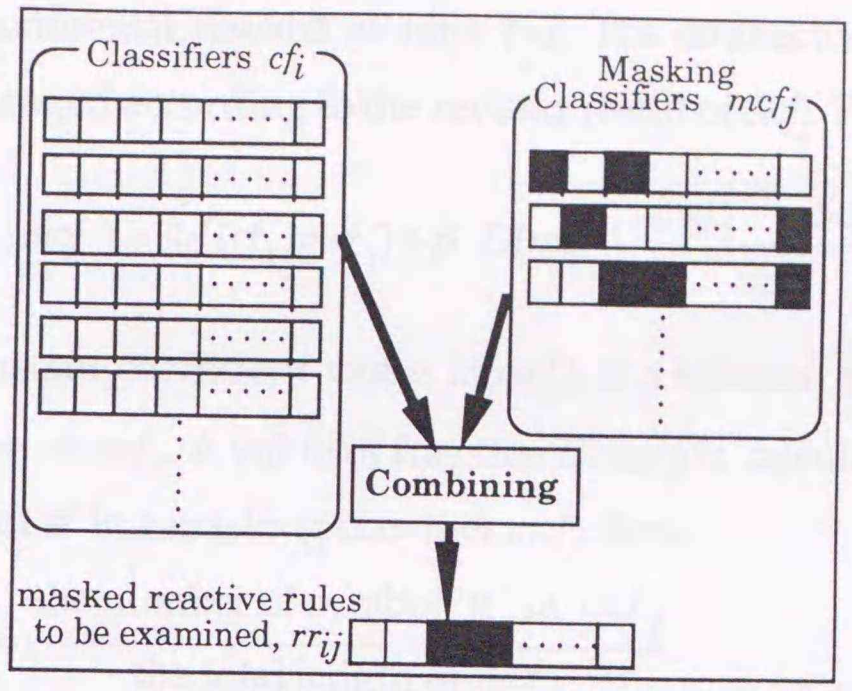


Figure 3.10 A schematic view of the masking operation.

3.4.3.3 The Conditional Matching and Biding Mechanisms

In simple classifier systems, biding values of matched classifiers are posted to resolve the rule conflict. The biding value of a classifier is calculated based on its strength value. Also in this masking classifier system, a biding value of a masked reactive rule, $B(rr_{ij})$, is calculated by the following equation.

$$B(rr_{ij}),= \alpha \cdot Scf_i \cdot Sc(cf_i, mcf_j) \tag{3-4-12}$$

where, α is a constant coefficient value, Scf_i is a strength value of cf_i, and $Sc(cf_i, mcf_j)$ is a value named the connective strength value that indicates the effectiveness of combination of cf_i and mcf_j. Based on the calculated biding value, a masked reactive rule is selected.

3.4.3.4 Revision of Strength Values

Strength values of classifiers and connective strength values are adjusted by reinforcement learning mechanisms. The strength value of classifier, Scf_i, is revised by the bucket brigade algorithm. That is,

$$Scf_i^{t+1} = Scf_i^t - B(rr_{ij}) + R^{t+1}, \tag{3-4-13}$$

where Scf_i^t is a strength value of the classifier that makes the winner reactive rule rr_{ij} at time t, and R^{t+1} is a paid biding value from a succeeding reactive

rule or an environmental reward at time $t+1$. The connective strength value, $Sc(cf_i, mcf_j)$, is changed according to the revised result of Scf_i. That is,

$$Sc^{t+1}(cf_i, mcf_j) = Sc^t(cf_i, mcf_j) + \beta \cdot D(mcf_j) \cdot \frac{\{Scf_i^{t+1} - Scf_i^t\}}{Scf_i^t}, \qquad (3\text{-}4\text{-}14)$$

where, β is a constant coefficient value, $D(mcf_j)$ is a function that measures the masking influence of mcf_j. A value of function $D(mcf_j)$ is calculated based on the number of symbol '#' in a masking classifier mcf_j, thus,

$$D(mcf_j) = \frac{\text{the number of symbol '\#' in } mcf_j}{\text{the total length of } mcf_j}. \qquad (3\text{-}4\text{-}15)$$

3.4.3.5 The Discovery of Masking Classifiers

In the masking classifier systems, if new classifiers or masking classifiers must be discovered by genetic algorithms, genetic operators are applied based on strength values as fitness values. In particular, masking classifiers are manipulated according to the average value of connective strength values, $S(mcf_j)$, that is represented as follows;

$$S(mcf_j) = \frac{1}{N} \cdot \sum_{i=1}^{N} Sc(cf_i, mcf_j). \qquad (3\text{-}4\text{-}16)$$

3.4.4 Numerical Experiments

3.4.4.1 Block Stacking Problems

To examine the proposed approach, the masking classifier system is applied to the simple block stacking problem that is one of robot task planning problems. A block stacking problem treated in this study is usually employed in AI or robotics fields. Given an initial state and a goal state of blocks, a solution of the problem is to be given as an optimal sequence of operations by which the given goal state is achieved with minimum cost.

In this problem setting, we assume that there is a set of blocks on a table. All of the blocks have the same size and no weight. A block can be placed on the table or on another block, and there is no restriction to height of a stack

of blocks. However on the table, there are only a bounded number of slots into which blocks can be allocated. The number of slots is given. Therefore, this problem in the constrained blocks domain is described as follows.

$$B = \{ b_i ; i \in I \}, \tag{3-4-17}$$

$$P(b_i) = (s_i , l_i), \tag{3-4-18}$$

$$s_i \in S, \quad l_i \in L, \tag{3-4-19}$$

where B is a set of blocks, I is a set of indexes of blocks, and a placed position of a block i is represented as $P(b_i)$. S and L are the sets of numbers of horizontal locations named slots, and vertical locations named layers respectively. Based on this state description, each robot can move only one block by following two operations at a time unit. That is,

PickUp(i): Pick up the top block in slot i. However no block is handled whenever slot i is empty or indicated i is not contained in the set S.

PutDown(i): Put down the block that is currently being held into slot i. But the block is held in the robot's hand whenever $i \notin S$.

Therefore, the solution to effectively achieve a given task is represented as a sequence of robot operations. Moreover I assume that each operation costs a pre-determined value. Simply, PickUp(i) and PutDown(i) cost 1 identically. In this problem setting, the objective is to find the optimal operation sequence that minimizes total costs. Figure 3.11 shows given experimental tasks. As this figure presents, the number of given blocks is four, and the number of slots is three. Hence, the upper limit of a vertical location where a block can be placed in the every slot is four.

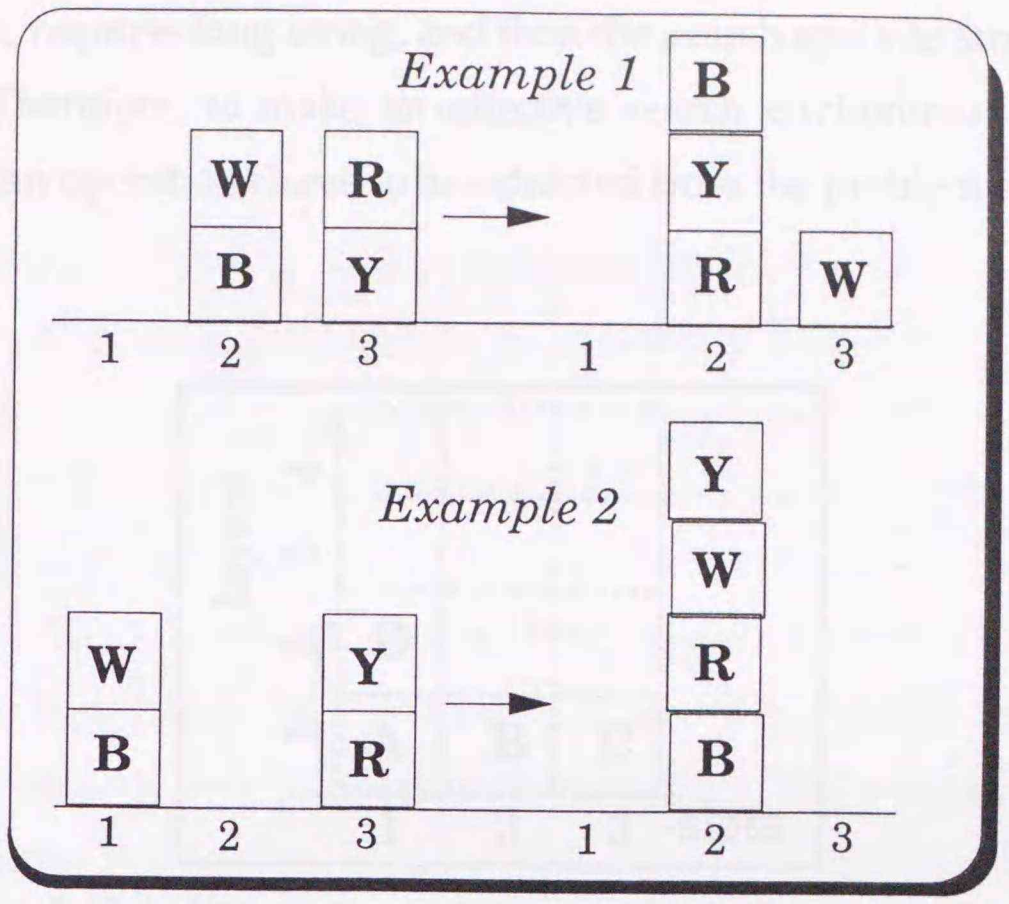


Figure 3.11 Given experimental tasks.

3.4.4.2 Implementation Methods

In the case where the masking classifier system is applied to this block stacking problem, a conditional part is compared with the current state of blocks. To perform the conditional matching, a current state is encoded into a string by an encoding rule. Here, the search performance of the problem is seriously affected by the applied encoding methods. In this experiment, a direct encoding method is adopted (figure 3.12). Thus, an encoding binary string consists of sub-strings. Each sub-string represents a corresponding allocable position where any block can be in. For given tasks, so, there are twelve allocable positions. Each sub-string has four bits, and when an *i*-th block is placed in an allocable position, the value of *i*-th bit in the corresponding sub-string is '1' and other bits take value '0'. If no block exists in an allocable position, then all bits take value '0'. Therefore, in every sub-string, one or none bit takes value '1' and all of others takes value '0'. As a result of this, a state of blocks is encoded into a 48-bit (=4*3*4) lengthen string. So, the length of a conditional part of a classifier and a masking classifier are equal to that of the encoded string. Even if these easy tasks, representing full information of a

problem state requires long string, and then the search space of strings becomes very large. Therefore, to make an effective search environment, only useful features to plan operations have to be extracted from the problem state.

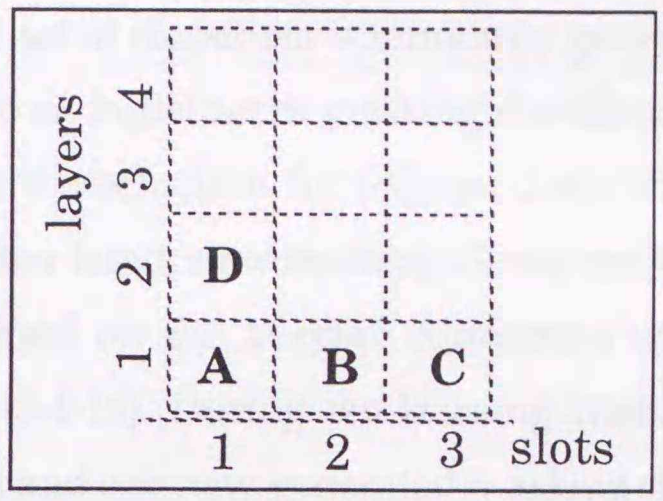


Figure 3.12 A direct representation method of a state of blocks.

Then, a winner reactive rule that is made by combining an original classifier and a masking classifier, is selected by the conditional matching and the rule competition defined above. According to the action part of a selected reactive rule, operations of robot at that time are instructed. An action part consists of two fields, thus, a_1 and a_2. Based on the instructed a_1 and a_2, $PickUp(a_1)$ and $PutDown(a_2)$ are carried out, then a state of blocks is transformed. These procedures mentioned above are repeatedly performed until attaining the goal state. Consequently, a sequence of operations is represented by a chain of executed reactive rules that transforms a problem state from an initial state to the goal state.

A system has to revise the strength values of classifiers based on each classifier's usefulness. Thus, this credit apportionment task involves evaluating which classifiers have played important role in determining an eventual success. In this experiment, this credit assignment mechanism is realized by the bucket brigade algorithm and the revision method defined in section 3.4.3. Also, the classifier that makes the last reactive rule in the sequence, receives an environmental reward because of achieving the goal state.

3.4.4.3 Experimental Results

A masking classifier system is applied to given tasks to examine the learning performance of proposed method. Table 3.8 represents experimental parameters. An initial set of classifiers is randomly generated such that there is no illegal pattern. Also an initial set of masking classifiers is created at random. The new rule discovery mechanism for original classifiers does not start up in learning trials. The other hand, new masking classifiers would be generated by genetic algorithms based on the average connective strength values, $S(mcf_j)$, defined by equation (3-4-16). During ten learning trials, one useless masking classifier is eliminated and one new generated masking classifier takes the place of that.

Figure 3.13 illustrates the learning curves of the masking classifier system for given experimental tasks. In this figure, the axis of abscissa is the learning trials, and the axis of ordinate represents total cost to attain the goal states. As this figure shows, the masking classifier system has searched solutions and has learned appropriate behaviors. This result indicates that the large potential rule space is effectively covered by combining a small number of classifiers and masking classifiers. The solutions of planning tasks converge at good solutions by a learning mechanism. That means that only one chain of stacking operations is made up, because the useful classifiers tend to have higher strength values than that of other classifiers by the reinforcement algorithms. Also, solutions of given tasks vibrate in the earlier period of learning. It seems that the system searches extensively in the problem domain to diminish the unknown spaces. Since the high learning performances are obtained, the strong learning ability of the masking classifier system is examined. Figure 3.14 represents strength values of classifiers, Scf_i, in the learned system for example 1 over 200 learning trials. This result expresses that higher strength values are assigned to some useful classifiers by reinforcement mechanisms in the masking classifier system. After the learning for example 1 over 200 learning trials, revised connective strength values are also shown in

figure 3.15. In this figure, plotted values are connective strength values between each masking classifier and the particular classifier, cf_{16}, that is verified as a good classifier in figure 3.14. As this result illustrates, it is clear that the useful mask is extracted for a classifier by the proposed strength revision mechanism.

Table 3.8 Experimental parameters.

the number of classifiers, N	100
the number of masking classifiers, M	20
initial strength value of classifiers, Scf_i	100.0
initial value of every connective strength, $Sc(cf_i, mcf_j)$	0.5
the reinforcement factor for the bucket brigade algorithm, α	0.2
the discount factor for the connective strength revision, β	0.1

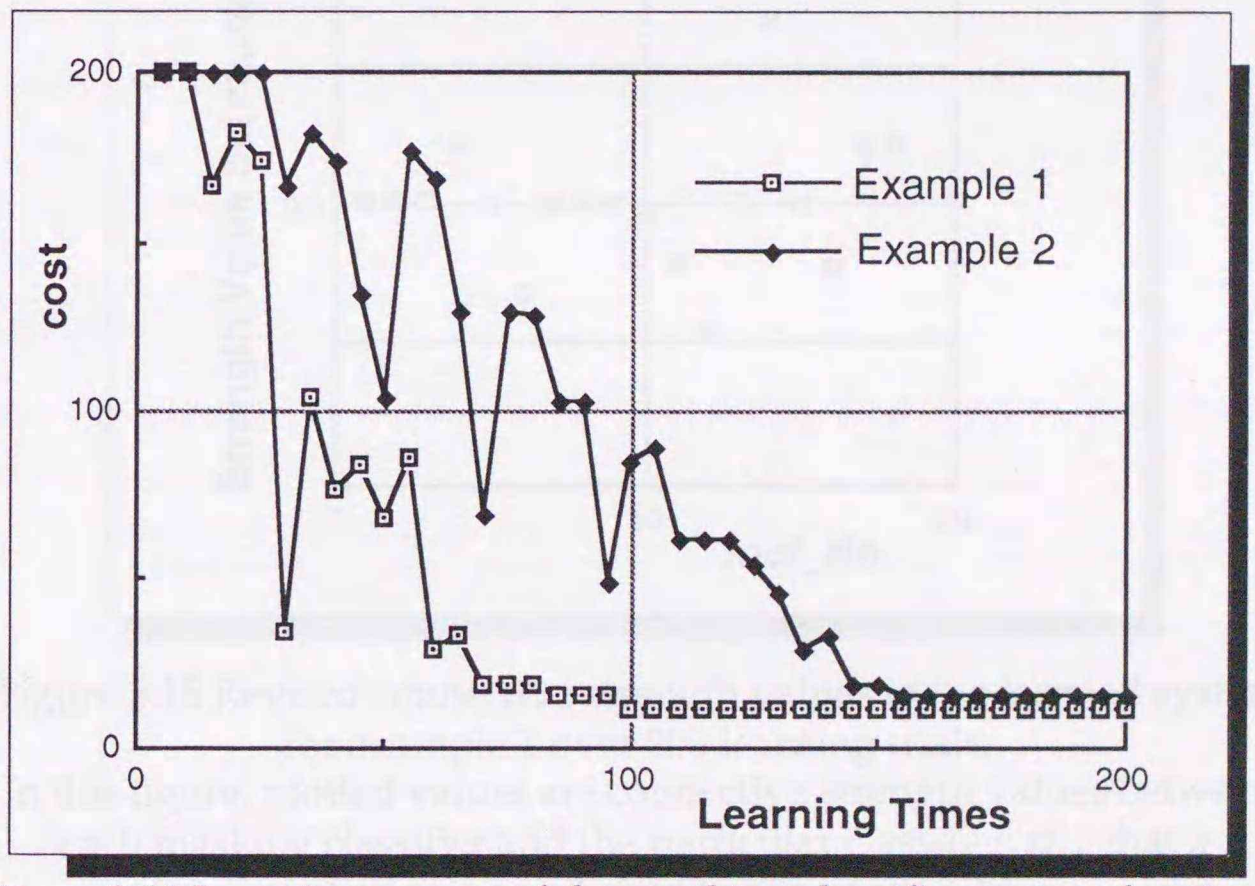


Figure 3.13 Learning curves of the masking classifier system for given experimental tasks.

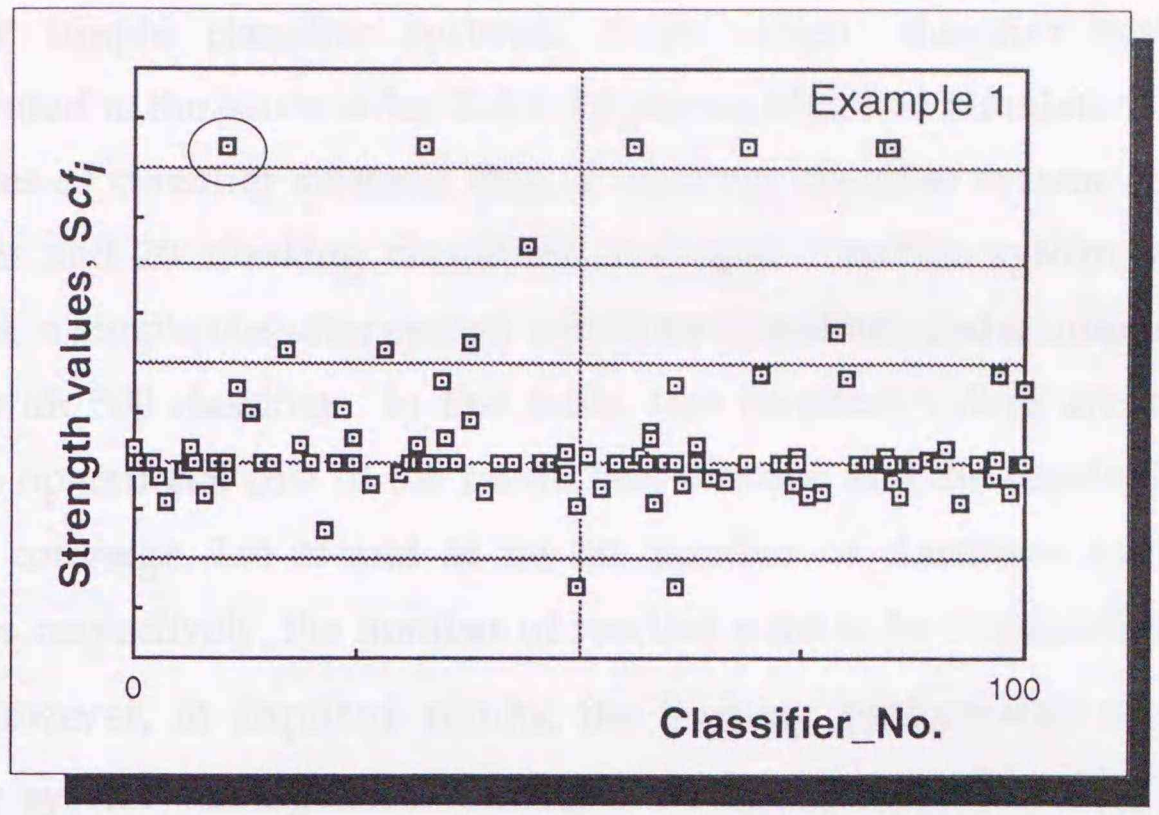


Figure 3.14 Strength values of classifiers, Scf_i, in the learned system for example 1 over 200 learning trials.

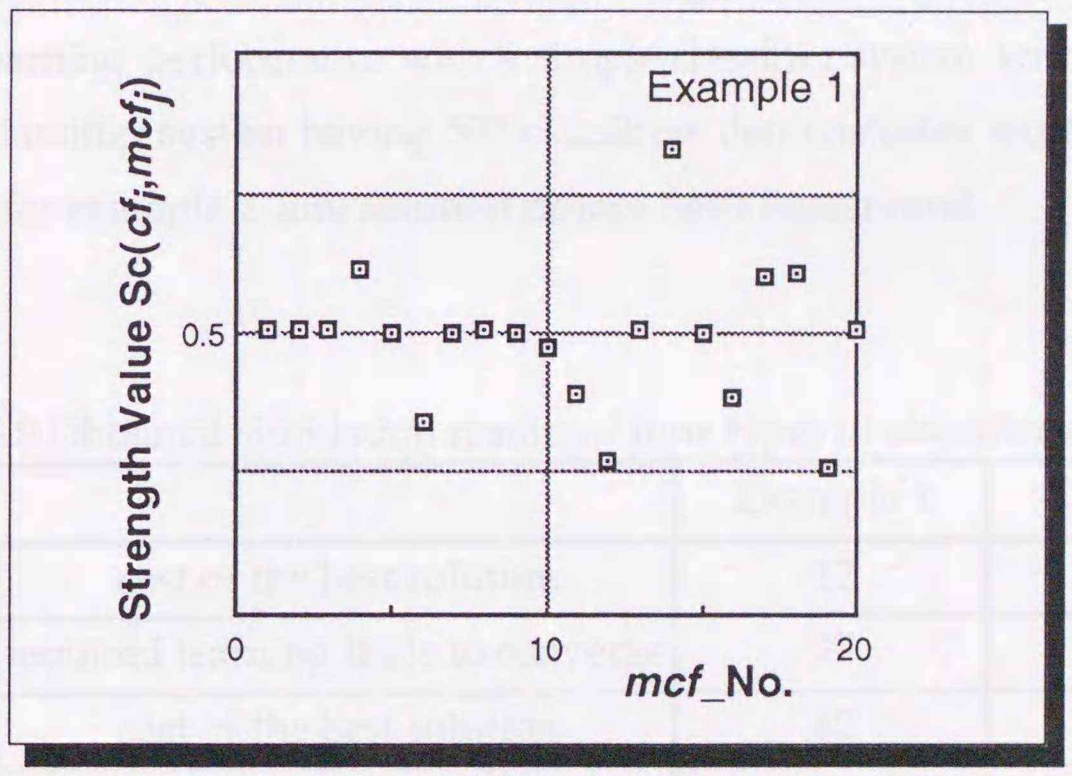


Figure 3.15 Revised connective strength values in the learned system for example 1 over 200 learning trials.
In this figure, plotted values are connective strength values between each masking classifier and the particular classifier, cf_{16}, that is verified as a good classifier in figure 3.14.

Next, to compare the learning performance of proposed approach with those of simple classifier systems, three simple classifier systems are implemented to the same tasks. Table 3.9 shows obtained simulation results of four types of classifier systems, thus, a masking classifier system having 100 classifiers and 20 masking classifiers, a simple classifier system with 2000 classifiers, a simple classifier system with 1000 classifiers, and a simple classifier system with 500 classifiers. In this table, two resultant values are expressed, thus, the operational cost of the found best solution and the required learning trials to converge. Let N and M be the number of classifiers and masking classifiers respectively, the number of reactive rules to be evaluated is equal to $N \times M$. However, in acquired results, the learning performance of a simple classifier system having 2000 classifiers is better than that of a masking classifier system with 100 classifiers and 20 masking classifiers. It seems that a simple classifier system with $N \times M$ classifiers tends to have higher diversity than a masking classifier system with N classifiers and M masking classifiers. It is also examined in this table that the masking classifier system has a similar grade of learning performance with a simple classifier system with 1000 rules. A simple classifier system having 500 classifiers demonstrates worse results. In particular, for example 2, any solution cannot have been found.

Table 3.9 Obtained simulation results of four types of classifier systems.

		Example 1	Example 2
100 cf_i + 20 mcf_j	cost of the best solution	12	20
	required learning trials to converge	97	153
2000 cf_i	cost of the best solution	12	16
	required learning trials to converge	138	212
1000 cf_i	cost of the best solution	14	24
	required learning trials to converge	59	87
500 cf_i	cost of the best solution	28	failure
	required learning trials to converge	48	—

Finally, I performed an additional experiment. Figure 3.16 shows a relatively complex task. In this task, there are twelve blocks and four slots. In this experiment, a thousand original classifiers and thirty masking classifiers are used in a masking classifier system. The learning curve of this masking classifier system for the task example 3 is illustrated in figure 3.17. Also for this task, a good solution is acquired by this machine learning system.

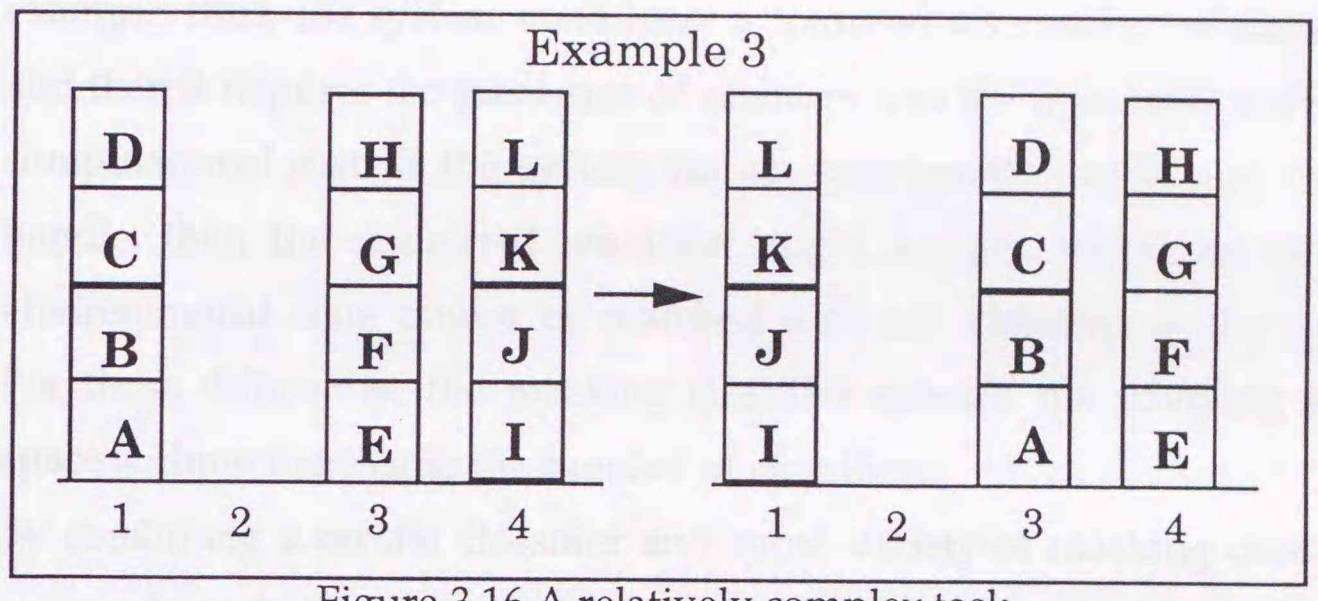


Figure 3.16 A relatively complex task.

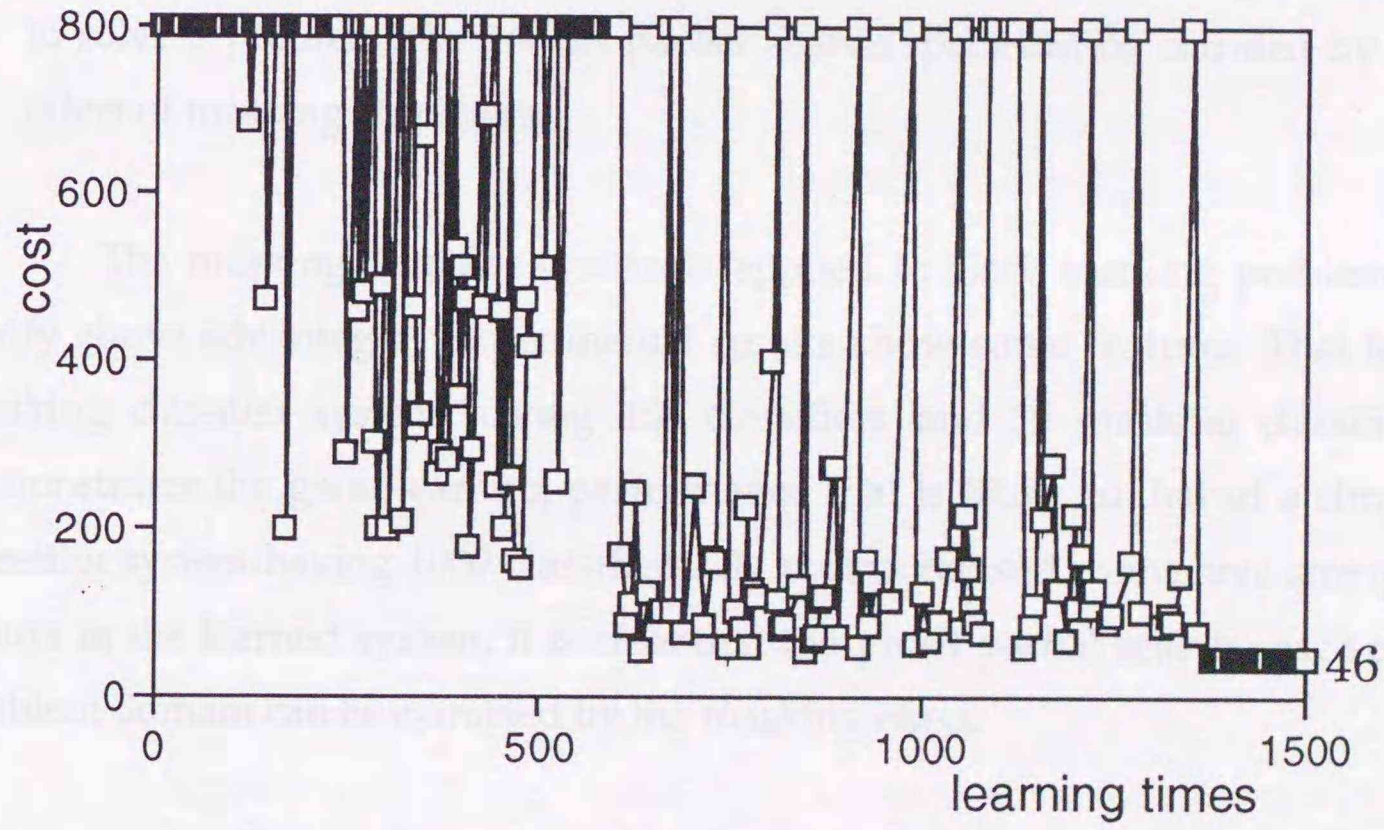


Figure 3.17 The learning curve of this masking classifier system for the task example 3.

3.4.5 Conclusions

In section 3.4, I defined the masking classifier to extend the representational performance of classifier systems. The proposed approach takes the following advantages;

(1) In a simple classifier system, when the size of internally represented problem space is huge, there are difficulties to learn a given task. For example, thus, the system must have a tremendous number of classifiers, and then it requires the great size of memory and the miserable amount of computational cost. If the system has an appropriate number of rules to handle, then the undesired situation might happen where an encoded environmental state cannot be matched with any classifier in the system. For these difficulties, the masking classifier extends the covering search space without increasing the number of classifiers.

(2) By combining a certain classifier and some variety of masking classifiers, some schemata can be examined for the classifier. As a result of this, a useful sub-space in a rule space is extracted for a problem state. That is, when only a partial information in a fully represented long string is needed to solve a problem, the benefit partial search space can be clarified by the effect of masking classifiers.

The masking classifier system is applied to block stacking problem to verify above advantages. Experimental results show some features. That is, a masking classifier system having 100 classifiers and 20 masking classifiers demonstrates the good learning performance that is likely to that of a simple classifier system having 1000 classifiers. By seeing revised connective strength values in the learned system, it is clear that the profit partial search space of a problem domain can be extracted by the masking effect.

References of Chapter 3

[Ackley and Littman 93] David Ackley and Michael Littman : "Interactions Between Learning and Evolution", Artificial Life II, Addison-Wesley, pp.487-509,1992.

[Antonisse 91] Antonisse, H.J. : "A Grammar-based Genetic Algorithms", in Foundation of Genetic Algorithms, Morgan Kaufmann, pp.193-204, 1991.

[Belew and Booker 91] Belew, R.K. and Booker, L.B. (ed.) : "Proceedings of the Fourth International Conference on Genetic Algorithms", Morgan Kaufmann, 1991.

[Belew and Gherrity 89] Belew,R.K. and Gherrity,M. : "Back Propagation for the Classifier System", in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.275-281, 1989.

[Belew et al. 92] R.K. Belew, J. Mclnerney, and N.N. Schraudoiph : "Evolving Networks: Using GA with Conectionist Learning", Artificial Life II, Addison-Wesley, pp.511-547,1992.

[Booker 88] Booker,L.B. : "Classifier Systems that Learn internal World Models", Machine Learning, 3, pp.161-192, 1988.

[Booker 89] Booker,L.B. : "Triggered Rule Discovery in Classifier Systems", in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.265-274, 1989.

[Booker 91] Booker,L.B. : "Representing Attribute-based Concepts in a Classifier System", in Foundation of Genetic Algorithms, Morgan Kaufmann, pp.115-127, 1991.

[Collard and Escazut 95] Collard,P. and Escazut,C. : "Relational Schemata: A Way to Improve the Expressiveness of Classifiers", in Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, pp.397-404, 1995.

[DeJong and Sarma 93] DeJong,K. and Sarma,J. : "Generation Gaps Revisited", in Foundation of Genetic Algorithms 2, Morgan Kaufmann, pp.19-23, 1993.

[DeJong and Spears 89] DeJong,K.A. and Spears,W.M. : "Using Genetic Algorithms to Solve NP-Complete Problems", in Proceedings of third ICGA, pp.124-132, 1989.

[Deb and Goldberg 89] Deb,K. and Goldberg,D.E. : "An Investigation of Niche and Species Formation in Genetic Function Optimization", in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.42-50, 1989.

[Dorigo 95] Marco Dorigo : "ALECSYS and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems", Machine Learning, Vol. 19, No.3, pp.209-240, 1995.

[Dorigo and Sirtori 91] Dorigo,M. and Sirtori,E. : "Alecsys: A Parallel Laboratory for Learning Classifier Systems", in Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, pp.296-302, 1991.

[Eshelman 91] Eshelman,L.J. : "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination", in Foundation of Genetic Algorithms, Morgan Kaufmann, pp.265-283, 1991.

[Eshelman 95] Eshelman, L.J. (ed.) : "Proceedings of the Sixth International Conference on Genetic Algorithms", Morgan Kaufmann, 1995.

[Feldman 93] Davis S. Feldman : "Fuzzy Network Synthesis with Genetic Algorithms," Proceedings of 5th ICGA, pp.312-317, 1993.

[Forrest 93] Forrest, S. (ed.) : "Proceedings of the Fifth International Conference on Genetic Algorithms", Morgan Kaufmann, 1993.

[Giani, Baiardi and Starita 94] Giani,A. Baiardi,F. and Starita,A. : "Q-Learning in Evolutionary Rule Based Systems", Parallel Problem Solving from Nature 3, pp.270-279, 1994.

[Goldberg 89] Goldberg, D.E., : "Genetic Algorithms in Search, Optimization & Machine Learning", Addison-Wesley, pp.217-307, 1989.

[Goldberg, Deb and Clark 93] Goldberg,D.E., Deb,K. and Clark,J.H. : "Accounting for Noise in the Sizing of Populations", in Foundation of Genetic Algorithms 2, Morgan Kaufmann, pp.127-140, 1993.

[Goldberg, Deb and Korb 91] Goldberg,D.E., Deb,K. and Korb,B. : "Don't Worry Be Messy", in Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, pp.24-30, 1991.

[Greene and Smith 94] Greene,D.P. and Smith,S.F. : "Using Coverage as a Model Building Constraint in Learning Classifier Systems, Evolutionary Computation, 2, 1, pp.67-91, 1994.

[Grefenstette 88] J.J. Grefenstette : "Credit Assignment in Rule Discovery Systems based on Genetic Algorithms", Machine Learning, Vol.3, pp.225-245, 1988.

[Grefenstette 91] Grefenstette , J.J. : "Lamarckian Learning in Multi-agent Environments ", in Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, pp.303-310, 1991.

[Grefenstette 93] Grefenstette , J.J. : "Deception Considered Harmful", in Foundation of Genetic Algorithms 2, Morgan Kaufmann, pp.75-91, 1993.

[Gruau 93] Frederic Gruau : "Genetic Synthesis of Modular Neural Networks," Proceedings of 5th ICGA, pp.318-325, 1993.

[Harp et al. 89] Steven A. Harp, Tariq Samad, and Aloke Guha : "Towards the Genetic Synthesis of Neural Networks," Proceedings of 3rd ICGA, pp.360-369, 1989.

[Holland 75] J.H.Holland, : "Adaptation in Natural and Artificial Systems", Univ. of Michigan Press, pp.20-120, 1975.

[Holland 85] J.H. Holland : "Properties of the Bucket Brigade Algorithm", Proceedings of 1st ICGA, pp.1-7, 1985.

[Holland et al. 86] J. H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R.Thagard, : "Induction", The MIT Press, pp.102-150, 1986.

[Horn, Goldberg and Deb 94] Horn,J. Goldberg,D.E. and Deb,K. : "Implicit Niching in a Learning Classifier System: Nature's Way", Evolutionary Computation, 2, 1, pp.37-66, 1994.

[Kawakami and Kakazu 93] Kawakami, T. and Kakaza,Y. : "A Study on an Autonomous Robot Navigation Problem using a Classifier System", Transactions of the Japan Society of Mechanical Engineers, C, Vol.59, No.564, pp.2339-2345, 1993. (in Japanese)

[Kawakami and Kakazu 95a] Kawakami,T., and Kakazu, Y. : "A Study on Robot Task Planning Problems in Multiagent Environments", in Proc. of IEEE International Conference on Robotics and Automation, pp. 2752-2757, 1995.

[Kawakami and Kakazu 95b] Kawakami,T. and Kakaza,Y. : "A Study on Evolutionary Synthesis of Classifier System Architectures by Genetic Algorithms", Transactions of the Japan Society of Mechanical Engineers, 1993. (submitted)

[Kitano 93] Kitano,H. : "Genetic Algorithms", Sangyo-Tosho, pp.234-262, 1993. (in Japanese)

[Maricic 92] Borut Maricic : "Dynamic versus Genetic versus Chaotic Programming", Dynamic, Genetic, and Chaotic Programming, John Wiley and Sons Inc., pp.501-533, 1992.

[Miller et al. 89] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde : "Designing Neural Networks Using Genetic Algorithms," Proceedings of 3rd ICGA, pp.391-397, 1989.

[Miyazaki et al. 94] Miyazaki,K., Yamamur,M. and Kobayashi,S. : "A Theory of Profit Sharing in Reinforcement Learning", Jour. of Japanese Society for Artificial Intelligence, vol.9, No.4, pp.580-587, 1994.(in Japanese)

[Montana and Davis 89] D. Montana and L. Davis : "training Feedforward Neural Networks Using Genetic Algorithms," Proceedings of IJCAI-89, 1989.

[Narendra and Thathachar 91] K.S. Narendra and M.A.L. : "Learning Automata :An Introduction", Prentice Hall, 1991.

[Patel and Dorigo94] Mukesh J. Patel and Marco Dorigo : Adaptive Learning of a Robot Arm, Evolutionary Computating (T.C. Fogarty ed.), Springer Verlag, pp.180-194, 1994

[Rawlins 91] Rawlins, G.J.E. (ed.) : "Foundations of Genetic Algorithms", Morgan Kaufmann, 1991.

[Rendon 91] Rendon,M.V. : "The Fuzzy Classifier System: A Classifier System

for Continuously Varying Variables", in Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, pp.346-353, 1991.

[Riolo 89] Riolo, R.L. : "The Emergence of Coupled Sequences of Classifiers", in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.256-264, 1989.

[Roberts 93] Roberts,G. : "Dynamic Planning for Classifier Systems", in Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, pp.231-237, 1993.

[Rumelhart, Hinton and Williams 86] Rumelhart,D., Hinton,G. and Williams,R. : "Learning Internal Representations by Error Propagation", in Parallel Distributed Processing, Bradford Books, 1986.

[Samuel 59] A.L. Samuel : "Some Studies in Machine Learning using the Game of Checkers", IBM Journal on Research and Development, Vol.3, pp.210-229, 1959.

[Schaffer 89] Schaffer, J.D. (ed.) : "Proceedings of the Third International Conference on Genetic Algorithms", Morgan Kaufmann, 1989.

[Shuurmans and Schaeffer 89] Shuurmans,D. and Schaeffer,J. : "Representational Difficulties with Classifier Systems", in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.328-333, 1989.

[Shu and Schaeffer 89] Shu,L. and Schaeffer,J. : "VCS: Variable Classifier Systems", in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.334-339, 1989.

[Smith 94] Smith, R.E. : "Introduction to the Special Issues", Evolutionary Computation, Vol. 2, No. 1, 1994.

[Smith and Wilson 89] Smith,S. and Wilson,S.W. : "Rosetta: Toward a Model of Learning Problems", in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.347-350, 1989.

[Spears and DeJong 91] Spears,W. and DeJong,K., : "An Analysis of Multi-point Crossover", in Foundation of Genetic Algorithms, Morgan Kaufmann, pp.301-315, 1991.

[Sutton 88] R.S. Sutton : "Learning to Predict by the Methods of Temporal Differences, Machine Learning, Vol.3, pp.9-44, 1988.

[Syswerda 89] Syswerda, G. : "Uniform Crossover in Genetic Algorithms", in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.2-9, 1989.

[Unemi 94] Unemi,T. : "Reinforcement Learning", Jour. of Japanese Society for Artificial Intelligence, vol.9, No.6, pp.830-836, 1994.(in Japanese)

[Unemi et al. 94] Tatsuo Unemi, et al. : "Evolutionary Differentiation of Learning Abilities", Artificial Life IV (R.A. Brooks and P.Maes

editioned), MIT press, pp.331-336, 1994.

[Watkins and Dayan 92] C.J.C.H. Watkins and P. Dayan : "Technical Note :Q-Learning", Machine Learning, Vol.8, pp.55-68, 1992.

[Whitely and Hanson 89] Darrel Whitely and Thomas Hanson : "Optimizating Neural Networks Using Faster, More Accurate Genetic Search," Proc. of 3rd ICGA, pp.391-397, 1989.

[Whitley 93] Whitley, L.D. (ed.) : "Foundations of Genetic Algorithms 2", Morgan Kaufmann, 1993.

[Whitley and Hanson 89] Whitley,D. and Hanson,T. : "Optimizing Neural Networks using Faster, More Accurate Genetic Search", in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.391-397, 1989.

[Wilson 94] S.W. Wilson : ZCS :A Zeroth Level Classifier System, Evolutionary Computation, Vol.2, No.1, pp.1-18,1994.

[Wilson and Goldberg 89] S.W. Wilson and D.E. Goldberg, : A Critical Review of Classifier Systems, in *Proc. of 3rd ICGA*, pp.244-255,1989.

[Wright 91] Wright, A.H. : "Genetic Algorithms for Real Parameter Optimization", in Foundation of Genetic Algorithms, Morgan Kaufmann, pp.205-218, 1991.

[Yamada and Taki 94] Yamada,S. and Taki,H. : "Current Studies on Machine Learning", Jour. of Japanese Society for Artificial Intelligence, vol.9, No.6, pp.817, 1994.(in Japanese)

[Yamamura et al. 95] Yamamura, M., Miyazaki,K. and Kobayashi,S. : "A Survey on Reinforcement Learning", Systems, Control and Information, Vol.39, No.4, pp.191-196, 1995.(in Japanese)

CHAPTER 4

Engineering Applications By The GAPS

4.1 Preliminary

In this chapter, some engineering problems are solved by implementing the GAPS. I have chosen some variety of problem domains to examine the problem solving performances of GAPS. Treated problem domains are combinatorial optimization problems, reactive planning problems, and scheduling problems. Definitely, at first, 3-Dimensional Packing Problems are effectively solved in section 4.2. In this application, 3-Dimensional packing strategies are evolved by adaptive tuning systems in which the hybrid tuning method of GA and heuristics is applied. In multiple containers environments, hierarchical tuning mechanisms are also proposed. Based on evolved packing strategies, the possibility of development of 3-D packing rule-base systems is discussed. Next in section 4.3, Autonomous Mobile Robot Navigation Problems are solved by a classifier system-based reinforcement learning mechanisms. In this application, multiagent environments, in which multiple mobile robots exist in one navigation area, are also treated. As the third application, Sequencing Problem in Process Planning is described in section 4.4. This problem is an ordering problem with geometrical constraints. In section 4.5, then, the GAPS also solved the Robot Task Planning Problems in multiagent environments, that is a distributed autonomous planning problem. In this problem domain, I attempt to realize a mechanism by which cooperative

strategies of agents can be acquired. Finally, Reactive Motion Planning Problems of robot manipulators is introduced in section 4.6.

4.2 The Development of Adaptive Tuning Systems of 3-Dimensional Packing Strategies

4.2.1 Introduction

In this application, the new approach of how to solve the 3-D packing problem automatically is proposed. When we attempt to construct the automatic packing system in an automated factory, we must solve some difficult problems. One of the most difficult problems is determination of packing sequence which largely effects the solution of 3-D packing problem. This problem is well known as a complex combinatorial problem as well as 3-D space allocation one. Namely, a number of given boxes with different sizes should be packed into a given rectangular space called a container so that the least amount of empty space is left. Even by skilled human workers, the problem is beyond their control, as the size and the variety of the boxes increase. Therefore, the development of an automatic system with high packing performance is required. Until today, there were some approaches to the problem, e.g., approaches from mathematical one to production rule-based one [Minagawa, Kanazawa and Kakazu 89]. And all of these approaches adopted the predetermined theories or rules which are subject to the problem itself. Therefore, it is difficult to apply the approaches to the automatic packing system.

So, I attempt to realize the mechanism that 3-D packing rule is automatically tuned and optimum packing solution is obtained. To realize this mechanism, I applied genetic algorithms [Holland 75] [Goldberg 87] [Davis and Steenstrup 87] which mimic the process of natural evolution system. By the mechanism, packing performance is improved gradually, and near-optimal allocation can be obtained. Genetic algorithms are well known in their applications for nonlinear optimization problems. Using genetic algorithms, an

object to be tuned is encoded to a string using alphabetic symbols and numbers. A population is generated as a set of strings, and each string is determined to survive or die depending on their search result. The strings evolve automatically by using genetic operators.

In this approach, genetic algorithms are hybridized with heuristics. That is, the packing strategy is controlled by two evaluation functions that dominate the heuristic packing rule. The heuristic 3-D packing procedure in a container consists of two steps. First, evaluation values of allocatable positions in the container are calculated by the evaluation function. Allocation positions are determined according to calculated values. Second, a box is placed on a predetermined next allocation position. The most suitable box is selected from a set of unallocated boxes according to another evaluation function. These two steps are executed recursively until the allocation space satisfies the termination conditions. Since calculated packing results largely depend on the combination of the weighted coefficient values of the two evaluation functions, an agent has to search for one which gives a near-optimal solution, i.e., the agent acquires the searched string of coefficient values as the packing strategy. To find the near-optimal strategy for the individual problem, the weighted coefficients of the evaluation functions are tuned by applying the genetic operators such as a reproduction operator, a crossover operator and a mutation operator.

And, to use the obtained tuned results as accumulated successful strategies, I construct a 3-D packing rule-base. The rules in rule-base are composed of "conditional part" which expresses the features of the given problem and "procedural part" which gives the packing strategy. To select a useful packing rule from rule-base for the given problem, conditional matching is carried out between the feature information of input data and the that of rules.

Furthermore, I treat the extended 3-D packing problems, that is in a multi-container environment. The difficulties of this problem are compounded in a multiagent environment in which there are many containers and multiple agents that perform the packing operations of corresponding containers. Because, it is very difficult to determine which box should be assigned to a container, this determination then affects the whole packing performance. In

this study, the 3-D packing strategy in a multiagent environment utilizes an agent strategy and a supervisor strategy. These strategies are acquired through a hierarchical tuning. This is achieved by an agent strategy tuning layer and a supervisor strategy tuning layer. In the first step, each agent acquires the individual packing strategy related to the corresponding container without the interactions from the other agent's strategies. In the next step, a supervisor strategy is searched as a priority sequence for agents. An agent packing strategy is represented by two evaluation functions, these are the same as the that in single container domain. After each agent strategy pertaining to a corresponding container is acquired by the genetics-based adaptation mechanism, a supervisor strategy should be also searched. A supervisor strategy is represented as a priority among agents. Conflicts among agents are solved by this strategy. Thus, each agent performs the packing of a container in the order of priority. The agent having higher priority carries out the packing, other agents having lower priorities have to wait for their executions. To implement genetic algorithms in a supervisor strategy, a string is formed as a sequence of the numbers of agents. However, simple genetic algorithms cannot be applied to such an order-based string. Therefore, a supervisor strategy must also be tuned by modified genetic operators for order-based strings, i.e., order-based uniform crossover and order-based mutation.

Based on the proposed method, I carried out some numerical experiments on a constructed 3-D packing simulator. First, experiments of the agent strategy acquisition are carried out by genetics-based adaptation. Next, I show the experimental results of 3-Dimensional packing rule-base systems. Furthermore, I also carried out experiments of the supervisor strategy acquisition in a multiagent environment. In this experiment, a priority sequence of eight agents having individual strategies is tuned, and the improvement of the whole packing performance is shown. In the remainder of this paper, the details of this methodology are descried, and the usefulness of the proposed method is examined .

4.2.2 Related Works

As we already know, a lot of interesting researches from theoretical investigations to engineering applications have been achieved from Holland presented the framework of his genetic algorithms[Holland 75]. The strong performance of genetic algorithms as an optimization technique has been examined for some difficult problems which are hard to solve theoretically. Particularly in the combinatorial optimization problem field, successful results have been reported for traveling salesman problem, job shop scheduling problems and one-dimensional bin packing problems [Goldberg and Lingle 85] [Davis 85] [Smith 85] [Yamamura, Ono and Kobayashi 92]. As a pattern nesting problem, Koakutsu, Sugai and Hirata proposed a method to optimize the placement of LSI blocks by simulated annealing based on genetic algorithms [Koakutsu, Sugai and Hirata 90], and the integration of artificial neural networks and genetic algorithms is explored as a solution approach for the irregular pattern nesting problem in [Poshyanonda and Dagli 93]. In the 2-dimensional space, a graph partitioning problem was also solved by genetic algorithms [Kitano 93]. However, few GA-based approach has been proposed for the three dimensional packing problem. Because, this problem is more complex than two dimensional nesting problems such as LSI pattern placement problems. In the 3-dimensional packing problem, complicated geometrical constraints and a huge number of feasible problem states make this problem complex. As an exclusive research, House and Dagli reported an approach to 3-dimensional packing using genetic algorithms [House and Dagli 92] [House and Dagli 93]. Their approach method is too simple and too direct. In their method, the packing procedure is described as follows:

(1) Random lists of the boxes to packed are created. The lists show the order the boxes are to be packed in the container.

(2) Then, take first box from the list and put it on a firstly found allocable position in a container.

(3) Get the next box from the list, and attempt to allocate it in a remaining space.

(4) Repeat until all boxes are checked.

(5) The lists are passed one at a time to the evaluation algorithm. The evaluation algorithm determines the fitness ratio of a list.

(6) Then a set of evaluated lists is modified by genetic operators.

Thus, a chromosome is represented as a packing list that is a permutation of all boxes. This means that when a lot of boxes are given, a represented chromosome becomes very long string. Since the search space becomes huge, a gigantic number of evaluation times are required to find an appropriate solution even by using genetic algorithms. Moreover, they didn't consider about the influence of a selected allocation position of a box. It, so, seems that their packing system cannot find good solution for complicated packing problems.

The other hand, my proposed approach adopts the hybrid method of the packing heuristics and genetic algorithms. The packing procedure is controlled by the packing heuristics, and the heuristics are evolved by genetic algorithms. This implies that the search space for genetic algorithms does not depend upon the number of given boxes. That is the main advantage of this approach method.

In addition, there is no research treating the 3-D packing problem in multi-container environments. Therefore this research would have a contribution for engineering.

4.2.3 Description of the Three-Dimensional Packing Problem

In what follows, I address the problem treated in this paper. In this description, I restrict the problem so that given containers and boxes have a dimensional attribute, but they don't have other attributes such as weight and so on.

Here, to solve the problem by the GAPS, I define the 3-dimensional packing problem as an environmental adaptive problem That is, the environmental adaptive problem, EAP, is represented by following four-tuple;

$$EAP = <E, A, G, R>, \tag{4-2-1}$$

where E is the environment of a system undergoing adaptation, A is a set of structures in the system, G is the adaptive plan which determines successive structural modifications in response to the environment, and R is a measure of the performance of the structure in the environment. The 3-D packing problem is described by the above four objects as follows:

1)*E*: The input data of the 3-D packing problem. The input data is described by a set of dimensions of a container dc and one of boxes DB. That is, I define the following objects about the size of the container and boxes.

$$dc = (W , L , H) \tag{4-2-2}$$

$$DB = \{ db_j ; j=1\sim N_b \} \tag{4-2-3}$$

$$db_j = (w_j , l_j , h_j) \tag{4-2-4}$$

$$NB = \{nb_j ; j= 1 \sim N_b\} \tag{4-2-5}$$

where dc is a given container, W, L and H are the dimensions of dc , db_j is the box of the j-th type , w_j , l_j and h_j are the dimensions of db_j . NB is a set of numbers of boxes, and nb_j is the number of the given j-th type boxes.

To treat the packing problem as a 3-dimensional geometrical one, I set a container in an orthogonal coordinates system so as that an arbitrary point (x, y, z) in the container satisfies the following equation.

$$(0 \le x \le W), (0 \le y \le L), (0 \le z \le H). \tag{4-2-6}$$

In this container, a box must be placed so that every surface of the box intersects with one of the coordinate axes orthogonally.

2)*A*: A set of the packing strategies. Each strategy is represented by heuristics. From a packing strategy, we can deduce a packing result which involves

the number of the placed boxes, the packing sequence, and the allocation positions in the container.

3)*G*: Genetic algorithms as the evolutionary adaptive plan. With genetic algorithms, each genetic operator is applied to the strategies according to the their packing results.

4)*R*: A measure of the packing result. The packing criteria, *R*, corresponds to the capacity rate that represents the degree of effectiveness of a packing result This capacity rate is defined as a ratio of the given containers' volumes to the sum of the packed boxes' volumes. The capacity rate is calculated by the following equation, and the optimal packing operation is searched so that the capacity rate is maximized.

$$R = \frac{\sum_{j=1}^{N_b} f_j \cdot (w_j \cdot l_j \cdot h_j)}{(W \cdot L \cdot H)}, \qquad (4\text{-}2\text{-}7)$$

where, f_j is the number of packed j-th type boxes. Thus, genetic operators are applied such that the capacity rate is maximized.

4.2.4 Hybrid Adaptation Method in Single Container Environments

In this section, I describe the 3-dimensional packing strategies that use heuristics to achieve the effective packing. Then, the implementation method of genetic algorithms is also defined to evolve this packing strategy.

4.2.4.1 The 3-Dimensional Packing Strategy

The 3-D packing strategy consists of the following two steps (figure 4.1) .

S1) Evaluation values of allocable positions in the given container are calculated by evaluation function. Next allocation positions are determined according to the calculated values.

S2) A box is placed on a predetermined next allocation position. The most suitable box is thus selected from the set of unallocated boxes according to another evaluation function.

These two steps are executed recursively until the allocation space satisfies the termination conditions.

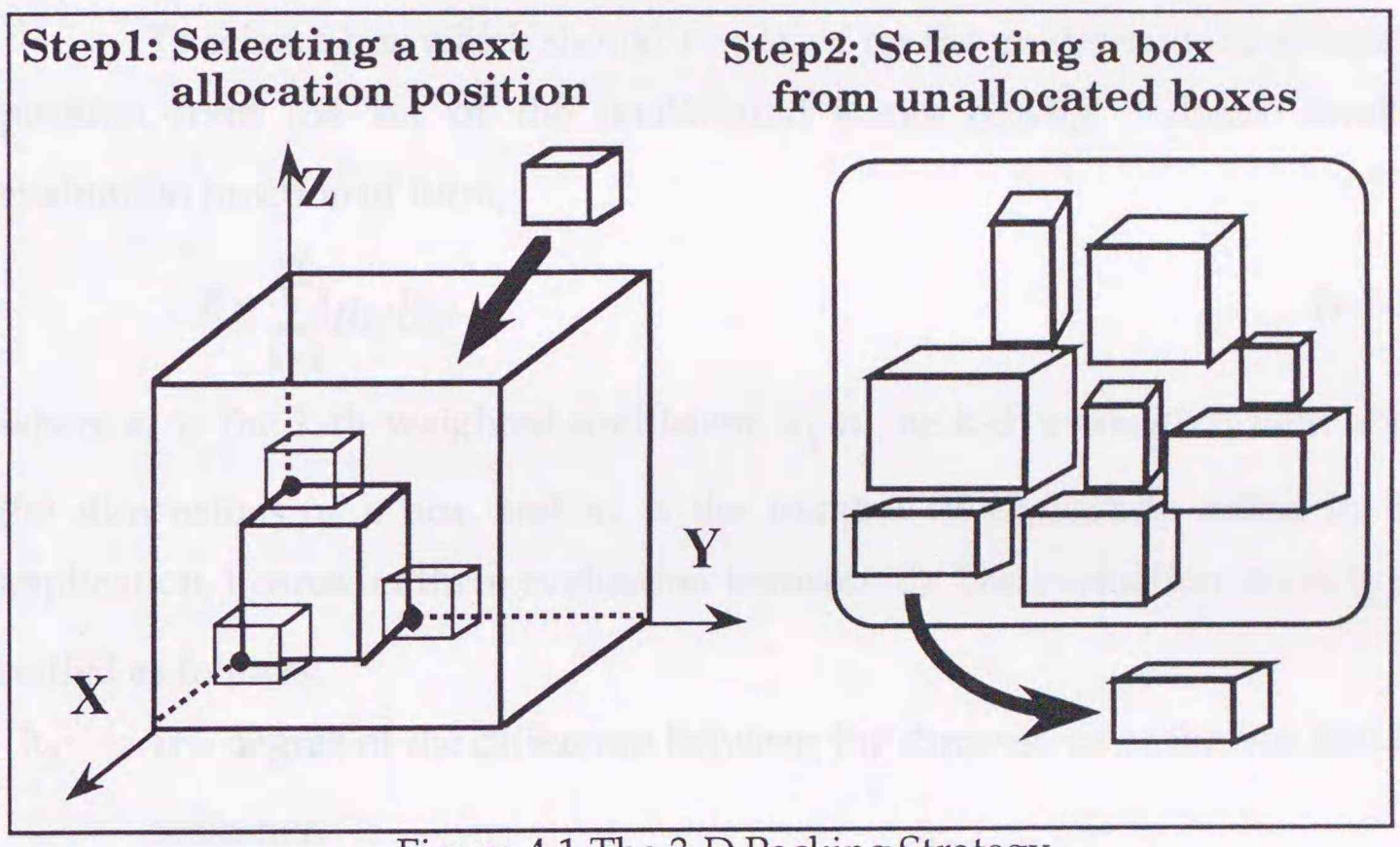


Figure 4.1 The 3-D Packing Strategy

4.2.4.2 Evaluation Function to Select an Allocation Position

To determine a next allocation position where a next box should be placed, an evaluation function is prepared. According to the calculated values of the evaluation function, a most suitable allocation position is selected from the set of the allocable positions $\{AP_i\}$. Thus, an evaluation value P_i is given as;

$$P_i = f(AP_i) \equiv f(x_i, y_i, z_i). \tag{4-2-8}$$

Here, the function P is defined in the form of;

$$P_i = e1 \cdot x_i^2 + e2 \cdot y_i^2 + e3 \cdot z_i^2 \qquad (4\text{-}2\text{-}9)$$

where (x_i, y_i, z_i) are the coordinates of the i-th allocatable position, and $e1$, $e2$ and $e3$ are weighted coefficients. According to the following equation, an allocatable position which has P^* is selected as the next allocation position.

$$P^* = \min_{i \in IP} [P_i] \qquad (4\text{-}2\text{-}10)$$

where IP is a set of the numbers of the allocatable positions. This evaluation function is formed to represent the heuristic packing knowledge about an allocation position.

4.2.4.3 Evaluation Function to Select a Box

To select a box which should be placed on the predetermined allocation position from the set of the unallocated boxes $\{UAB_i\}$, I define another evaluation function of form;

$$B = \sum_{k=1}^{ne} (a_k \cdot b_k) \qquad (4\text{-}2\text{-}11)$$

where a_k is the k-th weighted coefficient, b_k is the k-th evaluation item about the dimensions of a box, and ne is the number of evaluation items. In this application, I consider three evaluation items(ne=3). The evaluation items b_k are settled as follows:

b_1: The degree of the difference between the dimensions of the box and the container.

$$b_1 = (w_i/W)^2 + (l_i/L)^2 + (h_i/H)^2 \qquad (4\text{-}2\text{-}12)$$

b_2: The degree of tallness of the box.

$$b_2 = h_i^2/(w_i^2 + l_i^2) \qquad (4\text{-}2\text{-}13)$$

b_3: The degree of the difference of bottom area between the box and the container.

$$b_3 = (w_i * l_i)/(W * L) \qquad (4\text{-}2\text{-}14)$$

where w_i, l_i, and h_i are the dimensions of the i-th type box and W, L, and H are the dimensions of the container. According to the following function, a box

having B* is selected for next allocation.

$$B^* = \max_{i \in Inb}[B_i] \tag{4-2-15}$$

where *Inb* is a set of the numbers of types of given boxes. This evaluation function and evaluation items are also defined to represent the heuristic packing knowledge about a box.

4.2.4.4 Altering the Allocation Space

During the execution of the two procedures above, it is necessary to alter the current state of allocation space based on the latest allocation result. Thus, the set of allocatable positions $\{APC_i\}$ is altered every time a box is placed.

This process is summarized as follows:

1) New positions are added to the $\{APC_i\}$ after each allocation. When a box having the dimensions (w, l, h) is placed on coordinates (x, y, z), the coordinates of new allocatable positions are given as the following equation(figure 4.2).

$$NEW_P = \{ (x+w, y, z) , (x, y+l, z) , (x, y, z+h)\}. \tag{4-2-16}$$

The NEW_P are added to the $\{APC_i\}$.

2) The positions on which no box can be placed are deleted from the set of allocatable positions $\{APC_i\}$.

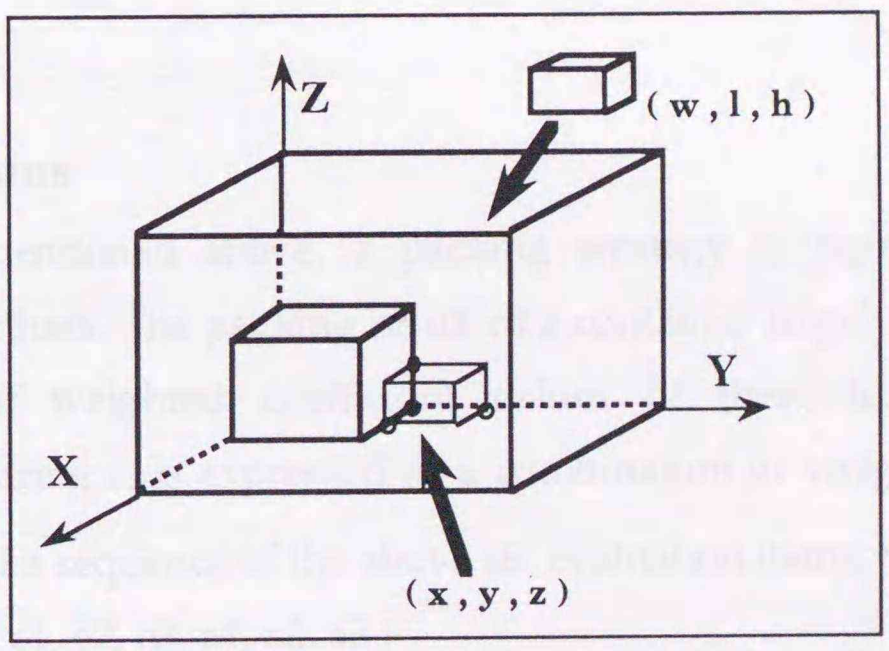


Figure 4.2 Altering a set of allocable positions.

4.2.4.5 The Termination Conditions

The packing procedure is terminated when an allocation space satisfies either or both of following conditions.

1) No box remains for further allocation, i.e.,

$$\{UAB_i\} = \phi. \tag{4-2-17}$$

2) No allocatable position exists in the container, i.e.,

$$\{APC_i\} = \phi. \tag{4-2-18}$$

4.2.4.6 Implementing the Genetic Algorithms

Calculated packing results largely depend on a combination of the weighted coefficients of the evaluation functions. Therefore, when I attempt to find the function which gives a near-optimal solution, i.e., I "adapt" the strings of the coefficient values by implementing genetic algorithms. Genetic algorithms are search algorithms based on natural selection and natural genetics organized by the evolutional pressure in a biological system. In genetic algorithms, an object to be tuned is encoded to a string using letters and numbers, and a population is generated as a set of strings. Each string is determined to survive or die depending on its fitness value. Values are calculated based on each search result. Using this mechanism, packing performance is improved gradually, and a near-optimal solution can be obtained.

Representations

As I mentioned above, a packing strategy is represented by two evaluation functions. The packing result of a container largely depends on the combination of weighted coefficient values of these functions. In this application, a string s_i is expressed as a combination of weighted coefficients which consist of a sequence of the above six evaluation items, that is,

$$s_i = (\, e1, e2, e3, a1, a2, a3 \,) \tag{4-2-19}$$

where s_i is an i-th string. As the set of the strings, a population A(t) at generation t is generated, that is,

$$A(t) = \{ s_i; i = 1 \sim pop_size \}, \tag{4-2-20}$$

where *pop_size* is the number of strings in a population that does not vary in time. To automatically tune up the strings, genetic operators such as conventional reproduction, crossover and mutation are applied over several generations.

Reproduction Operator

A reproduction operator takes the role of copying useful strings for use in next generation. This is carried out according to each string's selection possibilities that are calculated using the capacity rate, R, defined in section 4.2.3. Thus, the selection possibility of an *i*-th string, P_repro_i, is given as follows:

$$P_repro_i = \frac{(R_i - R_{worst})^2}{\sum_{j=1}^{pop_size} (R_j - R_{worst})^2}, \tag{4-2-21}$$

where,

$$R_{worst} = \min_{i \in IS}[R_i], \tag{4-2-22}$$

and, *IS* is a set of the string's numbers. Using P_repro_i, the reproduced number of the *i*-th string to the next generation *t+1* is given as;

$$N_repro_i = P_repro_i \bullet pop_size. \tag{4-2-23}$$

Each string is stored into a mating pool according to the N_repro_i for further operations.

Crossover Operator

A crossover operator effects on stored strings in the mating pool. This operation consists of the following two steps.

1) Random selection of the two strings to be mated.
2) The random exchange of sub-strings on the crossover position.

This is called the conventional one point crossover operation.

Mutation Operator

A mutation operator also has an effect on the strings in the pool. In this operation, the value of a selected string is alternatively changed from 1 to 0.

4.2.5 Development of a Three-Dimensional Packing Rule-base

To use the obtained tuned results as an accumulated set of successful strategies, I constructed a 3-D packing rule-base. The rule-base has a mechanism in which the feature information of a given problem is extracted, and a packing strategy with high performance is selected and applied to the problem(figure 4.3). Each rule in the rule-base is composed of two parts: a conditional part, which expresses the features of the given problem, and a procedural part, which gives the packing strategy. Thus, the procedural part corresponds to the tuned string above.

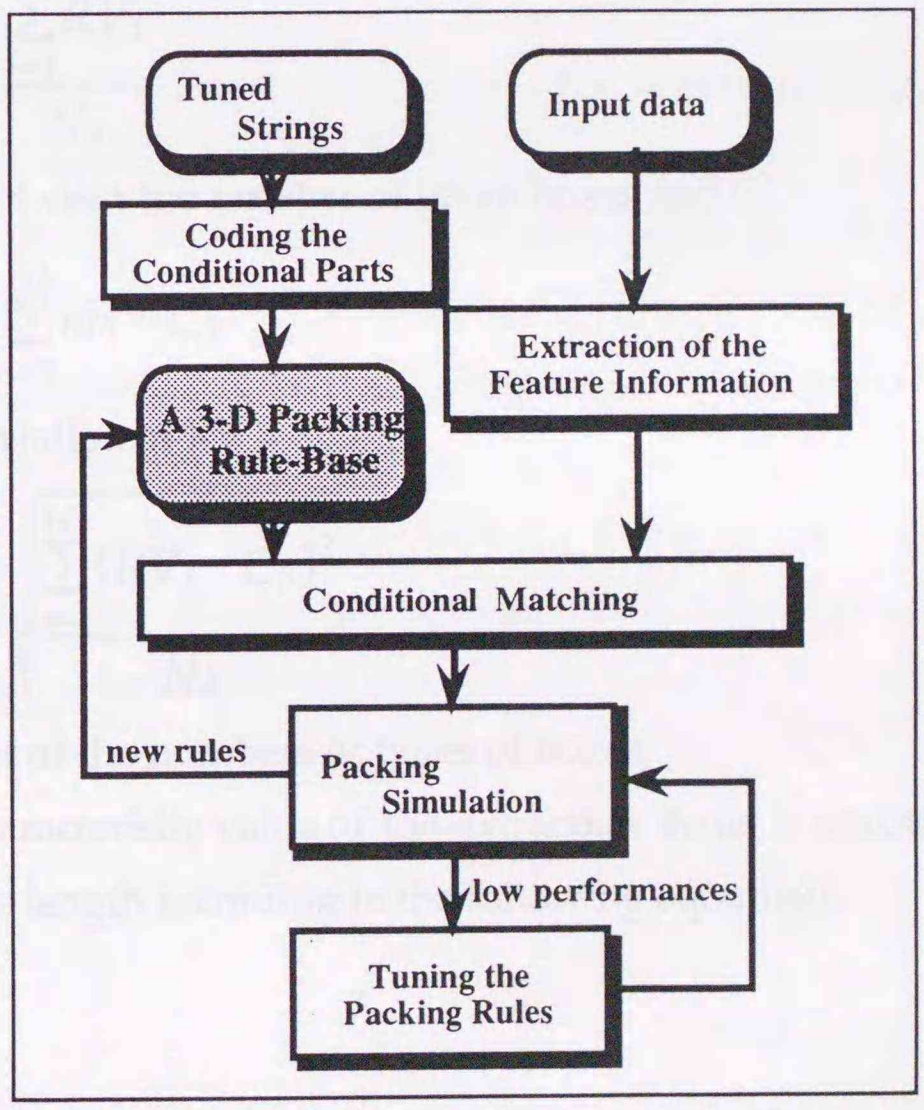


Figure 4.3 A conceptual flow of a 3-D packing rule-base.

4.2.5.1 Extraction of the Feature Information

To generate a conditional part, the feature information of the given problem is extracted. In the conditional part, this feature information is represented by using the ratio of the volumes RV between the container and each box, i.e.,

$$RV_i = \frac{(W \cdot L \cdot H)}{(w_i \cdot l_i \cdot h_i)} \tag{4-2-24}$$

Using RV_i, I define the characteristic values of the five different feature extraction items as follows:

C_1: Minimum value of RV

$$C_1 = \min_{i \in Inb}[RV_i] \tag{4-2-25}$$

C_2: Maximum value of RV

$$C_2 = \max_{i \in Inb}[RV_i] \tag{4-2-26}$$

C_3: Average value of RV

$$C_3 = \frac{\sum_{i=1}^{N_b} RV_i}{N_b} \tag{4-2-27}$$

C_4: Difference between the number of given boxes and C_3

$$C_4 = \sum_{i=1}^{N_b} nb_i - C_3 \tag{4-2-28}$$

C_5: Standard deviation of RV

$$C_5 = \sqrt{\frac{\sum_{i=1}^{N_b} (RV_i - C_3)^2}{N_b}} \tag{4-2-29}$$

where *Inb* is a set of the numbers of types of boxes.

Each characteristic value of the extraction items is encoded into a sub-string of five bits length according to the following equations.

$$\begin{cases} s_{ik} = 1\text{: } if\ (k-1)\cdot rc_i \le C_{ij} - \min C_i < k\cdot rc_i\ ,\ (k \ne 5) \\ \qquad\qquad (k-1)\cdot rc_i \le C_{ij} - \min C_i \le k\cdot rc_i\ ,\ (k = 5), \\ s_{ik} = 0\text{: } otherwise \end{cases}$$

$$(i=1,2,...,5,\ k=1,2,...,5), \tag{4-2-30}$$

where,

$$\min C_i = \min_{j \in Jr}[C_{ij}], \tag{4-2-31}$$

$$rc_i = (\max_{j \in Jr}[C_{ij}] - \min_{j \in Jr}[C_{ij}]) \Big/ 5, \tag{4-2-32}$$

and *Ski* is a value of the k-th bit of the i-th extraction item, *Cij* is a value of the i-th extraction item of the j-th rule, and *Jr* is a set of the numbers of rules in the rule-base. Finally, a conditional part is generated as a sequential combination of the five sub-strings above.

4.2.5.2 The Conditional Matching

To select a rule from rule-base for a given problem, conditional matching is carried out between the feature information of the given problem and the that of rules. The matching process is carried out on every feature item by simply comparing each of the sub-strings; the difference, which is evaluated by counting the number of positions having value 0 between those having value 1 , is calculated. The summation of each item's difference between the given problem and the conditional part of a rule is defined as the feature distance. As the winner rule of rule competition, I select the rule having a minimum value of the feature distance.

When conditional matching fails or a low packing performance is detected, the problem is entirely solved again using the above tuning mechanism. After this searching, the tuned result is added to the rule-base.

4.2.6 Hierarchical Adaptation of 3-D Packing Strategies in Multiagent environments

In this section, I extend the 3-dimensional packing problem defined above. In particular, the difficulties of this problem are compounded in a multiagent environment in which there are multiple agents that perform the packing of corresponding containers. Here, I attempt to realize the mechanism by which the 3-D packing strategy is automatically tuned and by which the optimum packing solution is obtained in a multiagent environment by applying genetic algorithms. The 3-D packing strategy in a multiagent environment is acquired through a hierarchical adaptation. This is achieved by an agent strategy tuning layer and a supervisor strategy tuning layer. First, each agent acquires the individual packing strategy related to the corresponding container. An agent packing strategy is controlled by two evaluation functions which are defined in section 4.2.4. Next a supervisor strategy is implemented as a priority sequence of agents. A supervisor strategy is also tuned by modified genetic operators for order-based strings.

4.2.6.1 The Extended Problem Setting

In what follows, I define the extended problem setting in multiagent environments. In this problem setting, a set of dimensions of containers, *DC*, is introduced, thus, the equation (4-2-2) is changed to following equations.

$$DC = \{ dc_i ; i=1\sim N_c \}, \tag{4-2-33}$$

$$dc_i = (W_i , L_i , H_i), \tag{4-2-34}$$

where dc_i is a i-th container, W_i , L_i and H_i are the dimensions of dc_i , N_c is the number of given containers. For these containers, there are packing agents pa_i and each agent achieves the packing of the corresponding container (figure 4.4).

$$PA = \{ pa_i ; i=1\sim N_c \}. \tag{4-2-35}$$

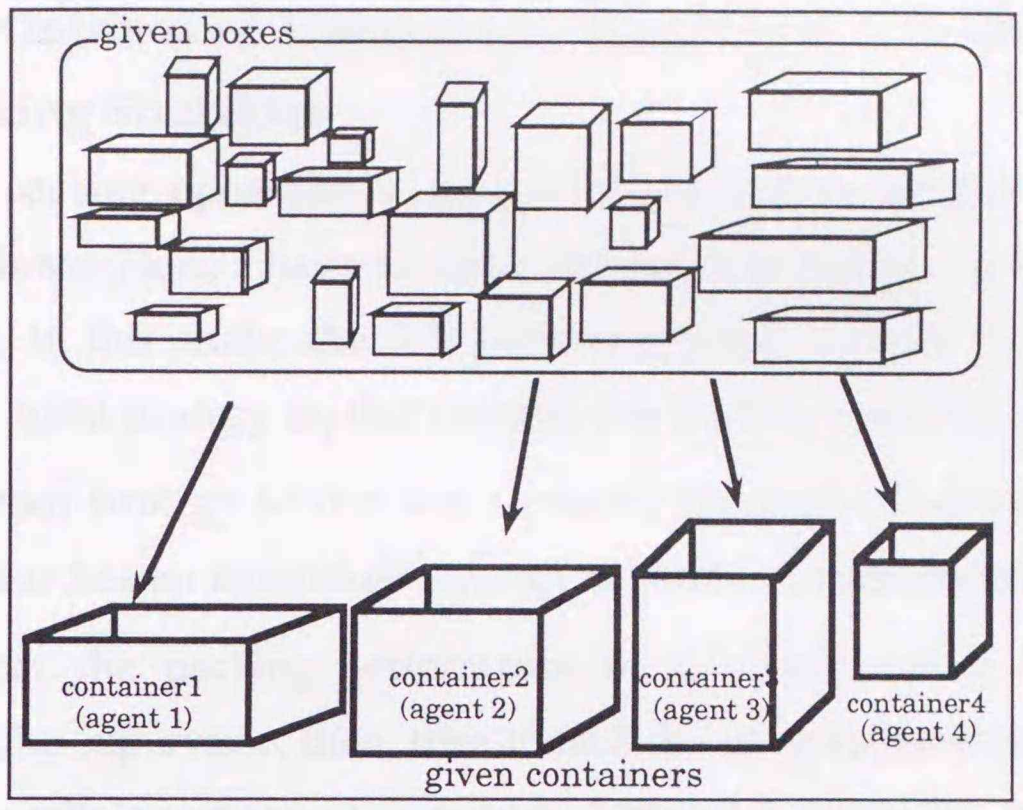


Figure 4.4 The 3-D packing problem in multiagent environment.

The 3-D packing problem in multiagent environment is very difficult to solve, because it is hard to decide which box should be assigned to the container so that the whole packing performance is maximized. Here, a measure of the packing result in a multiagent environment is also needed to change. The packing criterion corresponds to the capacity rate that represents the degree of effectiveness of whole containers. This capacity rate is defined as a ratio of the sum of the given containers' volumes to the sum of the packed boxes' volumes. The capacity rate is calculated by the following equation.

$$R = \frac{\sum_{i=1}^{N_c}\sum_{j=1}^{N_b} f_{ij} \cdot v_j}{\sum_{i=1}^{N_c} V_i}, \qquad (4\text{-}2\text{-}36)$$

where f_{ij} is the number of packed j-th type boxes in the i-th container, V_i and v_j are volumes of the i-th container and the j-th type box respectively. Thus,

$$V_i = W_i * L_i * H_i, \qquad (4\text{-}2\text{-}37)$$

$$v_j = w_j * l_j * h_j. \qquad (4\text{-}2\text{-}38)$$

4.2.6.2 An Outlines of Hierarchical Adaptation of 3-Dimensional Packing Strategies

The packing operation is achieved based on the problem definition mentioned above. Here, I have to show the packing strategy in a multiagent environment. In this study, the 3-D packing strategy consists of two layers. These are an agent strategy as_i that controls the packing operation of i-th agent and a supervisor strategy SS that sets a priority sequence of agents. Therefore, each agent searches an individual strategy as_i without interactions from other agents so that the packing performance of a corresponding container is maximized. The supervisor, then, tries to find the cooperative strategy SS that optimizes the whole packing performance. Figure 4.5 shows the outline of the proposed hierarchical tuning mechanism. The 3-D packing strategy PS in a multiagent environment is thus represented as follows;

$$PS = \{ AS, SS \}, \tag{4-2-39}$$

$$AS = \{ as_i ; i = 1 \sim N_C \}, \tag{4-2-40}$$

where, AS is a set of agent strategies.

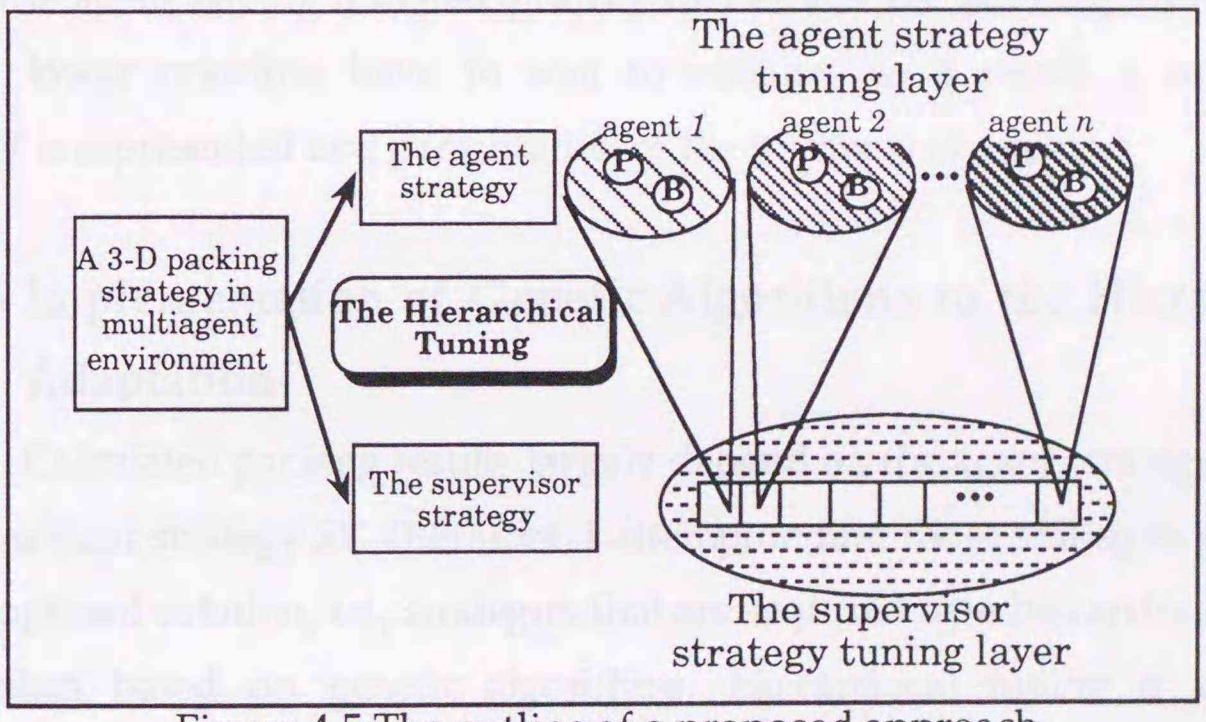


Figure 4.5 The outline of a proposed approach.

4.2.6.3 The Agent Packing Strategy

An agent packing strategy, as_i, is represented by two evaluation functions that are defined in section 4.2.4. These functions dominate heuristic packing procedures. That is,

- The evaluation function to select an allocation position.
- The evaluation function to select a box.

Moreover, the allocation space in each container is altered during the execution of the packing procedures in the same manner of single container environments.

4.2.6.4 The Supervisor Strategy

After each agent strategy pertaining to a corresponding container is acquired, a supervisor strategy should be also searched. A supervisor strategy is represented as a priority among agents. Since each agent acquires its individual packing strategy without interactions from other agents, a conflict of which a given box should be allocated in which container is caused. Conflicts among agents are solved by the supervisor strategy *SS*. Thus, each agent performs the packing of the container in the order of the priority sequence. While the agent having a higher priority carries out the packing, other agents having lower priorities have to wait to execute. As a result, a supervisor strategy is represented as a permutation of the numbers of agents.

4.2.6.5 Implementation of Genetic Algorithms to the Hierarchical Adaptation

Calculated packing results largely depend on the agent strategy *AS* and the supervisor strategy *SS*. Therefore, I attempt to find those strategies that give a near-optimal solution, i.e., strategies that are acquired by a hierarchical tuning mechanism based on genetic algorithms. Hierarchical tuning is achieved through an agent strategy tuning layer and a supervisor strategy tuning layer. In the first step, each agent acquires the individual packing strategy concerning the corresponding container without the interactions for other agent's strategies.

In the next step, a cooperative supervisor strategy is searched for a priority sequence of agents. Using this mechanism, packing performance is improved gradually, and a near-optimal solution can be obtained.

The Agent Strategy Tuning

A string as_i^j is expressed as a combination of weighted coefficients which consist of a sequence of the above six evaluation items, that is,

$$as_i^j = \text{e1 e2 e3 a1 a2 a3} \tag{4-2-41}$$

where as_i^j is a j-th string of the i-th agent. The set of the strings in the initial population of the agent i, $as_i(0)$, is generated randomly. This population evolves repeatedly with each generation, and a population $as_i(t)$ at generation t is generated as,

$$as_i(t) = \{ as_i^j ; j = 1\sim\text{pop_size} \}, \tag{4-2-42}$$

A reproduction operation is carried out according to each string's selection possibilities that are calculated using the capacity rate r_i of container i, as follows;

$$r_i = \frac{\sum_{k=1}^{N_b} f_{ik} \cdot v_k}{V_i} \tag{4-2-43}$$

where f_{ik} is the number of allocated boxes of k-th type in the container i, v_k is a volume of the k-th type box. In the current generation t, the packing operation of each string as_i^j is executed, and the corresponding capacity rate $r_i^j (j = 1\sim\text{pop_size})$ is calculated. Thus, the selection possibility $P_repro_i^j$ is given as follows:

$$P_repro_i^j = \frac{(r_i^j - r_worst_i)^2}{\sum_{j=1}^{pop_size} (r_i^j - r_worst_i)^2}, \tag{4-2-44}$$

where,

$$r_worst_i = \min_{j \in IS}[r_i^j] \tag{4-2-45}$$

and, *IS* is a set of the string's numbers. Using $P_repro_i^j$, the number of string reproductions as_i^j to the next generation *t+1* is given as

$$N_repro_i^j = P_repro_i^j * pop_size. \tag{4-2-46}$$

Each string is stored into a mating pool according to the $N_repro_i^j$ for further operations such as a simple two-point crossover and a conventional mutation.

The Supervisor Strategy Tuning

A supervisor strategy *SS* is represented as a priority among agents. Thus, a list of the numbers of agents is directly operated as a string to be tuned. So, a supervisor strategy SS_i is expressed as the following equation by a permutation of integer values that show agents' numbers, and the string length is equal to the number of given containers N_C.

$$SS_i = \{ int \}^{Nc} \tag{4-2-47}$$

An initial population *SS*(0) of SS_i is also generated randomly, and a population *SS*(*t*) at generation *t* is expressed as,

$$SS(t) = \{ SS_i ; i = 1 \sim pop_size \}. \tag{4-2-48}$$

To implement genetic algorithms with a supervisor strategy, a fitness value of a string is evaluated by the whole packing result $\boldsymbol{R}$ that is calculated based on the priority sequence of the string. Please note that simple genetic algorithms cannot be applied to such an order-based string. Therefore a supervisor strategy is also tuned by modified genetic operators for order-based strings, i.e., order-based uniform crossover and order-based mutation.

A reproduction operation is carried out in the same manner as agent strategy tuning. This is called roulette wheel reproduction. Each string's selection possibility is given as follows:

$$P_repro_i = \frac{(R_i - R_{worst})^2}{\sum_{j=1}^{pop_size} (R_j - R_{worst})^2}, \tag{4-2-49}$$

where,

$$R_{worst} = \min_{i \in IS}[R_i], \tag{4-2-50}$$

According to P_repro_i, the number of reproductions of string SS_i to the next generation $t+1$ is given as,

$$N_repro_i = P_repro_i * pop_size. \tag{4-2-51}$$

To generate more adapted strings, a crossover operation is applied to the reproduced strings. However, a conventional crossover operator can not be employed for strings that are encoded by order-based representations such as in this problem. Some research has been done on crossover operators for order-based strings, as in [Goldberg 89] and [Davis 91]. In this study, I use the uniform order-based crossover operator that is presented in [Davis 91] as the analog of general uniform crossover. This operator does for order-based strings what uniform crossover does for lists that preserves part of the first parent while it incorporates information from the second parent. The effect of uniform order-based crossover is to combine the relative orderings of nodes on the two parent strings in the two children. The way it works is somewhat complicated. This operation consists of the following steps.

(1) Select the two strings (parent 1 and parent 2) to be mated randomly.
(2) Generate a bit string named template that is the same length as the parents.
(3) Fill in some of the positions on Child 1 by copying them from Parent 1 wherever the template contains a "1".
(4) Make a list of the elements from Parent 1 associated with a "0" in the template.
(5) Permute these elements so that they appear in the same order that they appear on Parent 2.
(6) Fill these permuted elements in the gaps on Child 1 in the order

generated in 4.

(7) To make Child 2, carry out a similar process.

A mutation operator also has an effect on the strings in the pool. In this order-based description, a traditional mutation can not be operated. Therefore I consider an order-based mutation operation in which two values of a selected string are swapped mutually.

4.2.7 Numerical Experiments

Based on the proposed methods described above, I carried out some numerical experiments on a constructed 3-D packing simulator. In this experiment, I firstly performed packing simulations in single container environments. Next, experimental results of the constructed 3-dimensional packing rule-base were achieved. Finally, I also accomplished numerical experiments in multiagent environments.

4.2.7.1 Simulation Results in Single Container Environments

Firstly, to examine the solution searching performance of this approach, numerical experiments of 3-D packing in single container environments are carried out. In these experiments, experimental parameters are settled as table 4.1. As table 4.1 shows, the string length of a chromosome is 24 bits, and hence, each weighted coefficient is encoded into a four-bit length sub-string. Table 4.2 shows the simulation results of a typical input data. Figure 4.6 shows the graphical output of the resultant configuration of data 8.

Figure 4.7 shows how packing performances grow during the evolution through three types of data. In this figure, the axis of abscissa is the generation, and the axis of ordinate represents the results of the best strings, that is, the maximum values of the capacity rate. These results indicate that the strings having higher performances are generated by the tuning mechanism.

To examine the dynamics of the population, the best, the average, and the worst packing results are shown in figure 4.8. This result indicates that the

strings having low fitness values are eliminated and population evolves repeatedly each generation.

Table 4.1 Experimental GA parameters

The length of a chromosome	24 bits
The population size	50
crossover rate	0.8
mutation rate	0.01

Table 4.2 Given experimental input data and those simulation results.

	Dimensions of a container (W, L, H)	The number of boxes N_b	Minimum (w_i, l_i, h_i)	Maximum (w_i, l_i, h_i)	Rate of capacity R	The number of packed boxes
data 1	(35, 40,30)	50	(2, 3, 4)	(19, 20,17)	92%	37
data 2	(35, 40,30)	50	(6, 7, 8)	(19, 20,17)	86%	28
data 3	(35, 40,30)	50	(5, 5, 5)	(14, 15,14)	84%	45
data 4	(205,240,120)	200	(21,22,23)	(39, 40,37)	82%	169
data 5	(45, 50,40)	120	(2, 3, 4)	(19, 20,17)	94%	60
data 6	(50, 50,40)	100	(5, 6, 7)	(19, 20,17)	87%	34
data 7	(45, 50,35)	100	(5, 5, 5)	(14, 15,14)	85%	92
data 8	(506,316,363)	3000	(68,29,30)	(195,105,260)	91%	53

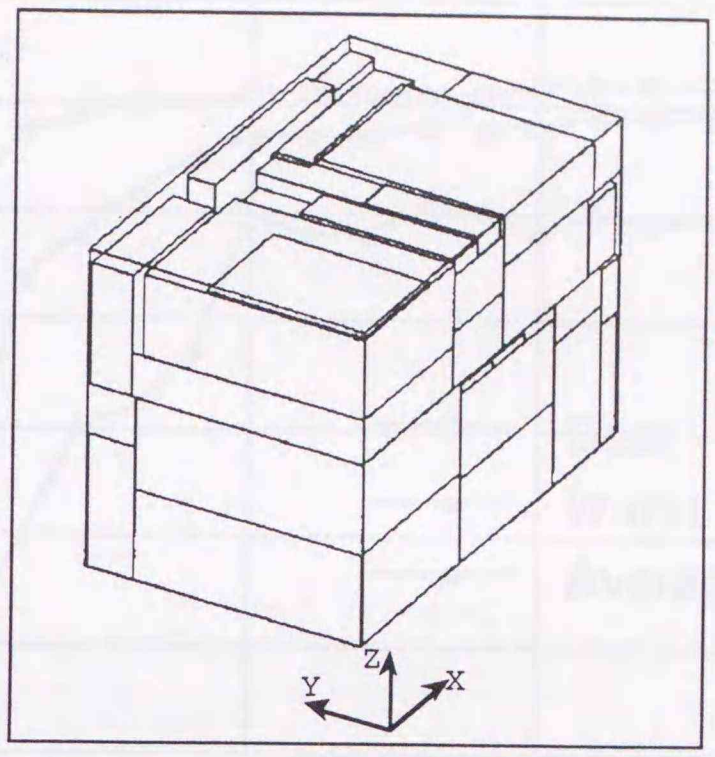


Figure 4.6 Graphical output of the resultant configuration of data 8.

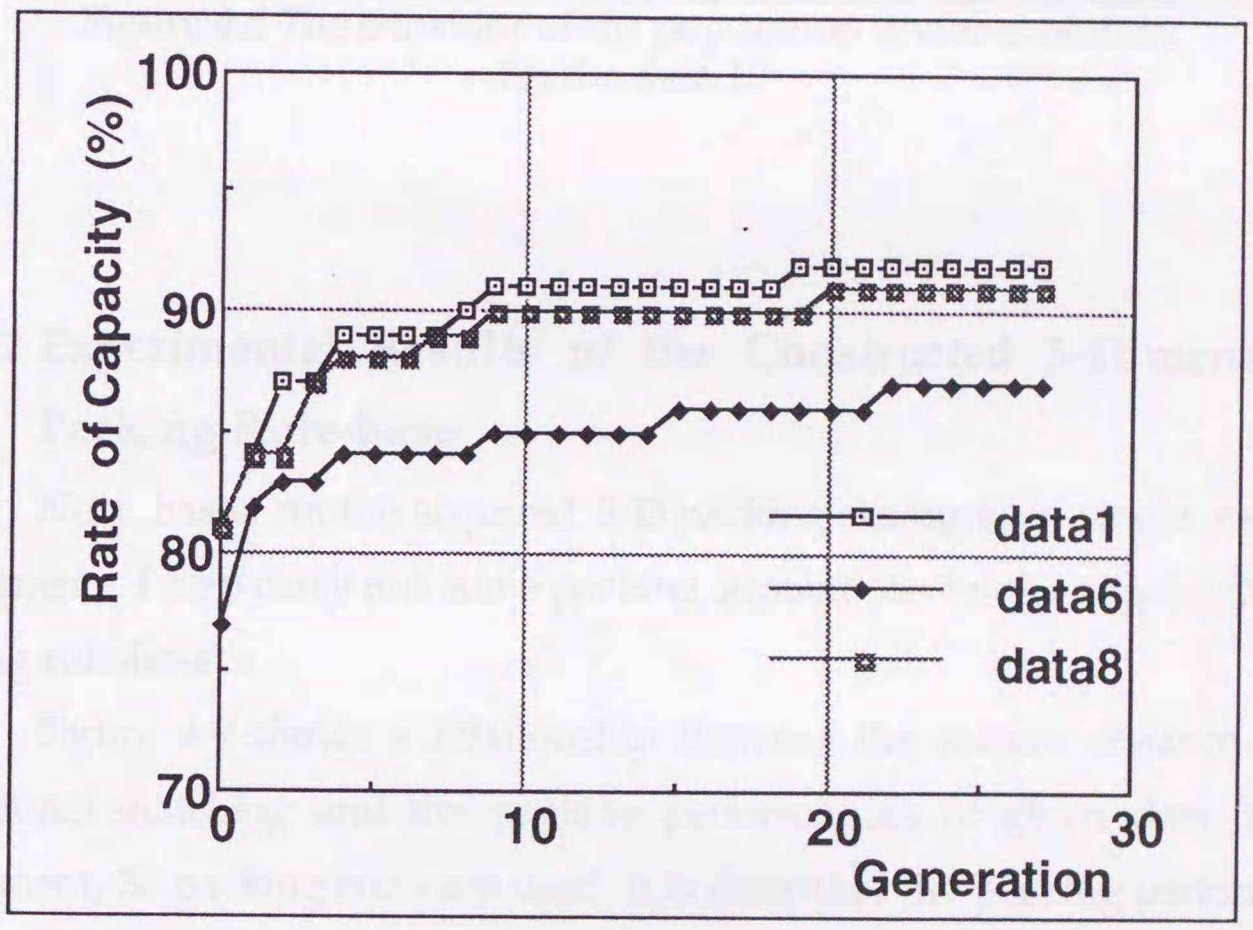


Figure 4.7 Growth of the packing performances for three typical input data.

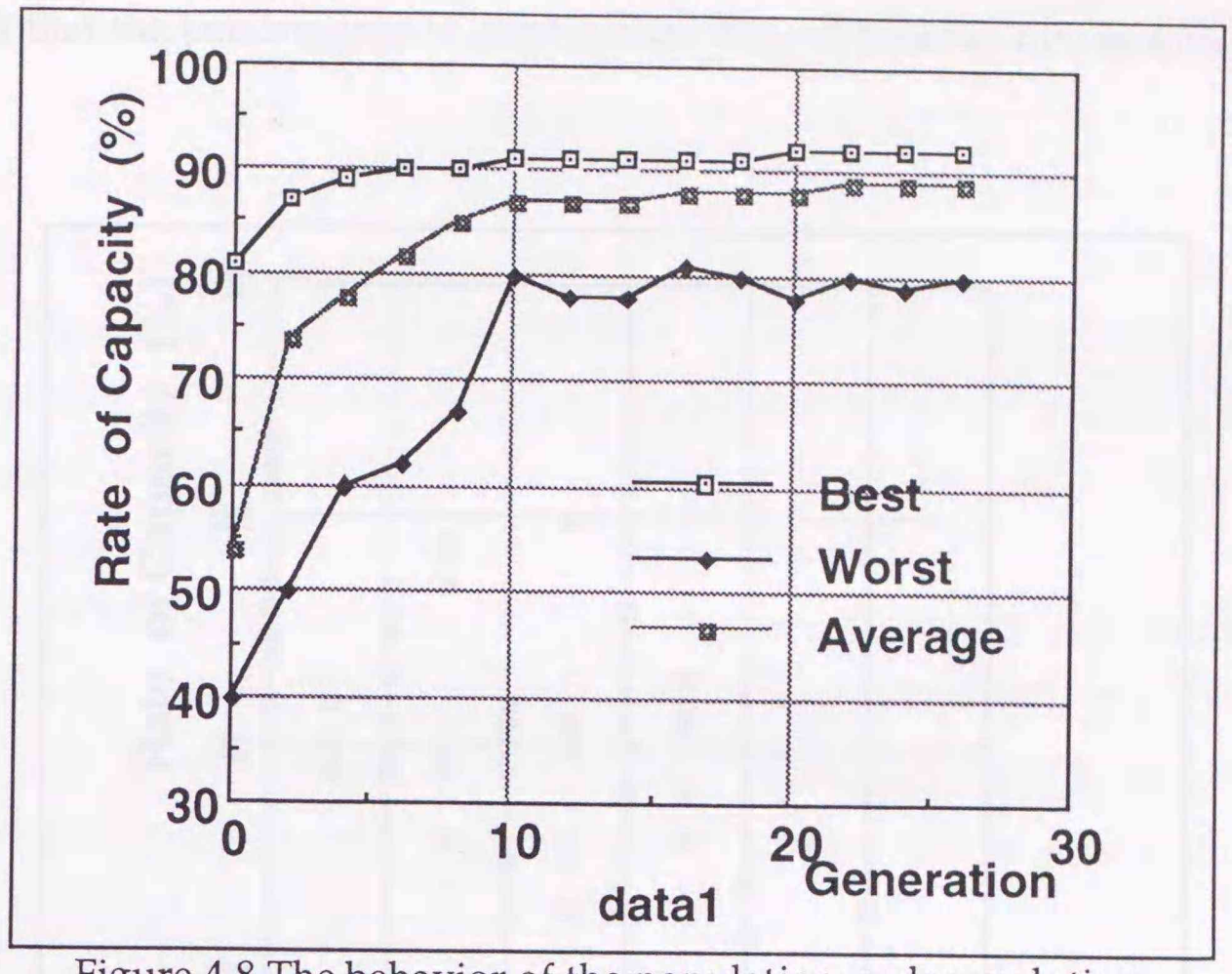


Figure 4.8 The behavior of the population under evolution for the data 1.

4.2.7.2 Experimental Results of the Constructed 3-Dimensional Packing Rule-base

Next, based on the acquired 3-D packing strategies in single container experiments, I also carry out some packing simulations by the constructed 3-D packing rule-base.

Figure 4.9 shows a relationship between the feature distance in the conditional matching and the packing performances of given data. In this experiment, 30 packing rules are used. It is clear that the packing performance becomes higher as the feature distance becomes smaller. The appropriate applicable range of the feature distance can be judged form this results.

On the basis of these results, I constructed a prototype of 3-D packing rule base. In this rule-base, the number of packing rules is set at 50, and the packing rule that the feature distance is less than 5 for each additional input data is applied as the packing strategy. Figure 4.10 illustrates the packing performances of the 3-D packing rule-base for additional 100 data. This result

shows that the performance is greater than 85% of capacity rate in 84% of 100 trials.

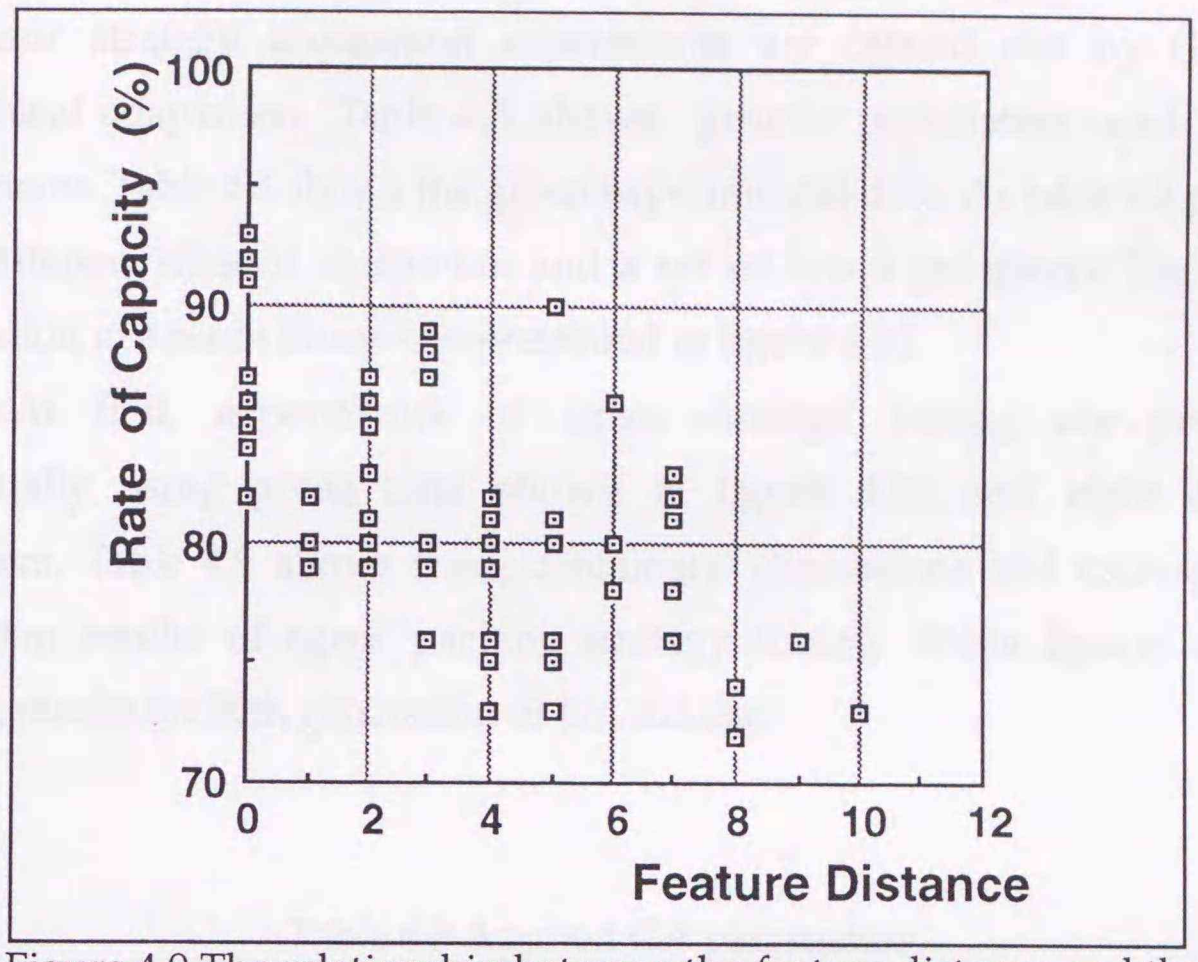


Figure 4.9 The relationship between the feature distance and the packing performances

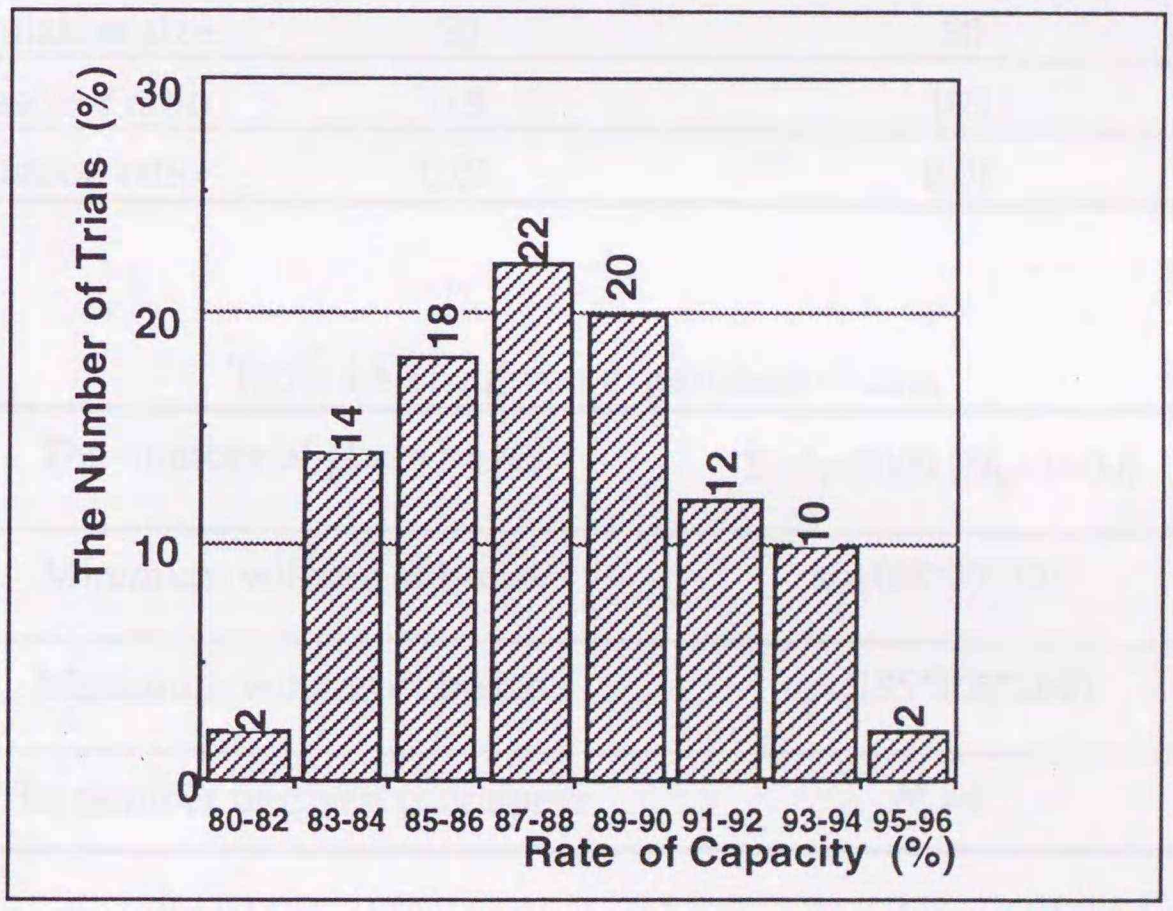


Figure 4.10 The packing performances of the constructed 3-D packing rule-base.

4.2.7.3 Numerical Experiments in Multiagent Environments

Finally, packing experiments in multiagent environments are carried out. In this experiment, the agent strategy acquisition experiments and the supervisor strategy acquisition experiments are carried out by GA-based hierarchical adaptation. Table 4.3 shows genetic parameters used in these experiments. Table 4.4 shows the given experimental data. As table 4.4 presents, eight different sizes of containers and a set of boxes are given. The volume distribution in a set of boxes is represented in figure 4.11.

At first, experiments of agent strategy tuning are performed individually using given data shown in figure 4.11 and eight different containers. Table 4.5 shows these containers' dimensions and corresponding simulation results of agent packing strategy tuning. These figures are best packing results on 30th generation of GA tuning.

Table 4.3 Applied GA parameters

	Agent strategy tuning	Supervisor strategy tuning
Population size	50	30
Crossover ratio	0.8	0.5
Mutation ratio	0.01	0.01

Table 4.4 The given experimental data.

The number of given boxes	$\Sigma\, nb_j$=3000 (N_b=1000)
Minimum volume of boxes	v_i=(68*29*30)
Maximum volume of boxes	v_i=(195*105*260)
The number of given containers	N_c=8

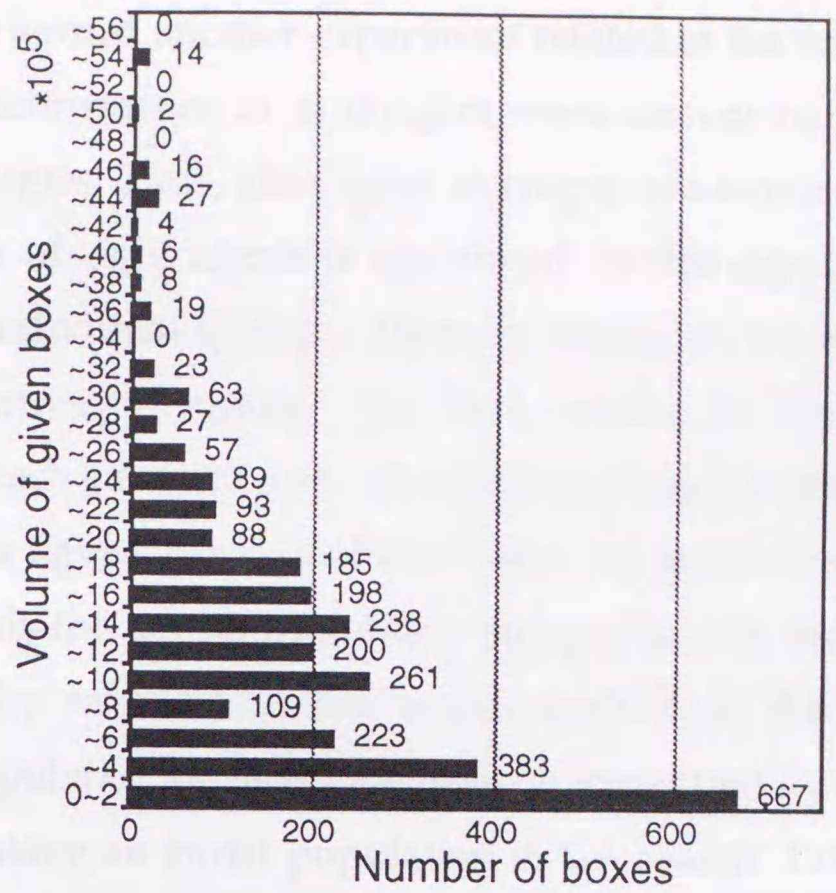


Figure 4.11 The volume distribution of given experimental boxes.

Table 4.5 Dimensions of given containers and individual tuned results.

Container No.	Dimensions of a container	Rate of capacity (%)	The number of packed boxes
1	(506,316,363)	91	53
2	(450,450,450)	87	79
3	(450,500,400)	92	82
4	(400,450,200)	82	35
5	(360,310,260)	86	29
6	(250,350,300)	84	33
7	(450,300,400)	85	46
8	(500,350,290)	90	51

I also performed another experiment related to the improvement of the whole packing performance in multiagent environment based on individual tuned agent strategies. Thus, after agent strategies are acquired individually, a priority sequence of eight agents is also tuned. In this experiment, each agent already has the individual strategy, these strategies are never changed during the supervisor strategy tuning. The best results of the whole packing performance on each generation of GA search and random search are shown in figure 4.12. In this figure, the axis of abscissa is the generation, and the axis of ordinate represents the results of the best strings, i.e., the maximum values of the whole capacity rate *R*. Random search method, in this case, means the method that a population of each generation is generated by the same manner as the way of making an initial population in GA search. Table 4.6 shows the best result of the whole packing performance and packing results of eight containers on the 15th generation of supervisor strategy tuning in multiagent environment. As these results indicate, whole packing performance in a multiagent environment can be improved by tuning the priority sequence. The random search also found a relatively good solution, because the number of given containers is a small value. When the number of agents is increasing, an appropriate packing solution cannot be searched.

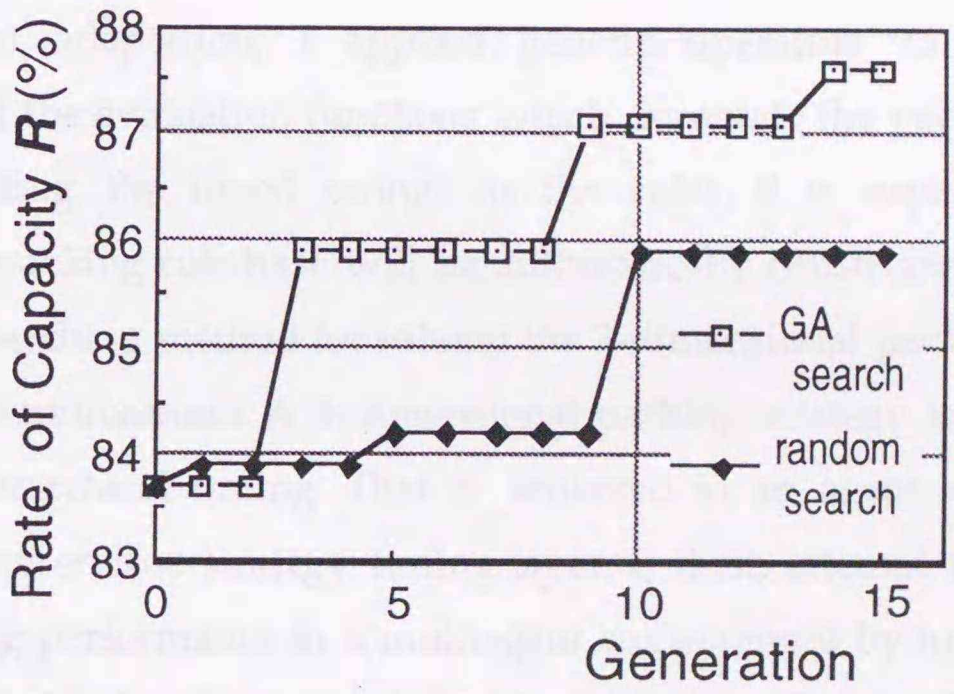


Figure 4.12 Simulation results of the supervisor strategy tuning by the GA search and a random search.

Table 4.6 Experimental results of supervisor strategy tuning in multiagent environment.

Container No.	Rate of capacity (%)	Tuned priority
1	91	3
2	86	2
3	92	1
4	80	7
5	85	6
6	84	8
7	86	5
8	89	4
Total	87.6	

4.2.8 Conclusions

Several conclusions may be drawn from this study. I proposed a searching method for solving the 3-dimensional packing problem using the genetic algorithms. To introduce automatic tuning mechanisms to environmental adaptation, I applied genetic operators to the weighted coefficients of the evaluation functions which dominate the packing strategies. By accumulating the tuned strings as the rules, it is expected that a 3-dimensional packing rule-base will be automatically constructed. Moreover, I proposed a searching method for solving the 3-dimensional packing problem in a multiagent environment. A 3-dimensional packing strategy is acquired by a GA-based hierarchical tuning. That is achieved in an agent strategy tuning layer and a supervisor strategy tuning layer. I, then, attempt to improve the whole packing performance in a multiagent environment by tuning a priority order of agents having the tuned individual strategy. That is also achieved by genetic algorithms that are modified for an order-based representation. Based

on the proposed approach, I constructed a 3-dimensional packing simulator. From some experimental results, high searching performance of genetic algorithms for this problem is shown.

References

[Axelrod 87] R.Axelrod, "The Evolution of Strategies in the Iterated Prisoner's Dilemma", *Genetic Algorithms and Simulated Annealing*, L.Davis(ed.),Pitman, pp.32-41,1987.

[Booker 87] L.Booker,"Improving Search in Genetic Algorithms", *Genetic Algorithms and Simulated Annealing*, L.Davis(ed.), Pitman, pp.74-88,1987.

[Davidor 91] Y. Davidor, "Genetic Algorithms and Robotics", World Scientific, pp.1-41, 1991.

[Davis 85] Davis, L., "Job Shop Scheduling with Genetic Algorithms," *Proceedings, ICGA'85*, L. Erlbaum, pp.136-140, 1985.

[Davis 91] Davis, L., "Handbook of Genetic Algorithms," VNR, pp.72-90, 1991.

[Davis and Steenstrup 87] L.Davis and M.Steenstrup,"Genetic Algorithms and Simulated Annealing:An Overview",*Genetic Algorithms and Simulated Annealing*,L.Davis(ed.), Pitman, pp.74-88,1987.

[Gabbert, et al. 91] P.S. Gabbert, D.E. Brown, C.L. Huntley, B.P. Markowicz, and D.E. Sappington, "A System for Learning Routes and Schedules with Genetic Algorithms", *Proc. of 4th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers*, pp.430-436, 1991.

[Goldberg 87] D.E.Goldberg, "Simple Genetic Algorithms and The Minimal, Deceptive Problem", *Genetic Algorithms and Simulated Annealing*, L.Davis(ed.), Pitman, pp.74-88,1987.

[Goldberg 89] D.E.Goldberg, "Genetic Algorithms in Search, Optimization & Machine Learning", Addison-Wesley, pp.1-25, 1989

[Goldberg and Lingle 85] Goldberg, D., and Lingle, R. : "Alleles, Loci, and the Traveling Salesman Problem, *Proceedings, ICGA'85*, L. Erlbaum, pp.154-159, 1985.

[Grefenstette 89] J.J. Grefenstette, "A System for Learning Control Strategies with Genetic Algorithms", *Proc. of 3rd International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp.183-190, 1987.

[Holland 75] J.H.Holland, "Adaptation in Natural and Artificial Systems", University of Michigan Press, pp.20-120, 1975.

[Holland, Holyoak, Nisbett, and Thagard 86] J. H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R.Thagard, "Induction", The MIT Press, pp.102-150, 1986.

[House and Dagli 92] House, R.L., and Dagli, C.H., "An approach to Three-dimensional packing using Genetic Algorithms," *Proceedings, ANNIE'92*, ASME-press, pp.937-942, 1992.

[House and Dagli 93] House, R.L., and Dagli, C.H., " Genetic Three Dimensional Packer," in Intelligent Engineering Systems Through Artificail Neural Networks (*ANNIE'93*), Vol. 3, ASME-press, pp.837-

842, 1993.

[Kawakami and Kakazu 94] Kawakami, T. and Kakazu,Y. : " Strategy Acquisition of the 3-D Packing Problem in Multiagent Environmment (GA-based Hierarchical Tuning)," Transactions of the Japan Society of Mechanical Engineers, Vol.60, No.577,C, pp.3219-3225, 1992. (in japanese)

[Kawakami et al. 91] Kawakami, T., et al., "Auto Tuning of 3-D Packing Rules using Genetic Algorithms," *Proceedings, IEEE IROS'91*, pp.1319-1324, 1991.

[Kawakami et al. 92] Kawakami, T., et al., "Automatic Tuning of 3-D Packing Strategy and Rule-Base Construction using GA," Transactions of Information Processing Society of Japan, Vol.33, No.6, pp.761-768, 1992.(in japanese)

[Kitano 93] Kitano,H. : "Genetic Algorithms", Sangyo-Tosho, pp.234-262, 1993. (in Japanese)

[Koakutsu, Sugai and Hirata 90] Koakutsu,S., Sugai,Y. and Hirata,H. : "Block Placement by Improve Simulated Annealing Based on Genetic Algorithm", *IEICE* (Institute of Electronics, Information and Communication Engineers of Japan) Transactions, Vol. J73-A, No.1, pp.87-94, 1990.

[Koza 92] Koza, J. R., "Genetic Programming", MIT press, pp.1-62, 1992.

[Mansour, and Fox 91] N. Mansour, and G.C. Fox, "A Hybrid Genetic Algorithm for Task Allocation in Multicomputers", *Proc. of 4th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers*, pp.466-473, 1991.

[Minagawa and Kakazu 90] M.Minagawa and Y.Kakazu "An Experimental Study on the Application of Genetic Algorithms to a Scheduling Problem", *Memoirs of the Faculty of Engineering*, Hokkaido University, Vol.152,1990.

[Minagawa, Kanazawa and Kakazu 89] M.Minagawa, R.Kanazawa and Y.Kakazu "Development of 3D Packing Simulator", *Memoirs of the Faculty of Engineering*, Hokkaido University, Vol.152, No.4, 1989.

[Poshyanonda and Dagli 93] Poshyanonda, P. and Dagli, C.H. : "Genetic Neuro-Nester for Irregular Patterns," in Intelligent Engineering Systems Through Artificail Neural Networks (*ANNIE'93*), Vol. 3, ASME-press, pp.825-830, 1993.

[Smith 85] Smith, D., "Bin packing with adaptive search," *Proceedings, ICGA'85*, L. Erlbaum, pp.202-206, 1985.

[Syswerda, and Palmucci 91] Syswerda, G., and Palmucci, J., : "The Application of Genetic Algorithms to Resource Scheduling", *Proc. of 4th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers*, pp.502-508, 1991.

[Whitley, Starkweather and Fuquay 87] D.Whitley, T.Starkweather and

D.Fuquay, "Scheduling Problems and Traveling Salesmen:The Genetic Edge Recombination Operator", *Proc. of 3rd International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp.133-140, 1987.

[Yamamura, Ono and Kobayashi 92] Yamamura,M., Ono,T. and Kobayashi,S. : "Character-Preserving Genetic Algorithms for Traveling Salesman Problem", Jour. of Japanese Society for Artificial Intelligence, vol.7, No.6, pp.117-127, 1992.(in Japanese)

4.3 The Autonomous Robot Navigator with Classifier Mechanisms

4.3.1 Introduction

When we consider the realization of fully automated and flexible manufacturing system or factory, we have to solve several difficult problems. One of the most difficult problems is the solution of the Robot Navigation task. In this task, an optimal solution of the path planning must be calculated in order to transfer the resources or parts effectively. For this purpose, a mobile robot is moving in the factory according to the instruction about movement of the robot from the navigation system, for example human supervisor, that determines an optimal path from starting position to destination position avoiding the unknown obstacles. When the mobile robot has such navigation system internally, this problem is called the autonomous mobile robot navigation problem. In particular, if there are multiple autonomous robots in the navigation area, then it is more difficult to solve the navigation task that involves the problem of collision avoidance among the robots, because it is difficult to predict the movement of other robots. Until today, there are many approaches to the problem, e.g., approaches from production rule-based one to fuzzy controlled one. However, all of these approaches adopted the predetermined rules or theories that are subject to the problem itself. Therefore, when multiple mobile robots are introduced and complicated navigation area is given, a new learning scheme that can solve the robot navigation problem autonomously is required.

Also, since the techniques of factory automation are developed and improved, the realization of the machine learning system with high performance is expected recently. This system must be able to dedicate the optimal solution by itself. To realize this machine learning system, we can usually use the Neural Networks that modeled information processing mechanisms of human brain. The other hand, Classifier System that is an

effective learning scheme from instances has been proposed, and the usefulness of classifier system is examined in simple experiments. The classifier system is a general purpose machine learning system that learns the simple production-like rules in the arbitrary environment. In this system, the rules that should be learned are called classifiers, and classifiers are used to induce the strategies of the system.

So, I attempt to solve the mobile robot navigation problem in an arbitrary environment automatically and flexibly. In order to realize an autonomous navigation system, I adopt a classifier mechanism. In this paper, the autonomous robot that individually has a classifier system as the navigation systems, is called an agent. Thus, there are multiple agents in a navigation area, and each agent induces an individual optimal solution for a given navigation task. In this paper, the navigation area is defined as a 2-dimensional maze space that is suitable for representation of factories or offices.

Based on the proposed method, the autonomous robot navigation simulator is constructed and some numerical experiments are carried out. First, single agent navigation tasks in which only one autonomous mobile robot is exist in the complex navigation area are performed. In these experiments, several useful results are obtained. That is, some aspects about the learning curves, the learning speed for different learning plans and the improvement of the learning performance by accumulating the knowledge of sub-problems are experimented. Second, the simulations of multiple agents' navigation tasks are carried out. In these experiments, multiple agents having individual tasks exist in the same navigation area and each agent moves synchronously according to an instruction of its autonomous navigation system. This navigation system must search the optimal path for a given task avoiding obstacles and the collision with other agents. Also, I assume that each agent has previously learned the individual task in single agent environment. Using these experimental results, I discuss several facts about the learning performances of the autonomous robots with classifier systems for the mobile robot navigation problem.

4.3.2 The Autonomous Robot Navigation Problem

In this section, I set the autonomous robot navigation problem that treated in this study. The autonomous mobile robot has an internal navigation system that indicates the near optimal path of the robot movement from a given origin position to a destination position using the leaned knowledge about navigation area. Thus, the solution of the robot navigation problem is obtained by the machine learning system without supervisor. To treat the robot navigation problem as a geometrical one, this navigation area is defined as a 2-dimensional maze space. That is, the navigation task is represented on the 2-dimensional X-Y absolute coordinate system of the environment. Furthermore, it isn't necessary that the navigation area is represented by a linear maze.

Here, the multiple autonomous robots navigation problem(MARNP) is represented by following five-tuple;

$$\text{MARNP} = < AR , OM , SP , DP, RP>, \tag{4-3-1}$$

and each items are defined as follows;

$$AR = \{ ar_i ; \text{i=1~}n_r \}, \tag{4-3-2}$$

$$OM = \{ om_{jk} ; j=1\sim \text{J} , k=1\sim \text{K}\}, \tag{4-3-3}$$

$$SP = \{ sp_i ; \text{i=1~}n_r \}, \tag{4-3-4}$$

$$sp_i = (x_{si} , y_{si}), \tag{4-3-5}$$

$$DP = \{ dp_i ; i=1\sim n_r \}, \tag{4-3-6}$$

$$dp_i = (x_{di} , y_{di}), \tag{4-3-7}$$

$$RP = \{rp_i^t ; i=1\sim n_r, t\in \text{T}\}, \tag{4-3-8}$$

$$rp_i^t = (x_i^t , y_i^t), \tag{4-3-9}$$

where, AR is a set of the autonomous mobile robots and ar_i is a i-th robot, OM is a 2-dimensional binary array describing the stationary obstacles in the navigation area(i.e. om_{jk}=1: obstacle, om_{jk}=0: free space), SP is a set of the coordinates of given starting positions, and DP is a set of the coordinates of given destination positions. Thus, a navigation task that should be solved by the i-th mobile robot is represented by sp_i and dp_i. A position of each mobile

robot at time t is represented by rp_i^t. In this area, there are multiple robots having individual navigation task concurrently, and each robot searches an optimal, i.e., shortest, path avoiding the obstacles and the collision with other robots. An example of the robot navigation problem is shown in the figure 4.13(in the case of single agent navigation task).

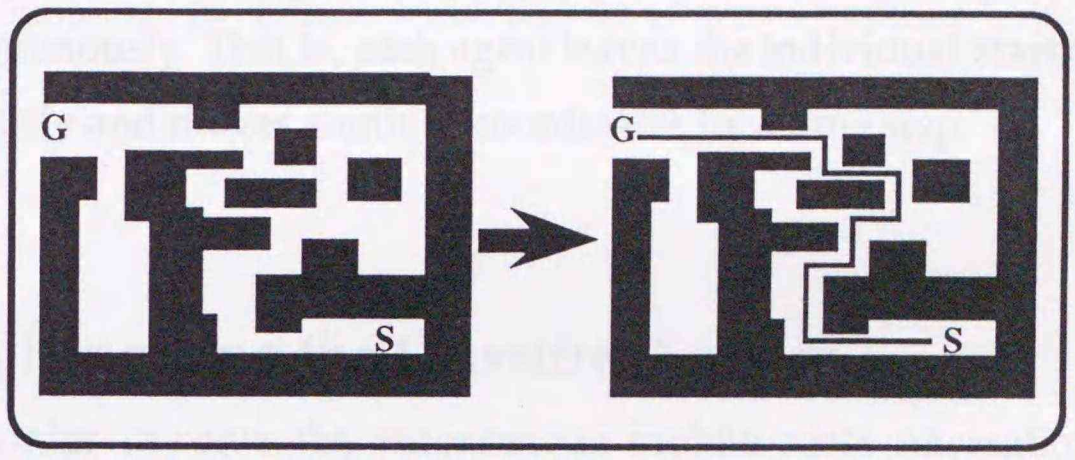


Figure 4.13 An autonomous robot searches the shortest path from starting position '**S**' to destination position '**G**'.

In this problem, a simulated autonomous mobile robot has the simple mechanisms to search the appropriate path. First, as a perceptive mechanism, the mobile robot can perceive the following facts:

- The coordinates of a current position, rp_i^t.
- The obstacles in the neighborhood of a current position, om_{jk}=1, where $j = x_i^t \pm 1$, and $k = y_i^t \pm 1$.
- The arrival at the destination position. This information is given by a judgment function, $f(rp_i^t)$. The function takes values according to the following equation.

$$f(rp_i^t) = \begin{cases} 1: \text{ if } rp_i^t = dp_i \\ 0: \text{ otherwise} \end{cases}. \qquad (4\text{-}3\text{-}10)$$

Therefore, each robot previously has no knowledge about the positions of obstacles and its destination position. The robot also perceives other robots as obstacles.

Next, as a mechanism of the robot action, the robot can move a unit of

coordinates by single time step according to an instruction from the navigation system. The directions of the robot movement are East, West, South and North. That is,

$$rp_i^{t+1} = rp_i^t \pm 1 = (x_i^t \pm 1, y_i^t \pm 1). \quad (4\text{-}3\text{-}11)$$

In addition, the robot can't predict the action of other robots. To simplify the interaction problem among each mobile robot, I set that multiple agents must move synchronously. That is, each agent leaves the individual starting position simultaneously and moves a unit of coordinates by a time step.

4.3.3 Implementing the Classifier System

In order to solve the autonomous mobile robot navigation problem mentioned above, the classifier system is implemented as a navigation system of a corresponding mobile robot. In the classifier systems and the production rule-based systems, a strategy is identically performed by the activated rules that satisfy the environmental conditions. But the rule syntax of the classifier systems is very different from that of the production rule-based systems. Thus, classifier systems depart from representational difficulties by restricting a rule to fixed length representation using alphabets or numbers. By this restriction, string rules can be operated easily. This is a main reason applying the classifier system to the autonomous robot navigation problem. In the multi-agent environment, each autonomous robot has such a classifier system individually.

In the case where the classifier system is applied to the robot navigation problem, a conditional part of a classifier, cf^c_j, corresponds to the coordinates of the robot, rp_i^t, and the time of current situation, t, and a direction of movement of the robot is indicated by an action part of the classifier, cf^a_j. For example, thus, a navigation rule is represented as follows:

if <current position is (x,y) and current time is t>

then < go *west*, i.e., $(x\text{-}1,y)$ >.

However, in the single agent navigation task case, the time factor can be

disregarded, because all obstacles in the area are static but unknown. To achieve this reactive planning process, through the detector as a perceptive organ of the classifier system, information from the environment is encoded to the finite length message that has an appropriate form for processing in the system. Then, the conditional matching and the rule competition are carried out. A winner classifier of the rule competition instructs a motion direction of the agent according to its action part. This determined strategy is performed through the effector where an action part is transformed to a linguistic formation.

Here, the learning plan by which a set of classifiers are evolved is presented. This evolutional mechanism is realized by the simple reinforcement algorithm. In this study, the environmental rewards are assigned to the selected classifiers based on the results of the instructed movements. Thus, the useful classifiers for the given task tend to have higher credit values, and the adaptive knowledge acquired from many instances. In fact, the credit assignment is performed as follows;

P1) *if* ($om_{jk} = 1 \mid j = x_i^{t+1}, k = y_i^{t+1}$) *then* $R < 0$ (4-3-12)

P2) *if* ($rp_i^{t+1} = rp_i^{\tau} \mid {}^{\exists}\tau \le t \in T$) *then* $R < 0$ (4-3-13)

P3) *if* ($rp_i^{t+1} \neq rp_i^{\tau} \mid {}^{\forall}\tau \le t \in T$) *then* $R > 0$ (4-3-14)

P4) *if* ($rp_i^{t+1} - rp_i^{t} = rp_i^{t} - rp_i^{t-1}$) *then* $R > 0$ (4-3-15)

P5) *if* ($rp_i^{t+1} = dp_i$) *then* $R > 0$ (4-3-16)

where R is an environmental reward. In these reinforcement plans, P1 means that *if* there is an obstacle on the indicated position, *then* give a negative reward, P2 means that *if* the indicated position is a previously passed position, *then* give a negative reward, P3 indicates that *if* the indicated position is an un-passed position, *then* give a positive reward, P4 is that *if* the indicated direction is straightforward for the last movement, *then* give a positive reward, and a mobile robot that attains the goal receives a positive reward by P5.

Since a current situation is made by the chain of the selected classifiers from a starting position, an environmental reward is distributed to later classifiers of the chain (figure 4.14). As a result of learning, one chain of appropriate classifiers that links from a starting position to a destination position is generalized.

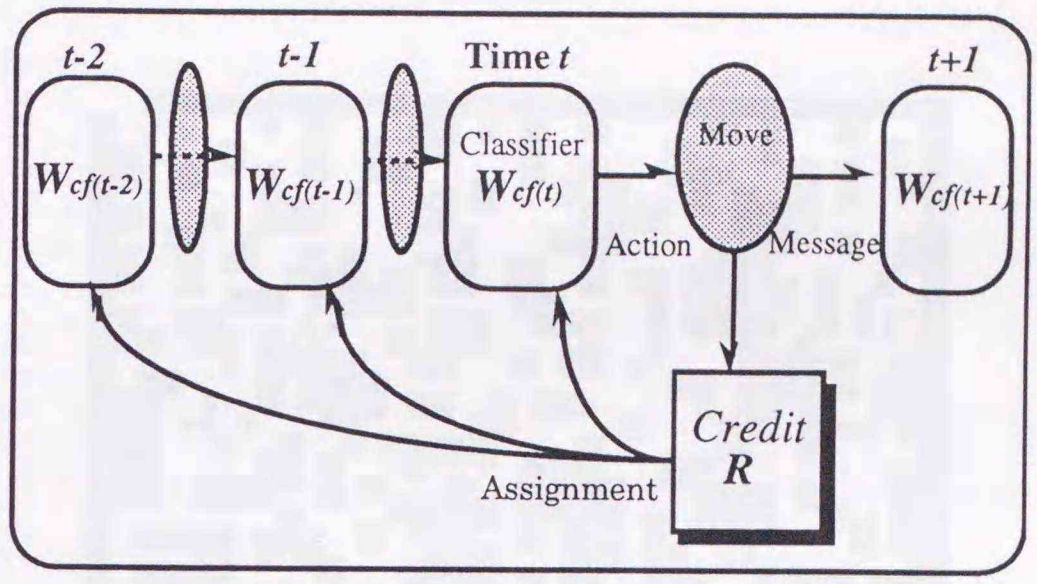


Figure 4.14 The credit distribution of environmental rewards.

4.3.4 Experimental Results

Based on the proposed method, the autonomous robot navigation simulator is constructed and some numerical experiments are carried out.

4.3.4.1 Experiments in Single Agent Domains

Firstly, experiments in the case where only single agent exists in a navigation area are carried out. In these experiments, given navigation tasks and a navigation area given as an environment are shown as figure 4.15. In this figure, the destination position is represented as a character 'G', and starting positions of given tasks are indicated by characters 'A'-'E'.

The number of classifier rules is fixed to 2000. The initial population of classifiers is generalized randomly, and every strength value of initial classifiers is set equivalently. The number of character '#' in a classifier is restricted to 10 percents of the length of a string.

To determine the effective reinforcement scheme, I performed primary

experiments using the task A in figure 4.15. In this experiment, the influence of the number of profit shared rules at a time for the learning performance is examined. Experimental result of this is shown in table 4.7. In this table, the length of searched path and the needed learning times to find the solution are figured. As the table presents, when an environmental reward R is assigned to three classifiers at each time, a good learning performance is demonstrated.

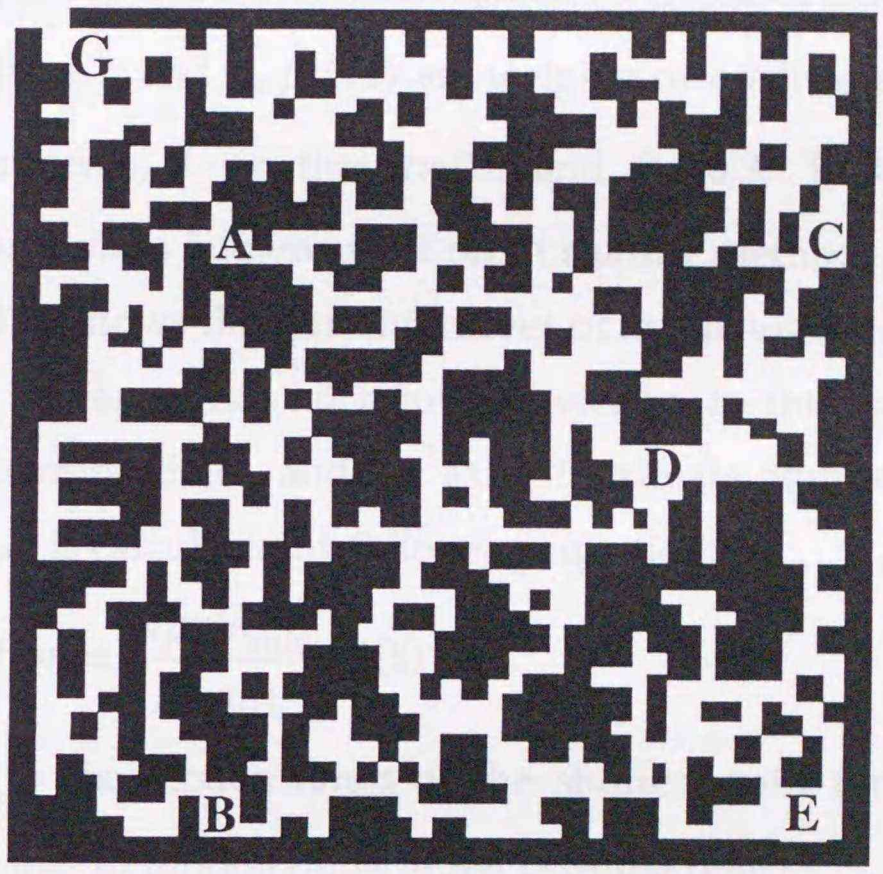


Figure 4.15 The given robot navigation tasks.

Table 4.7 The influence of the number of profit shared rules at a time for the learning performance.

The number of profit shared rules	Searched path length	Learning times
1 classifier	48	6
2 classifiers	28	9
3 classifiers	24	10
5 classifiers	24	36

As a rule reinforcement algorithm, therefore, the environmental reward R at time t is distributed to three classifiers that are later members of the searching chain. This distribution function is represented as follows;

$$S(Wcf(t)) = S(Wcf(t)) + \alpha R \tag{4-3-17}$$

$$S(Wcf(t\text{-}1)) = S(Wcf(t\text{-}1)) + \beta R \tag{4-3-18}$$

$$S(Wcf(t\text{-}2)) = S(Wcf(t\text{-}2)) + (1\text{-}\alpha\text{-}\beta)R \tag{4-3-19}$$

where, $S(Wcf(t))$ is the current strength value of a classifier that wins in the rule competition at time t, and α, β (<1) are weights of credit distribution. In the following experiments, I fix that α=0.6 and β =0.3. This type of credit assignment algorithms is known as the profit sharing method.

Figure 4.16 shows the learning curves of five navigation tasks described above in which the agent has no prior knowledge. In this figure, the axis of abscissa is the learning times, and the axis of ordinate represents the learning performances that is calculated by following equation;

$$Evaluation = \frac{Npath_{\min}}{Npath_i} \cdot 100 \tag{4-3-20}$$

where, $Npath_{\min}$ is the motion times of the shortest path for each task, and $Npath_i$ is the number of motion times at the learning time i.

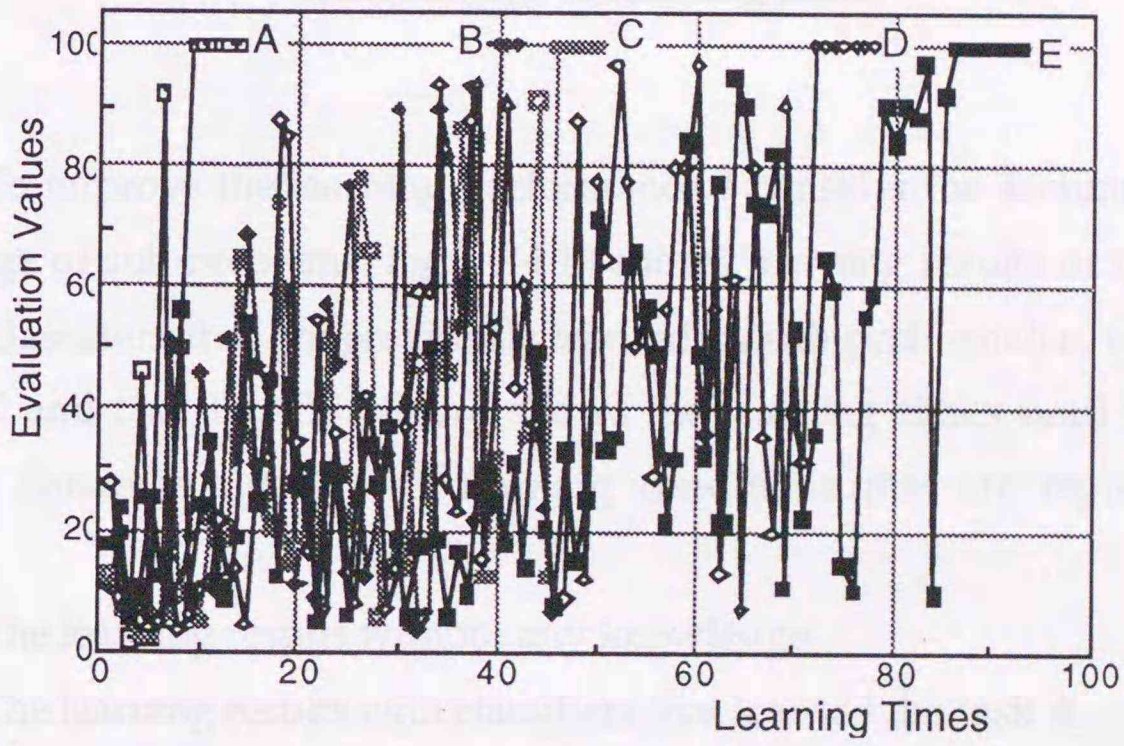


Figure 4.16 The learning curves of typical five navigation tasks.

To examine the learning times until solutions converge for different learning plans, the learning results by different combinations of learning plans are shown in figure 4.17. These learning plans are simple rules that are described in equations from (4-3-12) to (4-3-16). In this figure, the axis of ordinate represents the learning times until navigation solutions converge at optimal solutions, and the axis of abscissa represents the combinations of learning plans as the reinforcement algorithms; e.g., 2+5 means a combination of P2 and P5. These results indicate that the learning efficiency is varied based on the employed reinforcement algorithms.

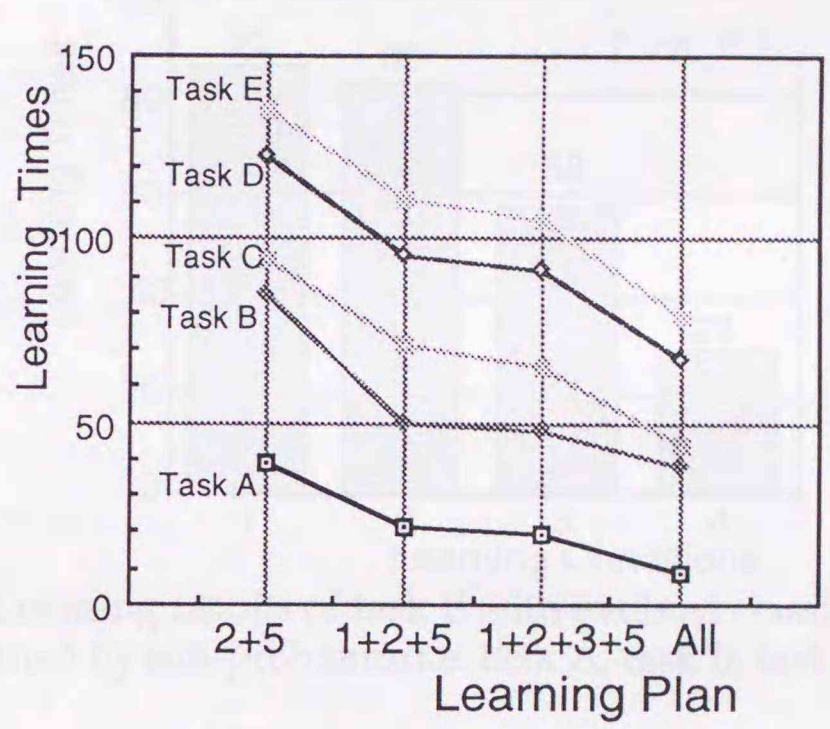


Figure 4.17 The learning results by different combinations of learning plans.

To improve the learning performance, I consider the accumulating the knowledge of sub-problems. Figure 4.18 shows learning results of task E with evolved classifiers that are previously learned by sub-problems(i.e. task A, task B, task C and task D). This figure shows the learning times until navigation solutions converge under four learning conditions that are represented as follows;

1: The learning results without any knowledge.

2: The learning results with classifiers that learned the task A.

3: The learning results with classifiers that learned the task A and the task D.

4: The learning results with classifiers that learned the task A, B, C and D. It is clear that the accumulating the knowledge of sub-problems facilitates to solve the larger problem.

Moreover, I examined the learning performance of learned classifier systems as a robot navigation rule-base. Figure 4.19 illustrates learning times to converge for randomly generated additional 50 tasks by using the learned classifier system that learned five tasks from A to E. From the result of this, it is expected to realize the robot navigation rule-base by learned classifier systems for some tasks.

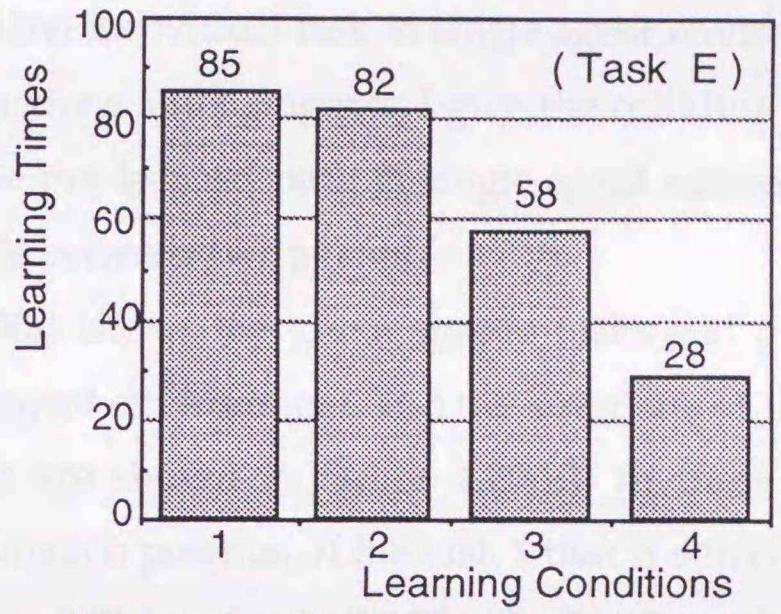


Figure 4.18 Learning results of task E with evolved classifiers that are previously learned by sub-problems(i.e. task A, task B, task C and task D).

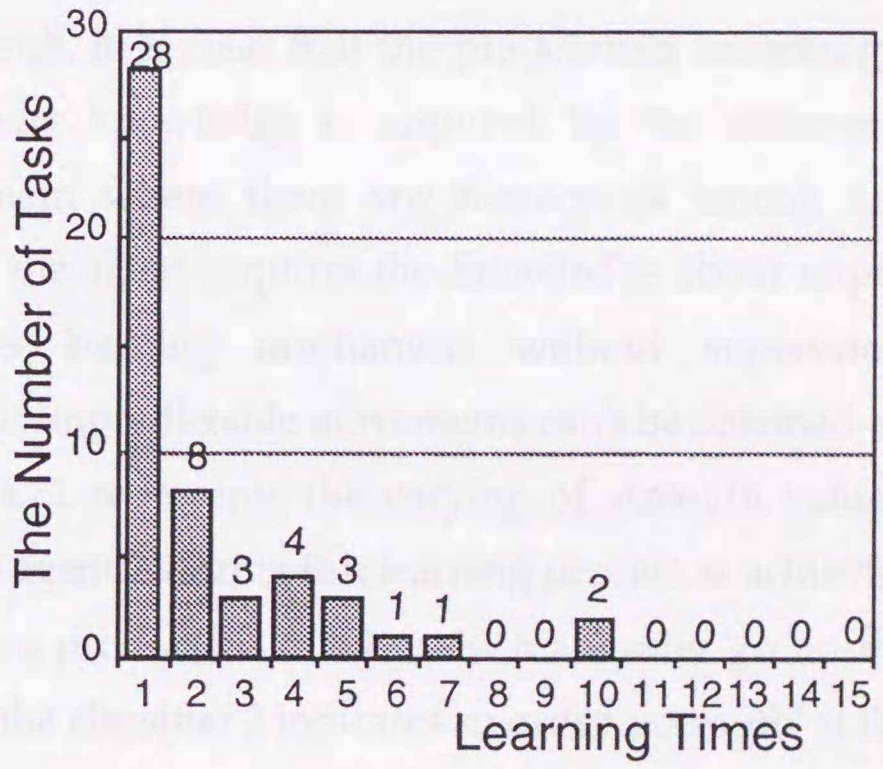


Figure 4.19 Learning times to converge for randomly generated additional 50 tasks by using the learned classifier system that learned five tasks from A to E.

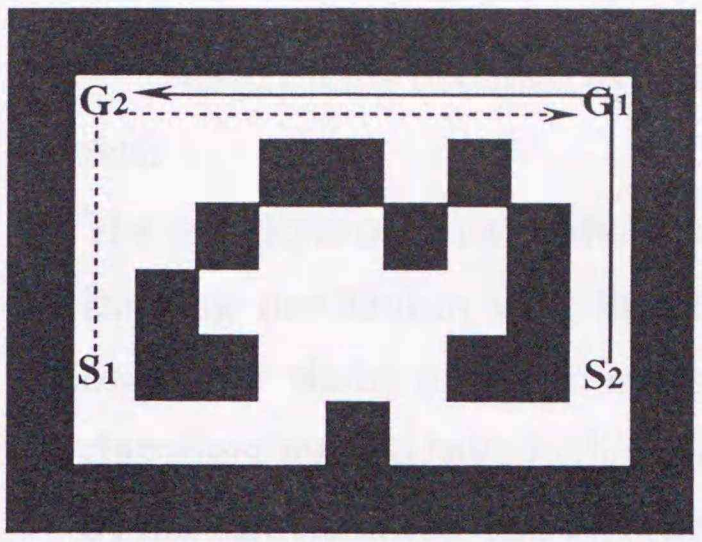
4.3.4.2 Experiments in Multiagent Domains

Secondly, I also carried out experiments of robot navigation problem in multiagent environment where multiple agents having individual tasks exist in the same navigation area and each agent moves synchronously according to an instruction of its autonomous navigation system. This navigation system must search the optimal path for a given task avoiding obstacles and the collision with other agents. To simplify this problem, I assume some restrictions; i.e., (1) the experiments are carried out in the two-agents domain, (2) these two agents move from individual starting positions alternately, and (3) each agent has previously learned the individual task in single agent environment. To examine the effects of interactions among agents, I give the colliding tasks where if each agent moves on the pre-learned path in single agent environment then it can't be escaped to collide between two agents.

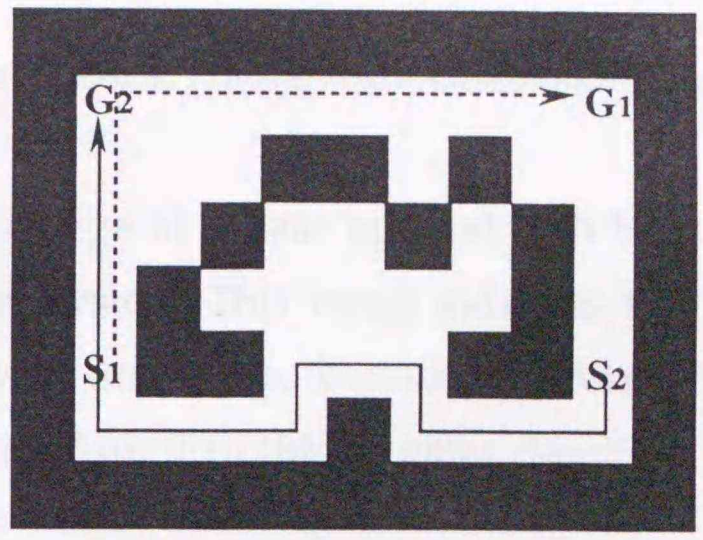
Figure 4.20(a) shows the given simple tasks and pre-learned paths of the tasks in single agent environment, and the solutions of those tasks in multi-agent environment are shown in figure 4.20(b). In these figures, a starting position and a destination position of the task 1 that is solved by the agent 1 are represented by 'S_1' and 'G_1' respectively. Identically, the task 2 is represented as 'S_2' and 'G_2'.

As a result, it is clear that the pre-learned knowledge of the agent is changed and new knowledge is acquired by the autonomous learning in multiagent domain where there are interactions among agents. It is very interesting that the agent acquires the knowledge about unpredictable objects by autonomous learning mechanism without supervisor. Because the knowledge about unpredictable movements can't be obtained easily.

Figure 4.21 represents the varying of strength values of two typical classifiers in the agent 2 during this learning process is achieving. In this figure, the classifier 1 is a production rule by which an order 'go north' is instructed at position S_2, and the classifier 2 indicates an order 'go south' at the same position, S_2.



(a) The given tasks and the learned paths of robot movements in the single agent environment individually.



(b) The learned results of given tasks in multi-agent environment.

Figure 4.20 Experimental navigation results in a multiagent environment.

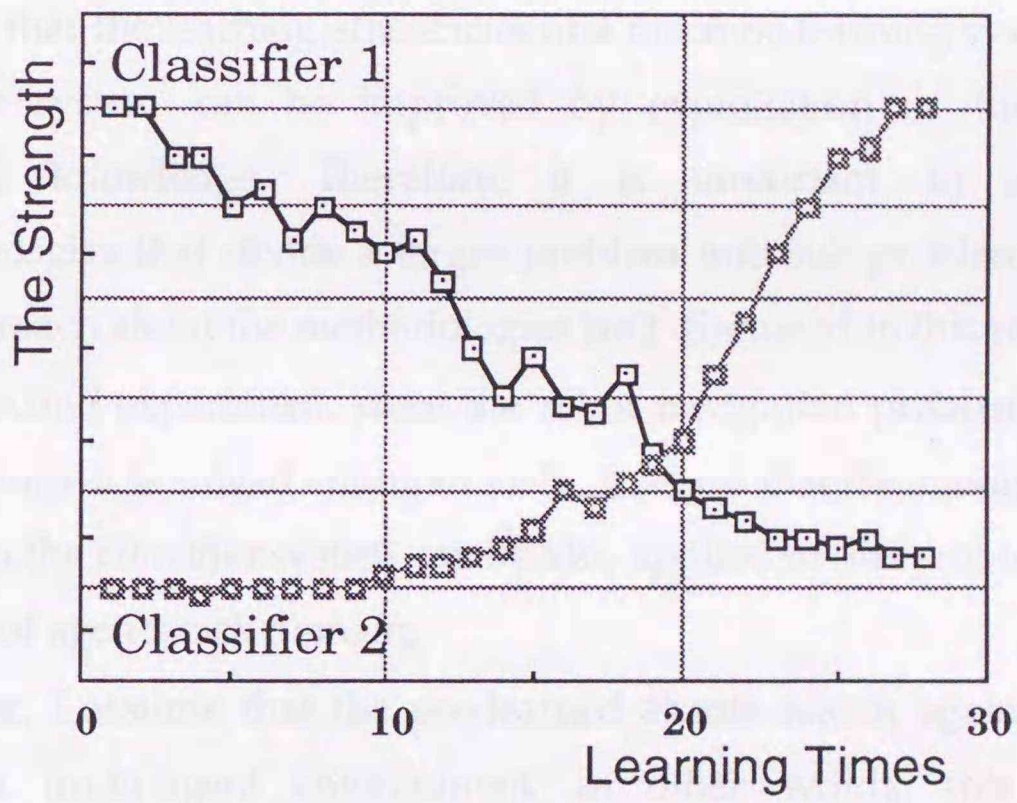


Figure 4.21 The varying behavior of strength values of two typical classifiers in the agent 2 during this learning process is achieving.
The classifier 1 is a production rule by which an order 'go north' is instructed at position S_2, and the classifier 2 indicates an order 'go south' at the same position, S_2.

4.3.5 Discussions

Based on the obtained experimental results, I make some discussions as follows;

- The solutions of a navigation task converge at a near optimal path by a learning mechanism with the classifier system. This result indicates that only one chain of robot movements is made up, because the useful classifiers tend to have higher strength values than that of other classifiers by the rule reinforcement algorithms.
- The solutions of a navigation task vibrate in the earlier period of learning. It seems that the agent searches extensively in the navigation area in order to diminish the unknown spaces.
- Since the high learning performances above are obtained for employed simple learning plans, the strong learning ability of the classifier system is examined.
- It seems that the learning efficiencies of a machine learning system with the classifier system can be improved by exploitation of the previously obtained knowledge. Therefore, it is important to develop the methodologies that divide a larger problem into sub-problems. However, any approach about the methodologies isn't discussed in this paper.
- In the second experiment, since the robot navigation problem in the two-agents domain is solved autonomously, it seems that the navigation system based on the classifier system can be also applied to this problem in a large number of agents task domain.
- Moreover, I assume that the pre-learned agents search again the optimal paths in multi-agent environment. In other words, this experiment indicates the environmental changes for previously learned agents. Therefore, the classifier system has a strong ability of adaptation to unpredictable changes of the environment.

4.3.6 Conclusions

A new approach of how to solve the autonomous robot navigation problem in multi-agent environment is presented in section 4.3. I applied the machine learning mechanism with the classifier system to realize an autonomous and self-learning navigation system of the agent, and I examined the higher learning performances based on the achieved numerical experiments. The advantages of applying the classifier system are summarized as three points:(1) it is easy to implement to robot navigation problems because of using the production-like rules, (2) since the rules called classifiers have a very simple rule syntax, it is easy to evaluate and generalize the rules, and (3) the learning performances can be improved by accumulating the knowledge. Thus, it may be possible to construct a knowledge base in the classifier system by accumulating the successful strategies. Moreover, in multi-agent environment, the agent acquires the knowledge about unpredictable objects by autonomous learning mechanism without supervisor. Thus it is clear that the machine learning system with classifier system can adapt itself autonomously for also unpredictable changes of the environment.

References

[Booker 87] L.Booker, :Improving Search in Genetic Algorithms, in *Genetic Algorithms and Simulated Annealing*, L.Davis(ed.), Pitman, pp.74-88,1987.

[Davidor 91] Y. Davidor, :A Naturally Occurring Niche & Species Phenomenon : The Model and First Results, in *Proc. of 4th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers*, pp.303-310, 1991.

[Davis and Steenstrup 87] L.Davis and M.Steenstrup, :Genetic Algorithms and Simulated Annealing:An Overview, in *Genetic Algorithms and Simulated Annealing*, L.Davis(ed.), Pitman, pp.74-88,1987.

[Goldberg 89] D.E.Goldberg, :Genetic Algorithms in Search, Optimization & Machine Learning, Addison-Wesley, pp.1-25, 1989

[Grefenstette 89] J.J. Grefenstette, :A System for Learning Control Strategies with Genetic Algorithms, in *Proc. of 3rd International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp.183-190, 1987.

[Grefenstette 91] J.J. Grefenstette, :Lamarckian Learning in Multi-agent Environment, in *Proc. of 4th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pp.303-310, 1991.

[Holland 75] J.H.Holland, :Adaptation in Natural and Artificial Systems, University of Michigan Press, pp.20-120, 1975.

[Holland,Holyoak,Nisbett and Thagard 86] J. H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R.Thagard, : Induction, The MIT Press, pp.102-150, 1986.

[Kawakami and Kakazu 92] Kawakami,T. and Kakazu,Y. : "The Autonomous Robot navigator with the Classifier Mechanism", Proceedings of First Japan-France Congress of Mechatronics, pp.163-168, 1992.

[Kawakami and Kakazu 93] Kawakami,T. and Kakazu,Y. :"A Study on an Autonomous Robot navigation Problem using a Classifier System", Transactions of the Japan Society of Mechanical Engineers, Vol.59, No.564,C, pp.2339-2345, 1993. (in japanese)

[Kawakami, Minagawa and Kakazu 91] T. Kawakami, M.Minagawa, and Y. Kakazu, :Auto Tuning of 3-D Packing Rules Using Genetic Algorithms, in *proc. of IEEE International Workshop on Intelligent Robots and Systems '91*, 1991.

[Tan 91] M. Tan, : Learning a Cost-Sensitive Internal Representation for Reinforcement Learning, in *Proc. of 8th International Conference on Machine Learning*, Morgan Kaufmann Publishers, pp.358-362,1991.

[Wilson and Goldberg 89] S.W. Wilson and D.E. Goldberg, : A Critical Review of Classifier Systems, in *Proc. of 3rd International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp.244-255,1989.

[Zhou and Grefenstette 89] H.H. Zhou and J.J. Grefenstette,:Learning by Analogy in Genetic Classifier Systems, in *Proc. of 3rd International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp.291-297,1989.

4.4 An Approach to Sequencing Problems in Process Planning

4.4.1 Introduction

In this application, the new approach to process planning problems is proposed. It is well known that how to get the solution of the Process Planning is said to be the one of the most difficult problems. In general a process planning task for a designed machine part involves generating a set of plans that outline operations, machine tools, fixtures, and tools required to produce that part as a machine component. On the basis of the design specifications provided by the design engineer, process planner has to determine the process plan for the designed part under consideration of minimizing a production cost, and at the same time maximizing rate of production and quality of a part. To get the good process plan requires the service of an expert process planner who has a skillful knowledge about machining processes, machine capabilities, and so on. However this kind of expert is going to disappear. So in the last decade, there has been a trend to develop an automated process planning system, since it is expected to play a role of just like an expert process planner. As a result many attempts have been made, e.g., from a practical variant approach, to a theoretical variant approach and AI-based expert approaches. However, all of these approaches have required the conformation of predetermined rules or procedures these are subjected to the given problem itself. Therefore, when the design specifications are changed, a new re-planning is necessary. This fact means when re-planning happens, we have to determine the needed data with respect to the changed situation. This looks so tedious. Then, in this study I propose a newly process planning idea that will have the function of learning and autonomy based on Genetic Algorithms.

Generally speaking, the task of the process planning is decomposed into several phases [Kusiak 90]. Due to the diversity of the process planning task, however, it is difficult to apply a uniform approach to all of the phases.

Some of the phases are basically performed by a knowledge-based subsystem. However, it is very difficult to solve the sequencing problem of machinable volumes since the optimal sequence of operations must be selected from many combinations of operations that satisfying the process constraints. Furthermore, the phase of sequencing of machinable volumes largely affects the processing time. Hence, solving this phase is one of the most important problems for developing the autonomous process planning system. So, I attempt to automatically and flexibly solve the sequencing problem of machinable volumes as a sequence of operations that enable to produce a machine part under machining constraints. In order to realize an autonomous planning mechanism, I perform the two approaches for this problem that is treated as a combinatorial optimization problem. First, the genetic algorithms are applied directly. As the second approach, the classifier system is applied to this problem to obtain the further advantages.

First, the genetic algorithms are applied directly. To implement the genetic algorithms, an order of removing operations is encoded as a string. Thus, a process that produces a finished part from a raw material by removing the machinable volumes is represented as a sequence of the numbers of machinable volumes. In this process, the volume removal costs, required tools and fixtures of each machinable volume are given. Also, tool setup costs and fixture setup costs are fixed. A fitness value of a string is calculated based on the sum of costs that are indicated by decoding the string. Using the genetic algorithms, an initial population of strings is evolved repeatedly, then a string having the minimum production cost is obtained.

As the second approach, the classifier system is applied to this problem to obtain the further advantages. In this approach, a classifier corresponds to a production rule that instructs the next removal volume for a current machining state. As a result of the learning, a near optimal process plan is represented as a chain of classifiers with higher strength values. A rule discovery mechanism is one of the important elements in machine learning system because all of feasible rules can not be searched. In this system, since the representation of rules is very simple and in fixed-length form, it is easy to generate a new rule by using genetic algorithms mentioned above. Based on the proposed

approaches, a genetic based process planning simulator is constructed and some numerical experiments are carried out. First, some process planning tasks are performed by the genetic algorithms approach. Second, the same tasks are solved by applying the classifier system. In these experiments, several useful results are obtained.

4.4.2 Problem Setting

In what follows, I address the primary problem treated in this approach. That is, the sequencing problem of a process planning task. The task of the process planning is decomposed into following phases[Kusiak 90];

- Volume decomposition of a designed part,
- Selection of alternative machines, tools and fixtures,
- Machining optimization,
- Decomposition of machinable volumes,
- Selection of machinable volumes,
- Generation of precedence constraints,
- Sequencing of machinable volumes.

Some of these phases are basically performed by a knowledge-based subsystem. However, it is very difficult to solve the sequencing problem of machinable volumes since the optimal sequence of operations must be selected from many combinations of operations that satisfying the process constraints. Furthermore, this phase largely affects the processing time. Therefor, solving this phase is one of the most important problems for developing the autonomous process planning system. In this approach, rather than considering the representation of a part to be designed with geometric features, the representation with machinable volumes is considered. Absolutely, each representation of a machinable volume has corresponding geometric features. An advantage is taken of alternative representation of the volume to be removed. Consider a part P presented in figure 4.22. The volume Q of material had to be removed from stock to obtain the part P. This volume V can be

segmented into machinable volumes in many ways. The segmentation process consists of following procedures;

(1) The volume Q is decomposed into some elementary sub-volumes v_i (i=1,..,n) shown as figure 4.22, and following equations are satisfied.

$$\bigcup_{i=1}^{n} v_i = Q, \tag{4-4-1}$$

$$\bigcap_{i=1}^{n} v_i = \phi_v, \tag{4-4-2}$$

where, ϕ_v is a geometrical void set.

(2) The selection of alternative machines, tools and fixtures is performed for each elementary volume.

(3)Some elementary volumes that can be machined concurrently by the same machining condition are combined into a machinable volume V_j (j=1,..,m).

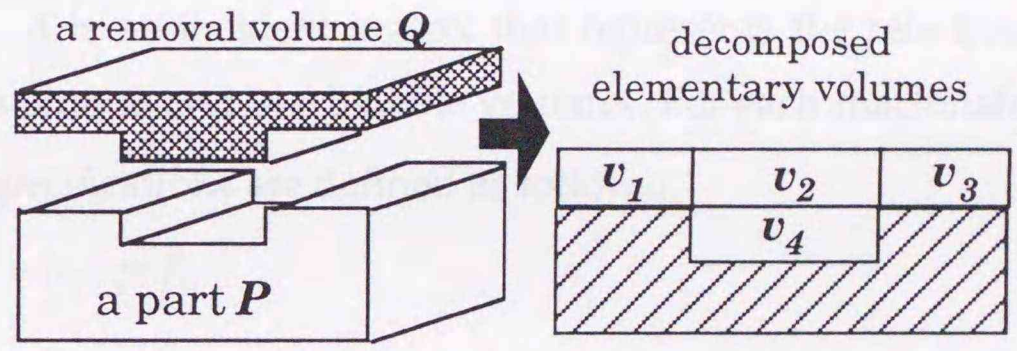


Figure 4.22 A part P and a decomposition of removal volume Q.

In the case of figure 4.22, for example, machinable volumes V_j are segmented from Q as follows;

$$V1 = \{v1, v2, v3\}, \tag{4-4-3}$$

$$V2 = \{v4\}, \tag{4-4-4}$$

$$V3 = \{v2, v4\}, \tag{4-4-5}$$

$$V4 = \{v1\}, \tag{4-4-6}$$

$$V5 = \{v3\}. \tag{4-4-7}$$

Each machinable volume may contain more than one elementary volume, and each elementary volume has to be included in at least one machinable volume. For convenience, these alternative segmentations are

represented by incidence matrix. Each alternative segmentation of volume Q may result in different volume removal cost, tool utilization cost and fixture utilization cost. Therefor, based on every machining constraint, appropriate machinable volumes must be selected and ordered for finishing the part P.

So I employ a following simple model to solve the sequencing problem of machinable volumes as a combinatorial optimization problem. In this problem, the optimal chain of machinable volumes is selected and sequenced from all given machinable volumes. The simple model is described as follows.

$$v = \{v_i \,;\, i \in I\}, \tag{4-4-8}$$

$$V = \{V_j \,;\, j \in J\}, \tag{4-4-9}$$

$$A = \{a_{ij} \,;\, i \in I, j \in J\} = \begin{cases} 1\text{: if } v_i \text{ corresponds to } V_j \\ 0\text{: otherwise} \end{cases} \tag{4-4-10}$$

where v is a set of elementary volumes, V is a set of machinable volumes, I and J are a set of the subscripts of elementary volumes and machinable volumes respectively, A is an incident matrix that represents the relationship between elementary volumes and machinable volumes. For each machinable volume V_j, machining specifications are defined as follows;

$$C = \{c_j \,;\, j \in J\}, \tag{4-4-11}$$

$$T = \{t_j \,;\, j \in J\}, \tag{4-4-12}$$

$$F = \{f_j \,;\, j \in J\}, \tag{4-4-13}$$

where, C is a set of removal cost of machinable volumes, T is a set of required tools for machining the V_j, and F is a set of required fixtures for machining the V_j. To calculate the total machining cost, I define other variables. Thus,

e_t : a tool changing cost,

e_f : a fixture changing cost,

n_t : the number of tool changing times,

n_f : the number of fixture changing times.

The objective function of this model minimizes the total sum of volume machining costs and utilization costs of tools and fixtures as follows.

$$Z = \min\left(\sum_{j \in J} c_j x_j + \sum_{t \in T} n_t e_t + \sum_{f \in F} n_f e_f\right), \tag{4-4-14}$$

where, x_j indicates that a machinable volume V_j is selected or not according to the following form.

$$X = \{x_j ; j \in J\} = \begin{cases} 1\text{: if } V_j \text{ is selected} \\ 0\text{: otherwise} \end{cases} . \qquad (4\text{-}4\text{-}15)$$

In this model, each elementary volume v_i has to be removed by selecting at least one machinable volume V_j in which v_i is included. That is ensured by the following equation.

$$\sum_{j \in J} a_{ij} x_j \geq 1, \text{ (for all } i \in I\text{)}. \qquad (4\text{-}4\text{-}16)$$

This problem can be regarded as a set covering problem described by equation (4-4-15) and (4-4-16). Generally, a planning problem is defined by a goal state to be achieved, a starting state from which the goal must be achieved, and a set of operators that can be used to achieve the goal. Consequently, I formulated the process planning problem as a sequence of operations and resources that enable to produce a finished part from the given initial state of material. Then a solution that minimizes total cost is searched. In a case of figure 4.22, for example, $V1$ and $V2$ are selected, then a removal sequence is made as ($V1 \Rightarrow V2$).

4.4.3 An Approach with Genetic Algorithms

Here, we attempt to realize the automatic planning mechanism by which the appropriate sequence of production operations is automatically searched and good planning solution is obtained. As a first approach to realize this mechanism, I applied genetic algorithms [Holland 75] [Goldberg 89] [Davis and Steenstrup 87] which mimic the process of natural evolution system.

4.4.3.1 Representations

A production cost largely depends on a sequence of machinable volumes for finishing a part. So, I attempt to find the permutation of removal volumes that gives a near-optimal production cost, i.e., I "evolve" the strings

that represent sequences of machinable volumes by implementing genetic algorithms. In this study, a string OP is directly represented as an order of numbers of machinable volumes V_j, e.g.,

$$\text{OP} = (2\,4\,1\,3\,5). \tag{4-4-17}$$

This string OP means a removing plan by which at first the machinable volume $V2$ is removed, then $V4$ is machined, and finally $V5$ is removed. Thus, a string has the length of values that is equal to the number of given machinable volumes V_j. In general, so, a string OP is represented as the following expression.

$$\text{OP} = \{int\}^{N_V}, \tag{4-4-18}$$

where N_V is the number of machinable volumes. As a set of these strings, a population $POP(t)$ at generation t is generated, that is,

$$POP(t) = \{\, OP_i \,;\, i = 1 \sim pop_size \,\}, \tag{4-4-19}$$

where *pop_size* is the number of strings in a population that does not vary in time. To automatically evolve the strings, genetic operators are applied over several generations.

Of course, in this way of representation, there may be a sequence by which the machining of a part can not be performed under physical constraints. Also a redundant representation may be formed. In this representation strategy of strings, each value is decoded into a corresponding machinable volume from a head of a string, and the decoded volume is removed. This decoding process terminates when a part is finished. The performance of each string is evaluated by summing each production cost until a part is finished. When the decoded machinable volumes can not be removed in the indicated order of a string under physical constraints, the total production cost of the string is calculated as an infinite value.

4.4.3.2 A Reproduction Operator

Reproduction operator takes the role of alternating the strings to next generation. This is carried out according to the selection possibility of each of the strings, P_sel_i, that is calculated using the sum of production costs PC_i for a string OP_i in the current generation t. Thus, the selection possibility $P_\ sel_i$ is

given as follows:

$$P_sel_i = \frac{pc_i}{\sum_{j=1}^{pop_size} pc_j} \quad , \tag{4-4-20}$$

where,

$$pc_i = (PC_i - PC_{worst})^2 \, , \tag{4-4-21}$$

$$PC_{worst} = \min_{i \in IS} [PC_i] \, , \tag{4-4-22}$$

and, *IS* is a set of the string's numbers. According to P_sel_i, the number of reproduction of string OP_i to the next generation $t+1$ is given as

$$N_repro_i = P_sel_i * pop_size \, . \tag{4-4-23}$$

Each string is stored into the mating pool according to the N_repro_i for the further operations.

4.4.3.3 Further Genetic Operators

To generate more adapted strings, the crossover operation is applied to reproduced strings. However, a conventional crossover operator can not be employed for strings that are encoded by order-based representation such as this problem. Some research has been done on crossover operators for order-based strings, as in [Goldberg 89] and [Davis 91]. In this study, I use the uniform order-based crossover operator that is presented in [Davis 91]. This operator is already described in the section 4.2 of this thesis. Here, the number of strings to which this operation is applied is defined to be 50% of *pop_size*.

A Mutation operator is also applied to reproduced chromosomes. In this order-based description, a traditional mutation can not be operated. Therefore I consider an order-based mutation operation in which two values of a selected string are swapped mutually. I set up that one mutation will occur during the transferring of 1000 values.

4.4.4 An Approach Using Classifier Systems

As the second approach to solve the process planning problem mentioned above automatically, the classifier system is implemented as an

autonomous planner that selects machinable volumes and makes an optimal sequence of those. The purposes of using a classifier system are as follows; (1) a realization of autonomous learning mechanisms, and (2) improving the learning performances by accumulation and practical use of previously learned knowledge. Thus, the aim of this approach is a construction of more autonomous planning system by including a concept of the learning.

4.4.4.1 The Encoding Method of Processing States

In the case where the classifier system is applied to this planning problem, a conditional part corresponds to the current processing state. That is, a conditional part is compared with the encoded state of decomposed elementary volumes. Therefore, the length of a conditional part l_C is equal to the number of elementary volumes v_i. To perform the conditional matching, a current processing state is encoded into a bit string E by following encoding rule.

$$E = (e_1\, e_2\, \ldots\ldots\, e_{Nv}), \tag{4-4-24}$$

where,

$$e_i = \begin{cases} 1\text{: if } v_i \text{ is already removed} \\ 0\text{: otherwise} \end{cases}, \tag{4-4-25}$$

N_v is the number of given machinable elements, and an expression ($N_v = l_C$) is satisfied. Then, the conditional matching between an encoded message, E, and the conditional part of every classifier is carried out. If one or more classifiers match the message completely, they concurrently become candidates to be activated. To select a classifier that instructs a strategy of the system from candidate classifiers, rule competition is also carried out. In this competition, the usefulness of a candidate is evaluated according to strength values. As a result of the competition, a classifier having the highest bid value is selected. A bid value B is calculated by

$$B(cf_i) = \alpha\, Scf_i, \tag{4-4-26}$$

where, Scf_i is a strength value of a classifier cf_i, α ($0<\alpha<1$) is a biding parameter.

4.4.4.2 The Decoding Mechanism of Classifiers

According to the action part of a selected classifier, a machinable volume V_j that should be removed at that time is instructed. Based on the instructed V_j, elementary volumes are removed as follows,

$$\textit{if} \quad a_{ij} = 1, \quad \textit{then} \quad \text{remove } v_i, \quad (\text{for all } i \in I). \tag{4-4-27}$$

These procedures mentioned above are repeatedly performed until the following termination condition is satisfied.

$$e_i = 1, \quad (\text{for all } i \in I). \tag{4-4-28}$$

Consequently, a sequence of machining is represented by a chain of selected classifiers that transports a processing state from an initial state to the goal state.

4.4.4.3 A Reinforcement Method of Classifiers

In this planning system, the strength values of classifiers have to be revised based on each classifier's usefulness. However, when a solution of a task is represented as a sequence of classifiers, it is very difficult to judge the contribution of each classifier. In this study, this credit assignment mechanism is realized by the bucket brigade algorithm already described in this thesis. This mechanism is performed by continuous payment of bids of classifiers. The last classifier in the sequence receives an environmental reward because of finishing the objective part. As a result of the reinforcement learning, an appropriate process plan is acquired as a chain of classifiers with higher strength values.

4.4.5 GA-Based Seeding Mechanism of Initial Rules in Classifier Systems

Here, I discuss an integration method of above two approaches, to realize the autonomous planning system having higher problem solving performance. As we know, human beings can effectively solve a new problem by using knowledge of past solved problems. In the machine learning systems that mimic this mechanism, learned results of past problems are transformed

for a new problem by analogical reasoning. Classifier systems seem to be an useful tool to realize a machine learning system.

In the application of genetic algorithms to this operation planning problem, a searched solution is represented as a sequence of ordered operations. Hence, it is difficult to modify the solution string for solving new problems. The other hand, in the approach using classifier systems, learned knowledge is memorized as a set of reactive rules. Therefore the ordering procedure of these rules to accomplish new task is automatically done by re-learning mechanism in a classifier system. Even if unsuitable rules are stored in an initial set of classifiers, reinforcement mechanisms revise strength values of these unsuitable rules, and profitable classifiers for a new task can be strengthen. However all of the potentially feasible rules cannot be prepared in an initial rule set. When potentially effective rules are not in the rule set, an appropriate plan cannot be searched by only reinforcement algorithms. So, a rule discovery mechanism is one of the important elements in learning classifier systems. However, it is pointed out that there is a problem of a trade-off between exploitation of existing classifiers and exploration of un-examined classifier rules. That is, frequent replacement of classifier rules does not improve the learning performance. Therefore an effective seeding mechanism of initial rules is required in classifier systems.

So, I propose a mechanism by which initial classifiers in a classifier system are generated based on evolved strings by genetic algorithms. Figure 4.23 shows an outline of this proposed framework. In this figure, the executive cycle is represented as follows. For a given new task, if there is not enough knowledge to achieve the task, a solution is firstly searched by the GA-based searcher. Then, the searched solution is encoded into classifier representations by the rule generator. The classifier system having generated rules as initial classifiers learns the given task. After learning, reinforced classifiers are stored in the plan-base. When a similar task is given, learned knowledge is extracted from the plan-base, and is modified by the rule modifier. By performing this cyclic process repeatedly in a problem domain, the plan-base is refined and planning performance is improved.

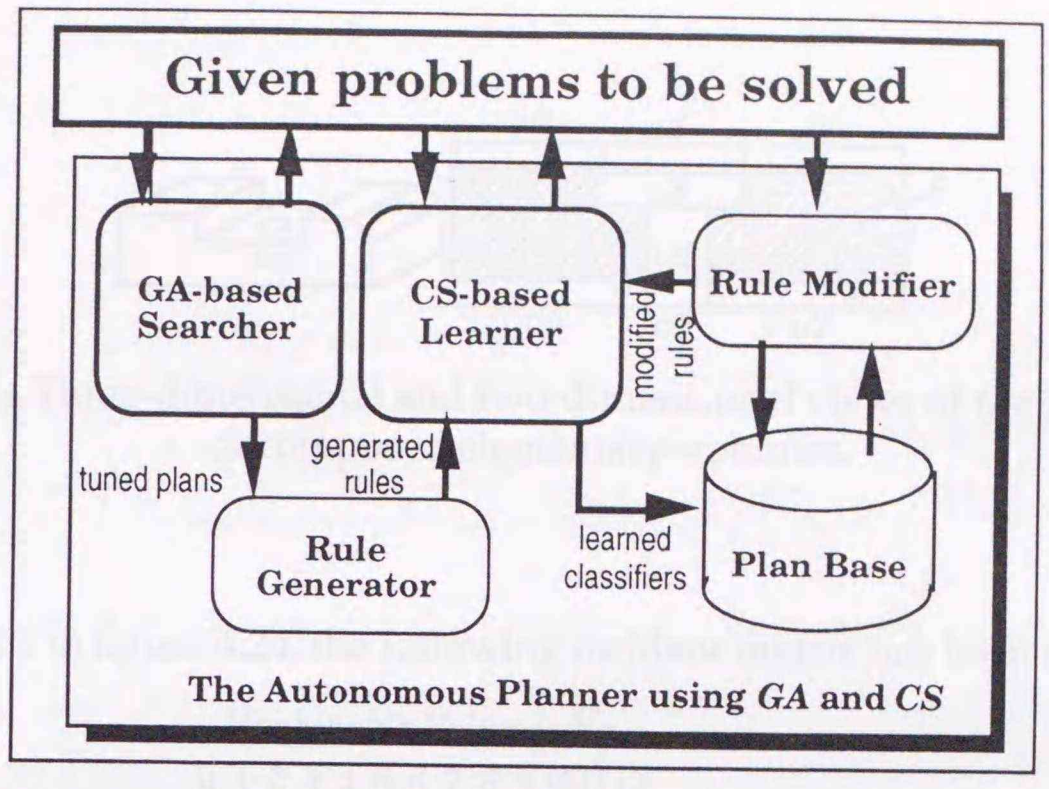


Figure 4.23 A proposed autonomous operation planning system integrated GA approach and CS approach.

4.4.6 Experimental Results

On the basis of the proposed approaches, a genetics-based process planning simulator is constructed and some numerical experiments are carried out. First, some process planning tasks are performed by the genetic algorithms approach. Second, the same tasks are solved by applying the classifier system.

4.4.6.1 Experimental Tasks

Here, I show the given experimental planning tasks as follows. Table 4.8 shows given experimental parts, and details of the part 1 are shown as figure 4.24.

Table 4.8 Given experimental parts.

	The number of elementary volumes v_i	The number of segmented machinable volumes V_j
Part 1	9	13
Part 2	20	23
Part 3	29	32

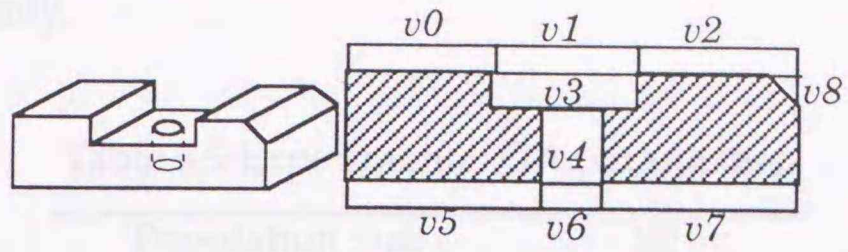


Figure 4.24 Three-dimensional and two-dimensional views of the part 1 with decomposed elementary volumes.

For the part 1 in figure 4.24, the following incident matrix has been constructed.

$$[a_{ij}] = \begin{array}{c} \\ \text{Elementary Volumes } v_i \\ \\ \end{array} \begin{array}{c} \text{Machinable Volumes } V_j \\ \begin{array}{c} 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12 \end{array} \\ \begin{pmatrix} 1 & 1 & & & & & & & & & & & \\ 1 & & 1 & & & & & & & & & & \\ 1 & & & 1 & & & & & & & & & \\ & & & & 1 & & & & & & & & \\ & & & & & 1 & 1 & & & & & & \\ & & & & & & & 1 & 1 & & & 1 & 1 \\ & & & & & & 1 & 1 & & & & 1 & \\ & & & & & & & 1 & 1 & & 1 & & \\ & & & & & & & & & 1 & & & \end{pmatrix} \end{array} \quad (4\text{-}4\text{-}29)$$

For each machinable volume V_j in an incident matrix (4-4-29), the following data have been specified;

1. Vector C of machining costs:

$$C = [69.8,18.3,29.3,22.1,50.3,16.4,19.4,58.1,56.0,\ 2.7,10.7,45.8,43.7]. \quad (4\text{-}4\text{-}30)$$

2. Vector T of required tools:

$$T = [t_1, t_2, t_2, t_2, t_2, t_3, t_3, t_1, t_1\ , t_4, t_2, t_5, t_1]. \quad (4\text{-}4\text{-}31)$$

3. Vector F of required fixtures:

$$F = [f_1, f_1, f_1, f_1, f_1, f_1, f_1, f_1, f_1, f_2, f_2, f_3, f_1]. \quad (4\text{-}4\text{-}32)$$

4. a tool changing cost: $e_t = 10.0$ (4-4-33)

5. a fixture changing cost: $e_f = 10.0$ (4-4-34)

4.4.6.2 Experiment 1

At first, three experimental process planning tasks in table 4.8 are solved by an approach using genetic algorithms. In these experiments,

experimental GA parameters are shown in table 4.9. Initial population is generated randomly.

Table 4.9 Experimental GA parameters.

Population size	30
Crossover rate	0.5
Mutation rate	0.01

Figure 4.25 shows how planning performances grow during the evolution through three types of data. In this figure, the axis of abscissa is the generation, and the axis of ordinate represents results of the best strings in each generation, that is, the minimum values of the total production costs. These results indicate that the strings having higher performances are generated by the tuning mechanism.

To examine the dynamics of the population, the best results, the average results, and the worst planning results are shown in Figure 4.26. This result indicates that the strings having low fitness values are eliminated and population evolves repeatedly each generation.

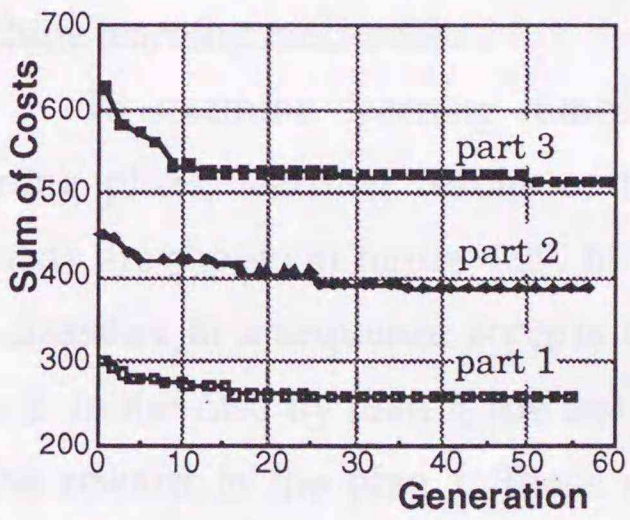


Figure 4.25 Search results of given three planning tasks by GA approach.

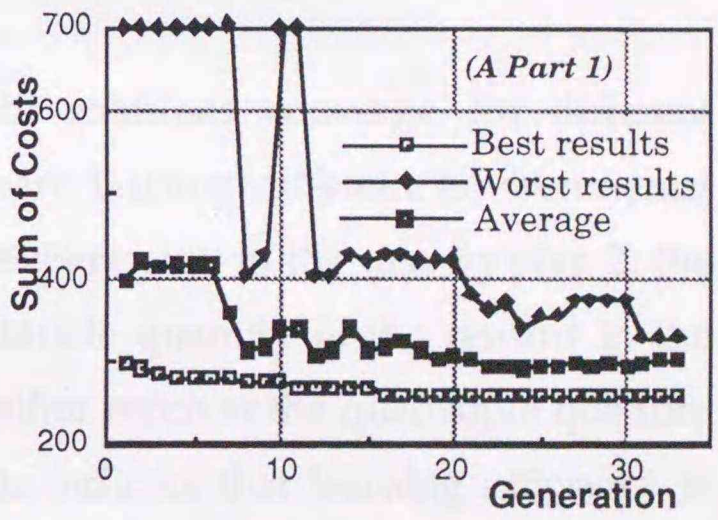


Figure 4.26 The behavior of the population while GA evolution.

4.4.6.3 Experiment 2

Second, experiments in the case where given planning tasks in table 4.8 are solved by an approach implementing a classifier system are carried out. In these experiments, the initial population of classifiers is generated randomly, and the number of classifiers is 2000. Also, strength values of initial classifiers are set equivalently. In the rule competition, when some classifiers having the same strength value are concurrently activated, a winner classifier is selected randomly.

Figure 4.27 shows the learning curves of three planning tasks above with no prior knowledge. In this figure, the axis of abscissa is the learning times, and the axis of ordinate represents the total processing costs. The solutions of a planning task converge at a near-optimal solution by a learning mechanism with the classifier system. This result indicates that only one chain of machining operations is made up, because the useful classifiers tend to have higher strength values than that of other classifiers by the rule reinforcement algorithms. Also, the solutions of a planning task vibrate in the earlier period of learning. It seems that the system searches extensively in the search space to diminish the unknown spaces. Figure 4.28 shows strength values of classifiers when the learning of the planning task of the part 1 is completed. It is clear that appropriate classifiers having higher usefulness values are extracted by this machine learning mechanism.

To examine learning times until solutions converge for different learning plans, learning results of the part 1 using different environmental rewards are shown in figure 4.29. In this experiment, in the case by plan 2, the last classifier in a sequence accepts the double quantity of the reward in the plan 1. In the case by plan 3, the last classifier receives the quadruple quantity of the reward in the plan 1. These results indicate that learning efficiency is varied based on the employed reinforcement algorithms.

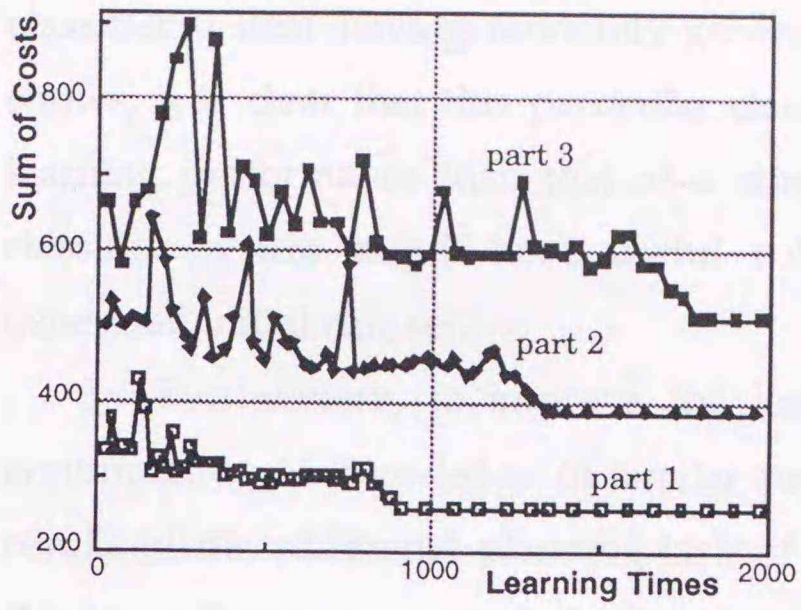


Figure 4.27 The learning curves of three planning tasks by CS approach.

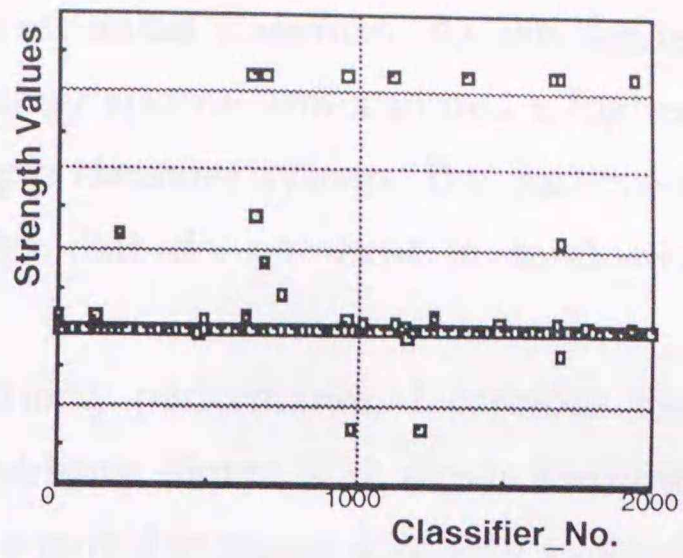


Figure 4.28 Strength values of the classifiers after learning the planning task of part 1.

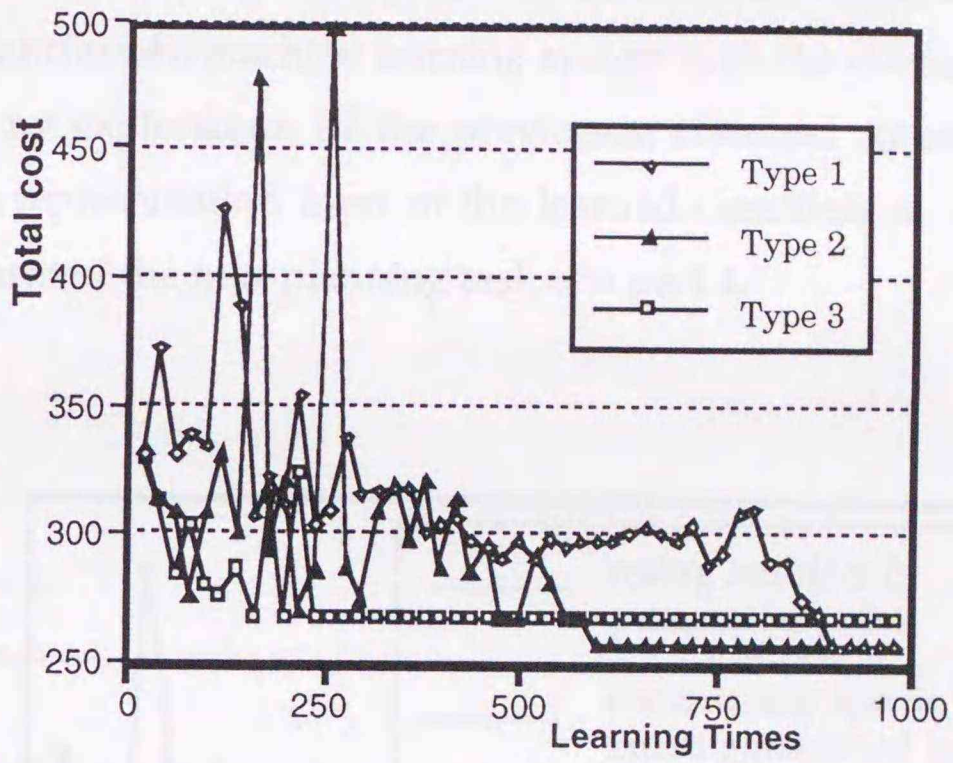


Figure 4.29 Learning efficiencies for different learning types (the part 1).

4.4.6.4 Experiment 3

I also performed a primary experiment for realizing an advanced operation planning system by integrating GA-approach and CS-approach. In this experiment, at first, a searched solution for the part 1 by GA-approach is transformed to a form of classifier representations. Then the sequencing task of part 1 is learned by a classifier system with transformed initial rules. Figure 4.30 illustrates learning curves of this particular classifier system and a simple

classifier system having randomly generated initial classifiers. As this figure shows, it is clear that this particular classifier system demonstrated a higher learning performance than that of a simple classifier system. The particular classifier system might have useful rules that do not exist in randomly generated initial rule set.

Furthermore, to improve the learning performance, I consider the accumulation of knowledge of similar problems. Figure 4.32 shows learning results of the additional planning task of a part 4 in figure 4.31 with evolved classifiers that are previously learned by a similar problem (i.e., planning task of a part 1). This figure shows the learning times until planning solutions converge. In this figure, the axis of ordinate represents learning times until planning solutions converge at optimal solutions. This result shows that the learning efficiencies of a machine learning system with the classifier system can be improved by exploitation of the previously obtained knowledge. In this experiment, a representation form of the learned classifiers is modified to an appropriate form of the new planning task of a part 4.

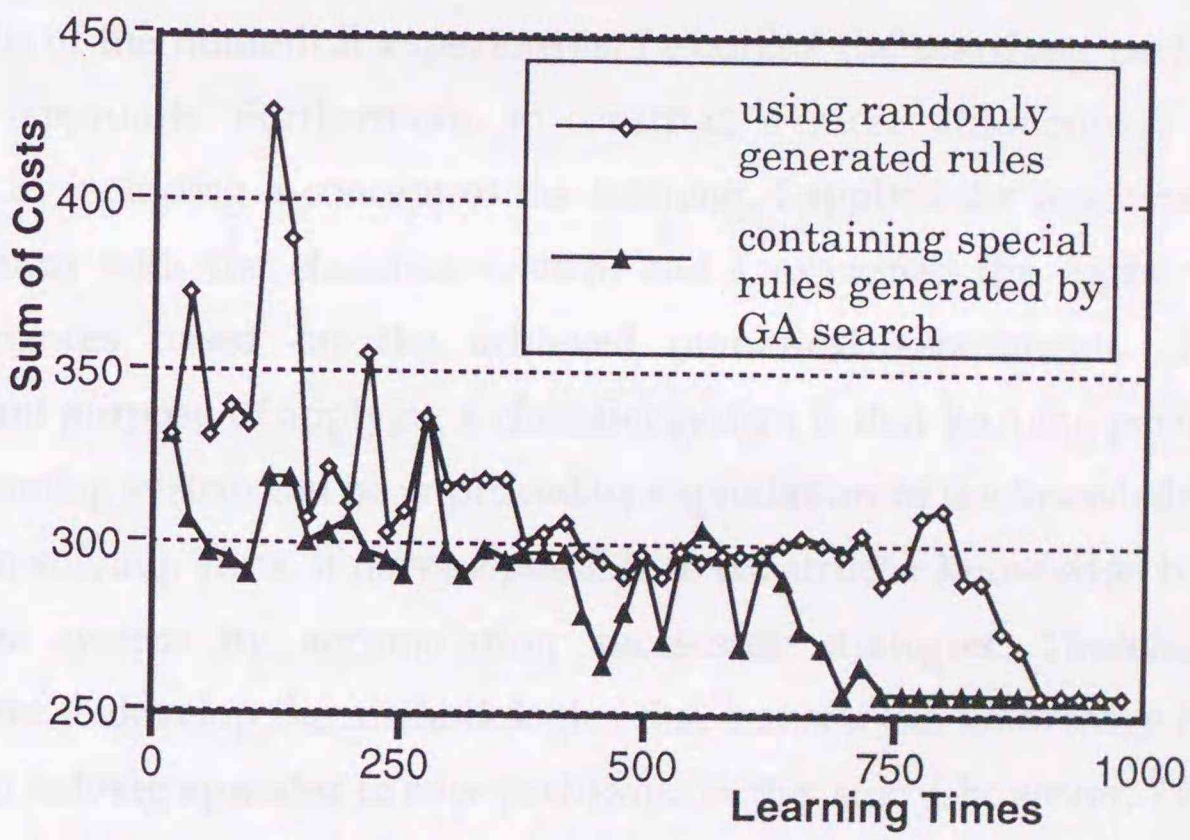


Figure 4.30 Learning curves of Part 1 using some special classifiers generated from the tuned plan by GA approach.

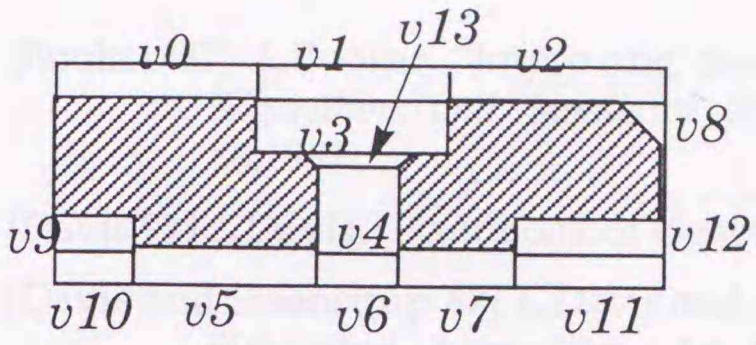


Figure 4.31 Two-dimensional view of a part 4 that is similar as a part 1.

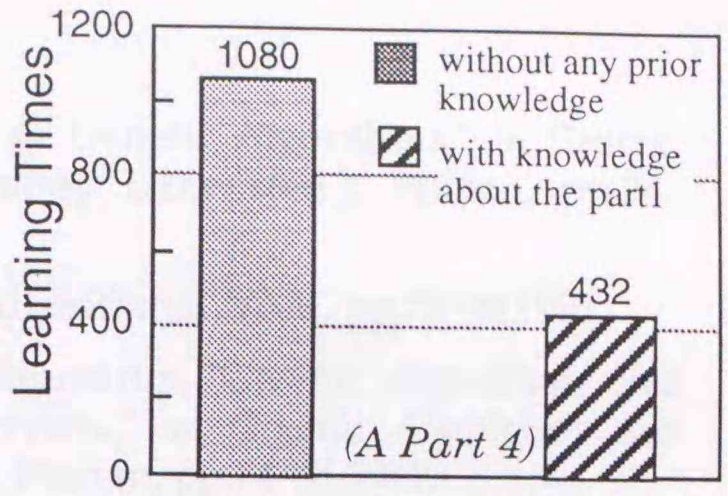


Figure 4.32 The improvement of learning performances through accumulation of knowledge of a similar problem.

4.4.7 Conclusion

In this study, new approaches on how to solve the sequencing problem in process planning tasks is illustrated. First, I proposed a searching method for solving this problem using the genetic algorithms, and to introduce automatic evolutional mechanisms to environmental adaptation, I applied order-based genetic operators to the permutations of the number of removal volumes. On the basis of the numerical experiments, I examine the searching performances of our approach. Furthermore, to construct a more autonomous planning system by including a concept of the learning, I applied the machine learning mechanism with the classifier system, and I examined the higher learning performances based on the achieved numerical experiments. The most important purpose of applying a classifier system is that learning performances of a planning system can be improved by exploitation of the knowledge of past problem solving. Thus, it may be possible to construct a knowledge base in the classifier system by accumulating successful strategies. Therefore, it is important to develop the methodologies that transfer the knowledge from past problem solving episodes to new problems. In this study, however, I treat only the case in which the knowledge of a similar task is learned. Therefore, in future work, an approach about this methodology in a case where quite different tasks are given will has to be discussed.

References

[Booker 87] L.Booker, "Improving Search in Genetic Algorithms," in *Genetic Algorithms and Simulated Annealing*, L.Davis(ed.), Pitman, pp.74-88,1987.

[Davis 91] L.Davis, "Handbook of Genetic Algorithms," VNR, pp.72-90,1991.

[Davis and Steenstrup 87] L.Davis and M.Steenstrup, "Genetic Algorithms and Simulated Annealing :An Overview," in *Genetic Algorithms and Simulated Annealing*, L.Davis(ed.), Pitman, pp.74-88,1987.

[Froeschl 93] K. A. Froeschl : "Two Paradigms of Combinatorial Production Scheduling Operations Research and Artificial Intelligence; Scheduling of Production Processes", Ellis Horwood, pp.1~21, 1993.

[Goldberg 89] D.E.Goldberg, "Genetic Algorithms in Search, Optimization & Machine Learning," Addison-Wesley, pp.1-25, 1989

[Grefenstette 87] J.J.Grefenstette, "A System for Learning Control Strategies with Genetic Algorithms," in *Proc. of 3rd International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp.183-190, 1987.

[Grefenstette 91] J.J.Grefenstette, "Lamarckian Learning in Multi-agent Environment," in *Proc. of 4th International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pp.303-310, 1991.

[Holland 75] J.H.Holland, "Adaptation in Natural and Artificial Systems," Univ. of Michigan Press, pp.20-120, 1975.

[Holland et al. 91] J. H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R.Thagard, "Induction," The MIT Press, pp.102-150, 1986.

[Kawakami and Kakazu 93] T. Kawakami and Y. Kakazu, "A Genetic Based Machine Learning System for the Process Planning", in *proc. of IEEE Int. Workshop on Neuro-Fuzzy Control*, pp.311-318, 1993.

[Kusiak 90] A. Kusiak, "Intelligent Manufacturing Systems," Prentice-Hall, pp.142-179, 1990.

[Tan 91] M.Tan, "Learning a Cost-Sensitive Internal Representation for Reinforcement Learning," in *Proc. of 8th International Conference on Machine Learning*, Morgan Kaufmann Publishers, pp.358-362,1991.

[Wilson and Goldberg 89] S.W. Wilson and D.E. Goldberg, "A Critical Review of Classifier Systems," in *Proc. of 3rd International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp.244-255,1989.

[Zhou and Grefenstette 89] H.H. Zhou and J.J. Grefenstette, "Learning by Analogy in Genetic Classifier Systems," in *Proc. of 3rd International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp.291-297,1989.

4.5 Autonomous Acquisition of Cooperative Robot Task Plans in Multiagent Environments

4.5.1 Introduction

To realize more flexible and intelligent engineering systems, it is expected to construct multiagent systems by which many difficult problems would be autonomously solved. However, the problem solving of many practical subjects is known as hard works. In these multiagent systems, multiple autonomous robots achieve given tasks and cooperate with each other to solve the task that is difficult or is never performed by single robot. Research interests on such an autonomous distributed robot system are related to interactions among agents. In other words, the important problem is the trade off between the autonomy of each agent and the adjustment of whole system.

I attempt to realize a mechanism by which cooperative strategies of agents can be acquired. This mechanism is realized by a machine learning using trial-and-error process. A block stacking problem is treated to examine our proposed method. A block stacking problem is well known as one of robot task planning problems, and is often used in an artificial intelligence research field. In this problem, given an initial state and a goal state of blocks, a solution is to be given as a sequence of each block's move to attain the given goal state. In particular, block stacking problems to be solved are described in a multiagent environment where multiple robots called agents exist in one problem domain. By the cooperation and apportionment of work, an efficient solution of a given task is obtained. In this problem domain, the work assignment of an agent is never known at first of learning. Through the adaptive learning mechanism, each agent learns an effective operation sequence. Moreover, each agent has to search the individual strategy to achieve a given task without communication with other agents.

In this approach, I adopt the classifier system proposed as a strong machine learning system to realize the adaptive planning mechanism. Each agent has a respective classifier system as an operation planner. Without an instruction from supervisor, agents learn operations individually.

On the basis of the proposed approaches, some numerical experiments are carried out on a constructed planning simulator. First, single-agent planning experiments are carried out to examine the learning performance of a classifier system for block stacking problems. Second, I also carried out experiments in two-agents environment. In this experiment, learning performances about acquisition of cooperative strategies are shown. In the remainder of this paper, the details of our methodology are descried, and the usefulness of the proposed method is examined.

4.5.2 Robot Task Planning Problem in Multiagent Environments

In what follows, I address the primary problem setting. A block stacking problem treated in this study is usually employed in AI or robotics fields. Given an initial state and a goal state of blocks, a solution of the problem is to be given as an optimal sequence of operations by which the given goal state is achieved with minimum cost.

This problem setting is the same as that defined in section 3.4.4. That is, there is a set of blocks, $B = \{ b_i ; i \in I \}$, where I is a set of indexes of blocks. All of the blocks have the same size and no weight. A block can be placed on the table or on another block, and there is no restriction to height of a stack of blocks. There are only a bounded number of slots, $s_i \in S$, into which blocks can be allocated. The number of slots is given. A placed position of a block i is represented as $P(b_i) = (s_i , l_i)$, where $l_i \in L$ is a vertical location named the layer.

In a multiagent environment, a set of robots R is described as

$$R = \{ r_k ; k \in K \}. \tag{4-5-1}$$

Based on this state description, each robot, r_k, can move only one block by following two operations at a time unit. That is,

Pick_Up(i): Pick up the top block in slot i. However no block is handled whenever slot i is empty or indicated i is not contained in the set S.

Put_Down(i): Put down the block that is currently being held into slot i. But the block is kept in the robot's hand whenever $i \notin S$.

Therefore, by using the command Pick_Up($i \notin S$) or Put_Down($i \notin S$), the waiting operation of a robot can be expressed. The solution to effectively achieve a given task is represented as a sequence of robot operations of robot k $op(t,k)$ at time t. Moreover I assume that each operation costs a pre-determined value. Generally, Pick_Up(i) and Put_Down(i) cost 1 identically. However the operation, by which a handled block is held on or no block is grasped as mentioned before, costs 0.5. Considering cooperative operations by multiple robots, there may be the special case in which to hold on the block and wait other robots' operations results more effective sequence. For the same task, consequently, better cooperative solutions may be searched by multiple robots than that in single robot domain. In this problem setting, the objective is to find the optimal operation sequence that minimizes total costs.

4.5.3 Implementation Method

To automatically solve the robot task planning problem mentioned above, learning classifier systems are implemented as an autonomous planner that makes an effective sequence of a corresponding agent. Reasons of employing a learning classifier system are as follows; (1) a realization of autonomous learning mechanisms, and (2) improving the learning performances by accumulation and practical use of previously learned knowledge. Thus, the aim of this approach is a construction of more autonomous planning system by including a concept of the learning.

In the case where the classifier system is applied to this block stacking problem, a conditional part is compared with the current state of blocks and the state of robot hands. To perform the conditional matching, a current state is encoded into a string by an encoding rule. Here, the search performance of the problem is seriously affected by the applied encoding methods. Thus, to perform the conditional matching between a state of blocks and a conditional part of a classifier, the problem state has to be encoded into an environmental message string. Representing full information of a problem state requires very long string, and then the search space of strings becomes very large. Therefore, to make an effective search environment, only useful features to plan operations have to be extracted from the problem state. However it is difficult to decide efficient encoding methods. If an useless extraction method is selected, then the search space may be greatly expanded, or important information may be lacked in encoded strings. There is a tradeoff the representational preciseness and the searching efficiency. But I won't discuss this problem in this thesis.

In our approach, to contract the searching space, a current problem state is encoded into a string *EM* as follows;

$$EM = ((mb_1, h_1), (mb_2, h_2), ...,(mb_n, h_n), rh_1,...., rh_m), \qquad (4\text{-}5\text{-}2)$$

where (mb_i, h_i) represents information of movable block in slot i. That is, mb_i is the number of movable block that means top block in the slot i, and h_i is height of position where movable block mb_i is placed, and rh_j represents a handling state of robot j, n and m are the number of given slots and robots respectively. Figure 4.33 shows proposed representational method.

Then, a winner classifier is selected based on the conditional matching and strength values of classifiers. According to the action part of a selected classifier, operations of corresponding robot at that time are instructed. An action part consists of two values, thus, a_1 and a_2. Based on the instructed a_1 and a_2, *Pick_Up*(a_1) and *Put_DOwn*(a_2) are carried out, then a state of blocks is transformed. These procedures mentioned above are repeatedly performed

until attaining the goal state. Consequently, a sequence of operations is represented by a chain of selected classifiers that transforms a problem state from an initial state to the goal state.

When there are multiple agents in a problem domain, I assume that each agent achieves an operation synchronously and alternately according to the predetermined order of agents (figure 4.34). So, the planning system of each agent has to search appropriate operations for a given task through cooperation with each other agent.

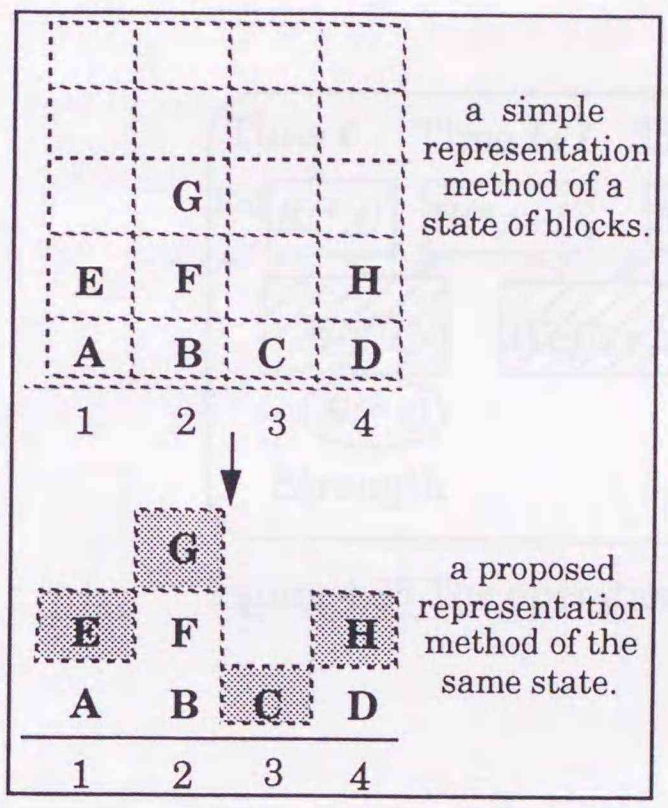


Figure 4.33 A proposed representation method of the state of blocks.

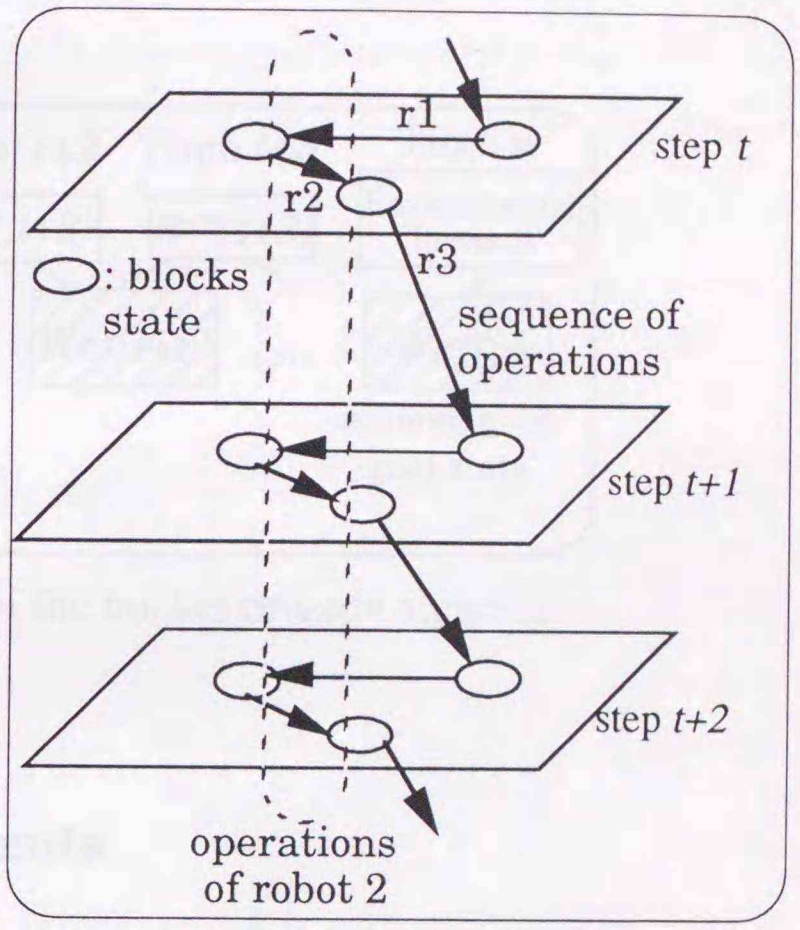


Figure 4.34 State transitions of blocks by alternative operations of multiple robots.

Here, the learning mechanism by which a set of classifiers is evolved is presented. A system has to revise the strength values of classifiers based on each classifier's usefulness. Thus, this credit apportionment task involves evaluating which classifiers have played important role in determining an eventual success. As a result of the learning, the system should assign higher strength value to useful classifiers that indicate a more confident strategy and lower strength value to useless classifiers. This credit assignment mechanism is realized by the bucket brigade algorithm that is a local technique for

apportioning the strength. That is shown as figure 4.35. Each matched classifier offers its bid for a rule competition at time t. After a competition, a winner classifier *Wcf(t)* pays its bid to a previous activated classifier *Wcf(t-1)*. Then *Wcf(t)* indicates a strategy of the system and accepts a bid from a succeeding classifier at time *t+1*. Finally the last classifier in the sequence receives an environmental reward because of achieving the goal state. To improve learning performances, environmental rewards are also given to the classifier by which sub-goal of the task is attained. As a result of the learning, a good operation plan is represented as a chain of classifiers with higher strength values.

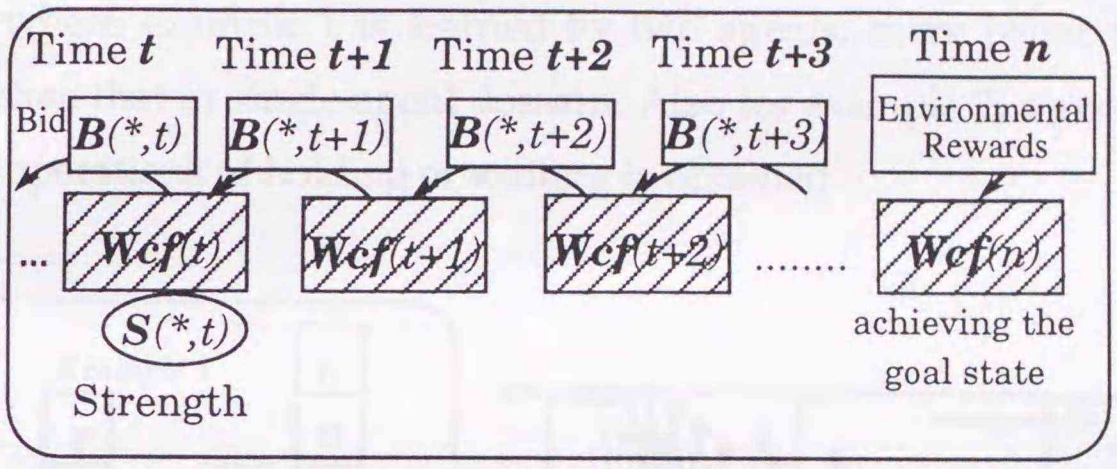


Figure 4.35 The operation of the bucket brigade algorithm.

4.5.4 Numerical Experiments

Based on the proposed method described above, I carried out some numerical experiments on a constructed task planning simulator. Figure 4.36 shows given experimental tasks. First, the task of example 1 is solved in two different domains; i.e., single-agent domain and two-agents domain. Secondly I also carried out an experiment of example 2 in a two-agents domain. The example 2 task cannot be achieved by only single agent. In our experiments, each agent has an individual classifier system in which there is an initial set of 500 classifiers. Each classifier is created randomly. Initial strength values of all classifiers are set equivalently. A new classifier is generated by genetic algorithms as a rule discovery mechanism.

Learning results of these experiments are shown in figure 4.37. In this

figure, the axis of abscissa is the learning times, and the axis of ordinate represents total cost to attain the goal states. The solutions of planning tasks converge at good solutions by a learning mechanism with the classifier system. This result indicates that only one chain of robot operations is made up, because the useful classifiers tend to have higher strength values than that of other classifiers by the rule reinforcement algorithms. Also, solutions of given tasks vibrate in the earlier period of learning. It seems that the agent searches extensively in the problem domain to diminish the unknown spaces. Since the high learning performances above are obtained for employed simple learning plans, the strong learning ability of the classifier system is examined. Moreover in a case where example 1 is learned by two agents, more better solution is acquired than that in single-agent domain. Also for example 2, a good solution involving operations of holding or waiting is obtained.

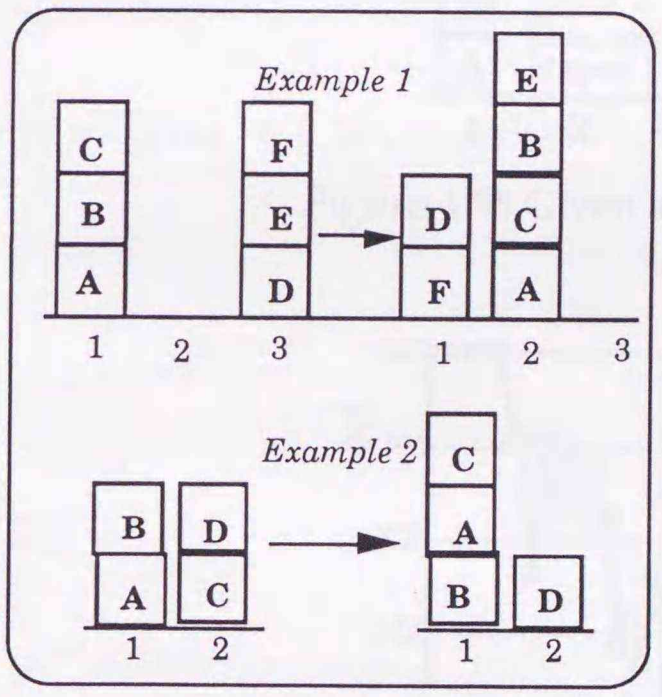


Figure 4.36 Given experimental tasks.

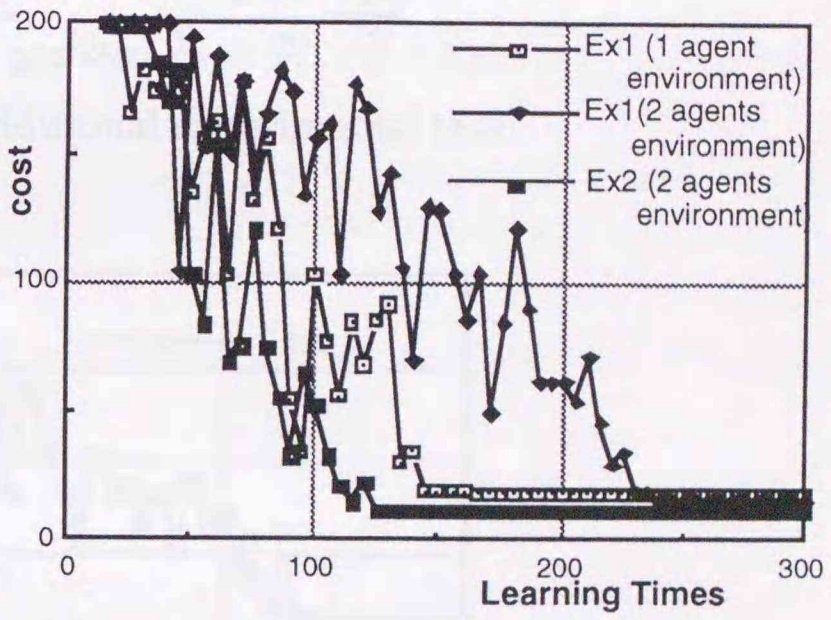


Figure 4.37 Learning curves of given experimental tasks.

Next I also carried out the experiment of example 3 shown in figure 4.38. In this experiment, each agent has the different task, and be in same domain. Therefore it is possible to obstruct other agent's operations, because each agent wants to achieve only individual task and has no way to communicate with others. For such a difficult task, I examine how cooperative and helpful (un-obstructive) operations are acquired by the distributed

autonomous learning mechanism. Each agent is assumed to suspend stacking operation after the agent attains the corresponding task. Figure 4.39 shows learning curves of example 3. In this figure, the axis of ordinate represents sum of respective total costs to attain goal states. As this figure shows, bad solutions are obtained in an earlier learning period. Thus, it seems that agents select obstructive operations. However in a later learning period, effective solutions for the whole system are found.

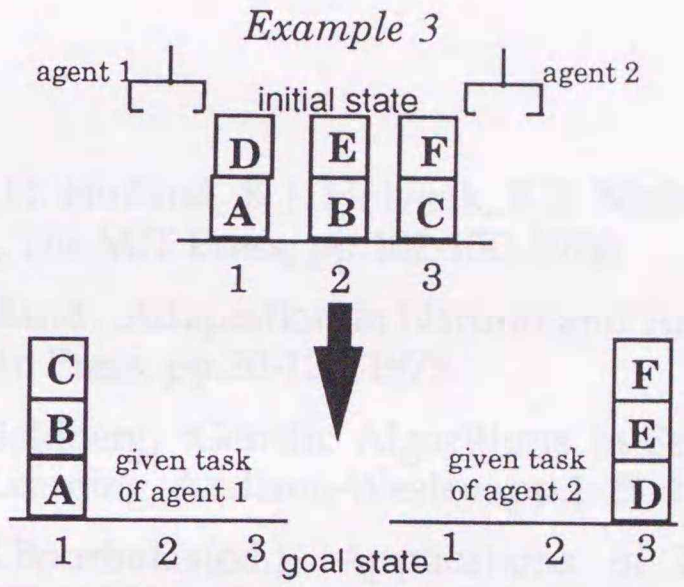


Figure 4.38 Given additional experimental task.

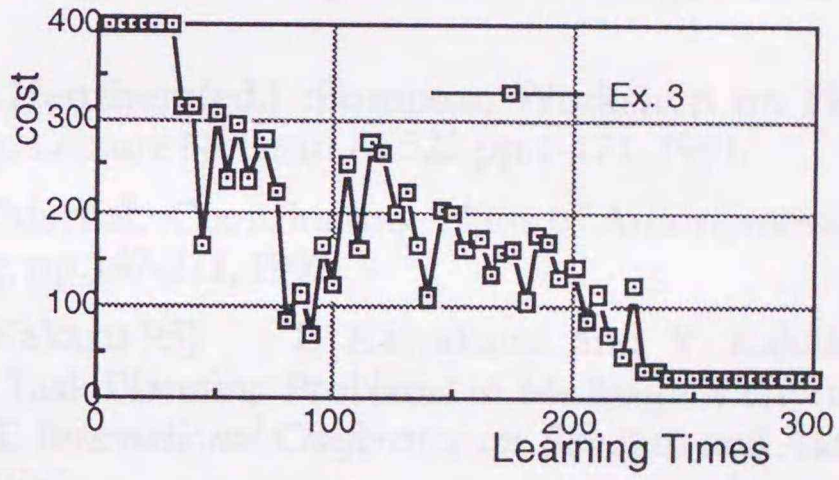


Figure 4.39 Learning curves of example 3.

4.5.5 Conclusions

In this study, I proposed an autonomous solving mechanism for a task planning problem by a machine learning system using a classifier system. I also attempted to acquire cooperative strategies in a multiagent environment where

multiple autonomous agents exist in the same problem domain. In this approach, a solution of an agent is obtained as a sequence of learned classifiers that correspond to production rules that instruct the appropriate operation. In multiagent environment, each agent has a respective classifier system as a robot task planner. On the basis of the proposed approaches, some numerical experiments are carried out. From some experimental results, high learning performance of the classifier system is shown.

References

[Holland et al. 86] J. H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R.Thagard, : Induction, The MIT Press, pp.102-150, 1986.

[Holland 75] J.H.Holland, :Adaptation in Natural and Artificial Systems, Univ. of Michigan Press, pp.20-120, 1975.

[Goldberg 89] D.E.Goldberg, :Genetic Algorithms in Search, Optimization & Machine Learning, Addison-Wesley, pp.1-25, 1989

[Bourbakis 91] N.G.Bourbakis(ed.), :Applications of Learning & Planning Methods, World Scientific, pp.183-364, 1991.

[Grefenstette 91] J.J. Grefenstette, :Lamarckian Learning in Multi-agent Environment, in *Proc. of 4th ICGA*, Morgan Kaufmann, pp.303-310, 1991.

[Hertzberg 91] J.Hertzberg(ed.) :European Workshop on Planning, Springer-Verlag, Lecture Notes in AI 522,pp.1-121, 1991.

[Martial 92] F.V. Martial, :Coordinating Plans of Autonomous Agents, Springer Verlag, pp.147-211, 1992.

[Kawakami and Kakazu 95] T. Kawakami and Y. Kakazu, :"A Study on Robot Task Planning Problems in Multiagent Environments", in *Proc. of IEEE International Conference on Robotics and Automation*, pp.2752-2757, 1995.

[Tan 91] M. Tan, :Learning a Cost-Sensitive Internal Representation for Reinforcement Learning, in *Proc. of 8th International Conference on Machine Learning*, Morgan Kaufmann Publishers, pp.358-362, 1991.

[Wilson and Goldberg 89] S.W. Wilson and D.E. Goldberg, :A Critical Review of Classifier Systems, in *Proc. of 3rd ICGA*, Laurence Erlbaum Associates, pp.244-255, 1989.

[Zhou and Grefenstette 89] H.H. Zhou and J.J. Grefenstette, :Learning by Analogy in Genetic Classifier Systems, in *Proc. of 3rd ICGA*, Laurence Erlbaum Associates, pp.291-297, 1989.

[Zoltkin and Rosenschein 91] G. Zoltkin and J.S. Rosenschein : Incomplete

Information and Deception in Multi-Agent Negotiation, in *Pro. of IJCAI'91*, pp.225-231, 1991.

[Schuurmans and Schaeffer 89] D. Schuurmans and J. Schaeffer : Representational Difficulties with Classifier Systems, in *Pro. of 3rd ICGA*, pp.328-333, 1989.

4.6 An Approach to Reactive Motion Planning Problems of Robot Manipulators

4.6.1 Introduction

In this section, I attempt to realize an autonomous motion planning mechanism of a robot manipulator by the GAPS. The motion planning problems of robot manipulators are well known that how to get the solution of the problems is said to be the one of the most difficult problems in robotics and artificial intelligence fields. In this problem, given initial state of a robot manipulator, the appropriate sequence of robot motions to attain the objective state is planned as the solution of the problem. It is difficult to induce a solution of the problem when a complex problem domain is given. Appropriate manipulator motion plans mean the plan by which a manipulator is avoiding any collision with existing obstacles in a problem domain, and an end-effector of the manipulator effectively reaches the given goal position. As general approaches for this problem, some methods were proposed until today [Noborio 91]. That is, the configuration space method in which a state of manipulators and obstacles are represented in a specific workspace called the configuration space, and the potential method that utilizes definite field functions to accomplish given tasks, called the potential function. In both methods, information of positions and forms must be completely defined in the search space. Therefore, it is hard to apply both methods to the particular class of tasks in which problem domains are uncertain or non-stationary. In this application, I treat this problem under uncertain constraints where unknown obstacles exist in a problem domain and reactive plans have to be made based on sensed environmental information. To realize an autonomous reactive planning mechanism, a reinforcement learning algorithm that utilizes trial-and-error processes is applied to this problem. As a reinforcement learning system, I

employ genetics-based learning classifier systems. The learning classifier system is a general purpose machine learning system. On the basis of the proposed approaches, some numerical experiments are carried out and some successful results are obtained.

4.6.2 Motion Planning Tasks of Robot Manipulators

The motion planning problem of robot manipulators is a typical planning problem in the robotics research field. In this problem, given initial state of a robot manipulator, the appropriate motion plans to attain the objective state is made as the solution of the problem. It is difficult to induce a solution of the problem when a complex problem domain is given. For example, when present obstacles are static, and position information and configuration information of obstacles are completely given in a workspace of the manipulator, then some already proposed methods can be utilized. That is, the configuration space method [Noborio 91] and the artificial potential method [Khatib 86] are well known.

In the configuration space method, a state of a manipulator and obstacles are represented in a joint angle space of the manipulator called the configuration space. A motion planner finds a path solution in the configuration space. The searched path solution expressed in the configuration space is uniquely translated into a trajectory in the real workspace. Since a robot manipulator is represented a point in the configuration space, it is easy to treat a manipulator in a search space, and it is comfortable to design planning algorithms. However, the configuration space has high dimensionality that is proportional to the number of joints. So, huge computational cost is required to calculate the configuration space.

The other hand, in the artificial potential method, the virtual potential function that is settled based on the distance between links and obstacles, is defined. This potential function produces a kind of repulsive force in the potential field. Concurrently, another virtual potential function that gives a sort of adsorptive force from the goal position, is defined. Then these forces are

transformed into the torque of joints. It is said that this method does not directly depend on the degree of freedom of the manipulator. However, appropriate potential functions that make dead-lock free motion plans, must be defined in this method. If a complex problem domain is given, then large computational cost is also required to calculate these potential functions.

Moreover, in both methods, information of positions and forms must be completely defined in the search space. Therefore, it is hard to apply both methods to the particular class of tasks in which problem domains are uncertain or non-stationary.

In this problem setting, the motion planning is carried out in uncertain problem domain. That is, the planning agent does not initially know any information of a goal position nor obstacles. I employ a simplified model of robot manipulators as shown in figure 4.40. This model is defined as follows;

$$L = \{ l_i ; i = 1,2,..,n \}, \tag{4-6-1}$$

$$J = \{ j_i ; i = 1,2,..,n \}, \tag{4-6-2}$$

$$LL = \{ ll_i ; i = 1,2,..,n \}, \tag{4-6-3}$$

$$TL = \{ tl_i ; i = 1,2,..,n\text{-}1 \}, \tag{4-6-4}$$

$$Q = \{ q_i ; i = 1,2,..,n \}, \tag{4-6-5}$$

$$OB = \{ob_j ; j=1,2,..,m \}, \tag{4-6-6}$$

where, L is a set of links of a manipulator, J is a set of joints of a manipulator, LL is a set of length of links, TL is a set of twisted angles of links, Q is a set of joint variables, and OB is a set of existing obstacles. I assume the problems in which every joint is the revolving joint, and values of all twisted angles are 0 degree. Thus, a robot arm in the problem is a planar manipulator. I also assume that all obstacles and given goal position are static.

A configuration of the manipulator is represented by a vector of joint angles, $\boldsymbol{q}$, that is,

$$\boldsymbol{q} = (q_1, q_2, \cdots, q_i, \cdots\cdots, q_n)^{\mathsf{T}}, \tag{4-6-7}$$

where q_i is a joint variable and T is a transposed operator. Motions of a robot manipulator are controlled by a displacement vector of joint angles, $\Delta\boldsymbol{q}$, at each

time step (figure 4.41).

$$\Delta q = (\Delta q_1, \Delta q_2, .., \Delta q_i, ..., \Delta q_n)^T. \quad (4\text{-}6\text{-}8)$$

Therefore, a configuration at time t, $q(t)$, is represented as,

$$q(t) = q(t\text{-}1) + \Delta q(t). \quad (4\text{-}6\text{-}9)$$

A controllable range of each Δq_i at unit time is also fixed as,

$$-\alpha \leq \Delta q_i \leq \alpha. \quad (4\text{-}6\text{-}10)$$

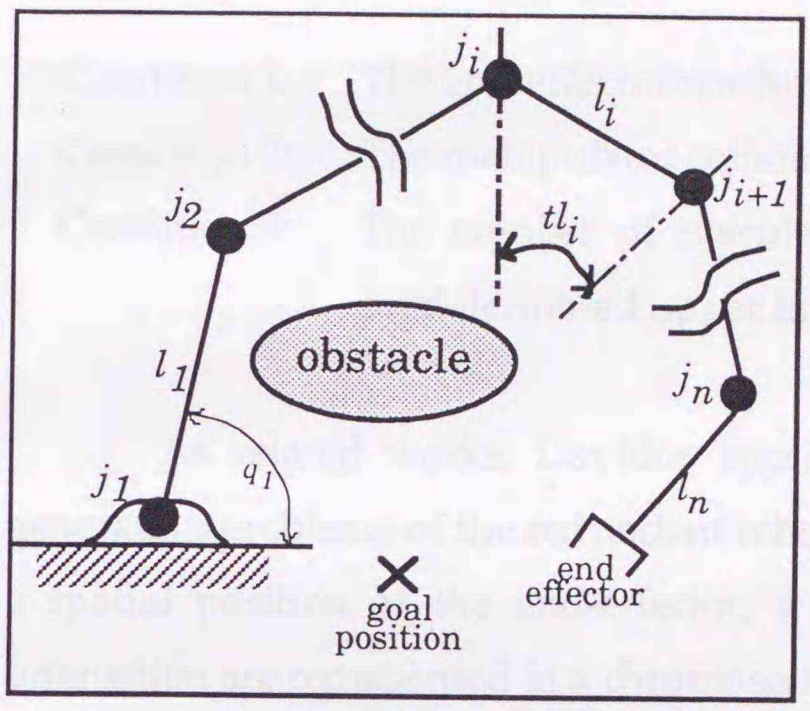


Figure 4.40 A simplified model of robot manipulators.

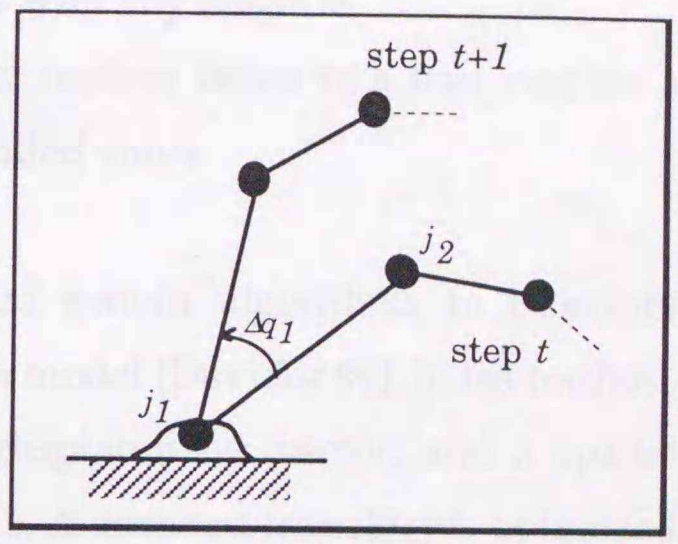


Figure 4.41 The motion control of a manipulator.

To perform a reactive planning in uncertain problem domain, some sensors are implemented on the agent. Each joint variable, q_i, is perceived by proprioceptive sensors implemented on each joint. The agent also has a collision detective sensor that perceives a collision information between robot arms and obstacles. This collision detective sensor can judge only information that indicates 'colliding' or 'not colliding'. The agent can receive no information about contacting positions on arms, nor information about distances between obstacles and the manipulator. Therefore, existing obstacles in the environment cannot be explicitly modeled in the planning agent. Since the agent initially has no information about the goal position, conventional planning methods cannot be applied. As reinforcement signals, the environment gives only distance

information between the goal position and the end-effector of the manipulator. That, but, does not indicate direction information.

On the basis of this problem description, the agent executes one motion at every unit time. The agent makes motion plans from given initial state to an objective state in which the end-effector reaches the goal position. A solution of motion plans is represented as a sequence of angle displacement vectors, $\Delta q(t)$, from an initial state to an objective state. One planning cycle called a trial terminates when one of the following conditions is satisfied.

Condition 1: The end-effector reaches the goal position.
Condition 2: The manipulator collides with any obstacle.
Condition 3: The number of executed motion times in a trial reaches a predetermined upper limited value.

As related works, Davidor applied genetic algorithms to trajectory generation problems of the redundant robot model [Davidor 91]. In his method, a spatial position of the end-effector, a displacement vector, and a spatial orientation are represented in a chromosome. A decoded trajectory is optimized by genetic algorithms, but it does not concern with learning process. Patel and Dorigo implemented learning classifier systems on a robot arm to learn simple light approaching tasks [Patel and Dorigo 94]. In their study, a robot arm is specified to a two-link manipulator, and the aim of it is to investigate the nature of some sensory information. So, the objective is different from this research. Dorigo developed distributed classifier systems named ALECSYS [Dorigo and Sirtori 91], and he applied the ALECSYS to simulated autonomous mobile robots to examine the performance of proposed systems [Dorigo 95].

4.6.3 Implementing Learning Classifier Systems

To solve a motion planning problem defined above, the genetics-based learning classifier system is implemented in the planning agent. The learning classifier system is implemented in the same manner of method described in

section 3.3.5. That is, a joint angle vector, q, is treated in the system as an environmental information. Through the detector of the classifier system, a joint angle vector, q, is encoded into an internal message, and then a conditional matching process is carried out to extract matched classifiers. Then, the system selects a winner classifier by a rule competition process. An angle displacement vector, Δq, is indicated by decoding the action part of the selected classifier. As a result of these processes, the state of the manipulator is changed, and a new environmental information is observed (figure 4.42).

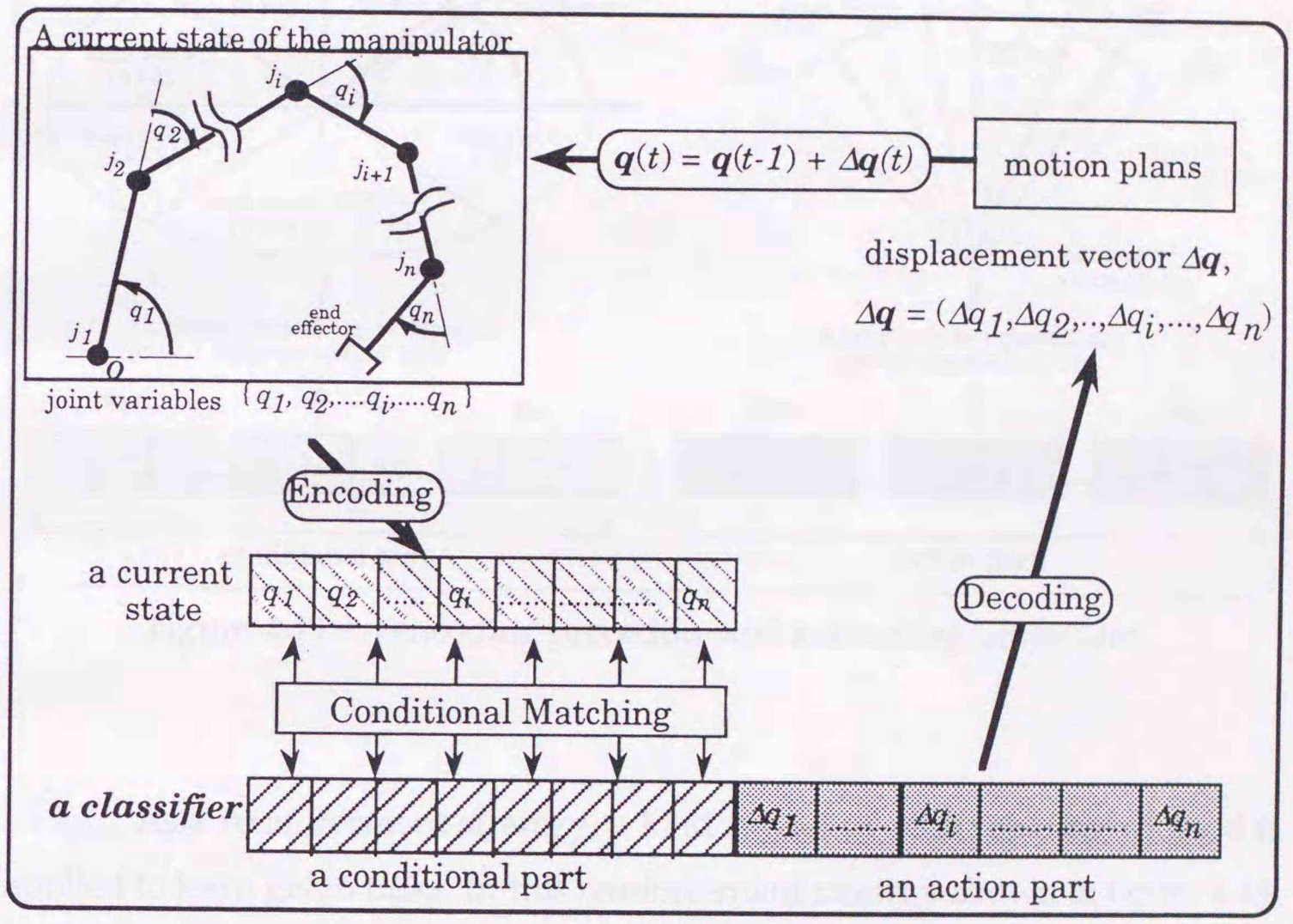


Figure 4.42 An implementation method.

In this application, the encoding process and the decoding process are accomplished based on the examined good resolutions of sensors and controllers in section 3.3.5. That is, the measurable range of joint variables is divided in eight regions, and a joint variable, q_i, is encoded on a three bit-lengthen sub-string. Similarly, the controllable angle displacement values are

specified in eight discrete quantities, hence an angle displacement value, Δq_i, is acquired by decoding a three bit-lengthen sub-string. A gray coding method is also applied rather than a usual binary coding method. Because in a gray coding method, a Hamming distance among neighboring regions is to be 1. An encoding procedure and a decoding procedure are carried out as shown in figure 4.43.

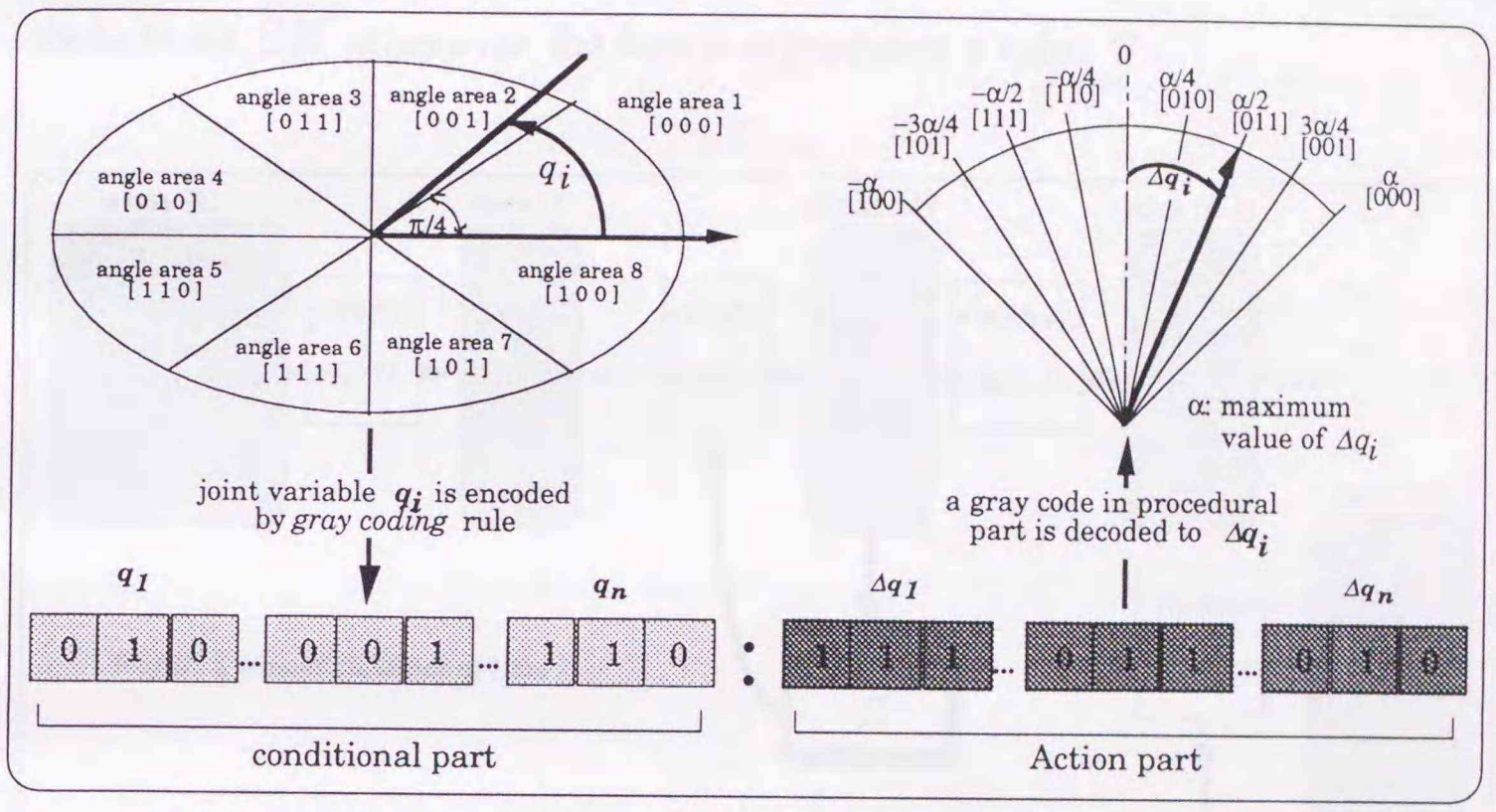


Figure 4.43 An encoding procedure and a decoding procedure.

As a reinforcement strategy, a kind model of profit sharing method is applied to learn given tasks. In this reinforcement strategy shown in figure 4.44, strength values of classifiers are revised by given environmental reward signals that are determined according to the moved result of the manipulator at every time step. Let $\Delta q(t)$ be a displacement vector at time t, $q(t)$ be a new state of manipulator observed as a result of a motion $\Delta q(t)$, $EE(t)$ be a position of the end-effector in a state $q(t)$, and p be a goal position. Then an environmental reward signal r^t is given according to the following conditions. That is, a strength value of the classifier that indicates a motion $\Delta q(t)$, is changed by adding the r^t.

1) *if* $d(p, EE(t)) < d(p, EE(t\text{-}1))$ *then* r^t = fixed small reward value

2) *if* $d(p, EE(t)) < \varepsilon$ *then* r^t = fixed large reward value

3) *if* $CC(\boldsymbol{q}(t), \boldsymbol{OB}) = 1$ *then* r^t = fixed large penalty value

where, d is a distance function among two points, ε is a small value to judge the accomplishment of a goal state, CC is a collision detective function that shows a value '1' when the manipulator collides with any obstacles in an existing obstacle set $\boldsymbol{OB}$, otherwise, the function produces a value '0'.

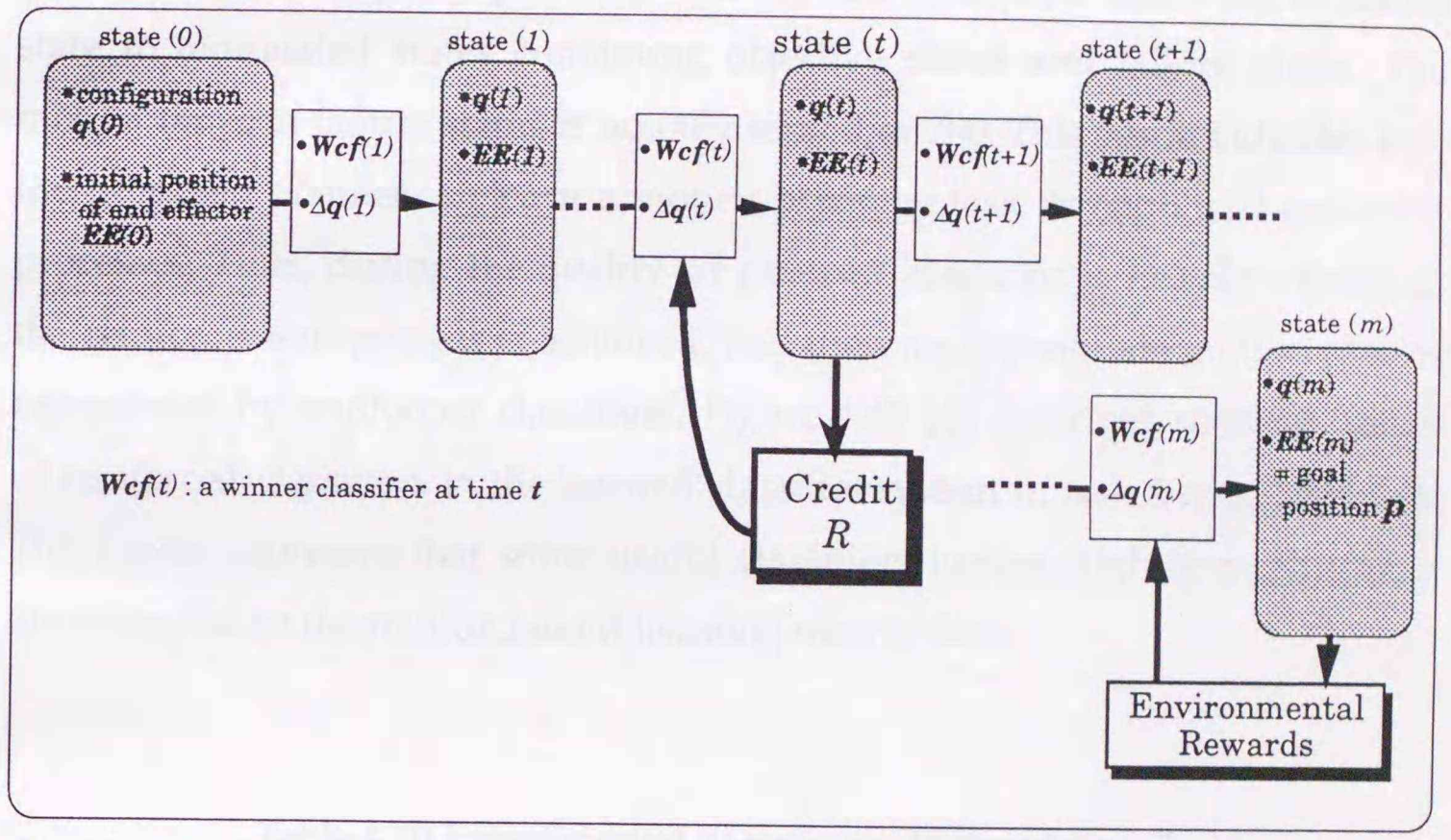


Figure 4.44 A reinforcement strategy. Strength values of classifiers are revised by given reward signals that are determined according to the moved result of the manipulator at every time step.

4.6.4 Experimental Results

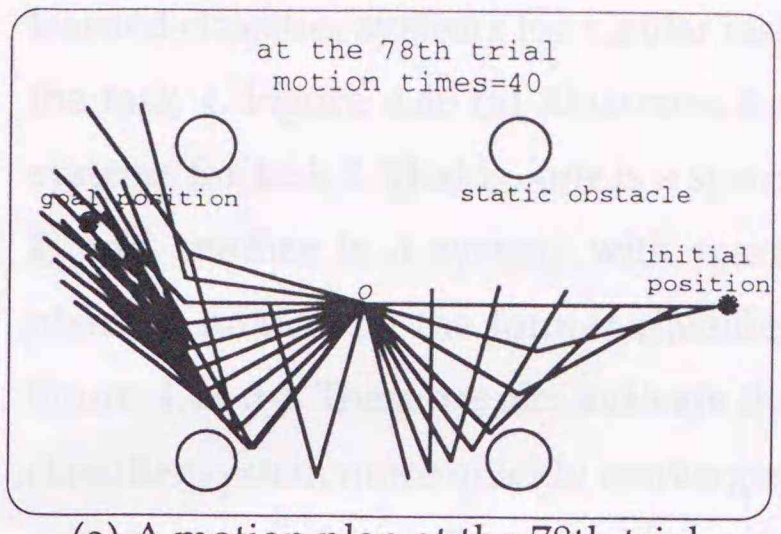
On the basis of the proposed approaches, some numerical experiments are carried.

4.6.4.1 Simulation 1: a 2-link planar manipulator in the unknown workspace.

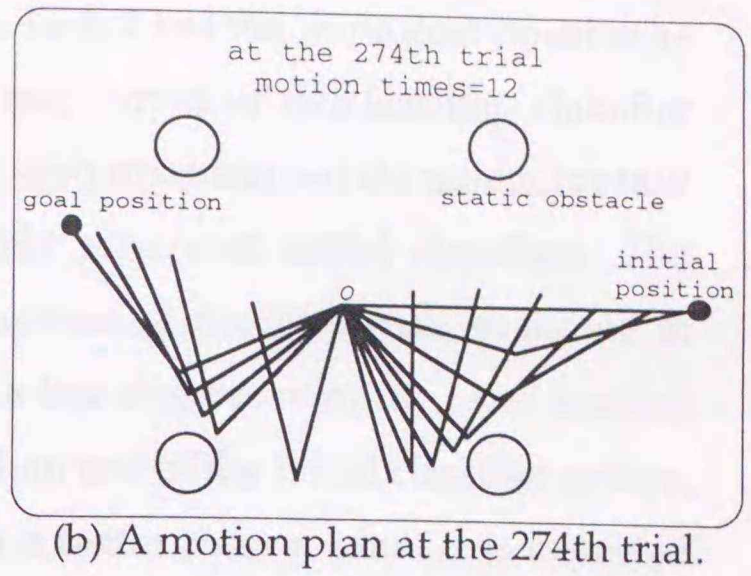
Firstly, to examine the learning performance of the motion planner based on learning classifier systems, simple experiments are carried out. In this simulation, experimental parameters are settled to searched parameters in section 3.3.5 (table 4.10). Figure 4.45 gives the results of motion planning experiment of the task 1. Figure 4.45(a) shows a planned trajectory at the 78th trial in learning phase. The best motion plan searched by this planner is shown in figure 4.45(b). Figure 4.45(c) illustrates the learning curve of task 1. In this figure, the vertical axis represents total motion times in a trial from an initial state to terminated states containing objective states and failure states. The motion times in failure states is equally settled at 200. This result indicates that the proposed planner can learn a motion planning task through trial-and-error processes. Thus, during the quality of planned solutions is heavily vibrating, the reinforcement process is achieved, and then an appropriate motion plan is represented by reinforced classifiers. Figure 4.45 (d) describes strength values of reinforced classifiers in the learned classifier system of task 1 over 300 trials. This figure expresses that some useful classifiers having high strength values are extracted by the reinforcement learning mechanism.

Table 4.10 Experimental parameters in simulation 1.

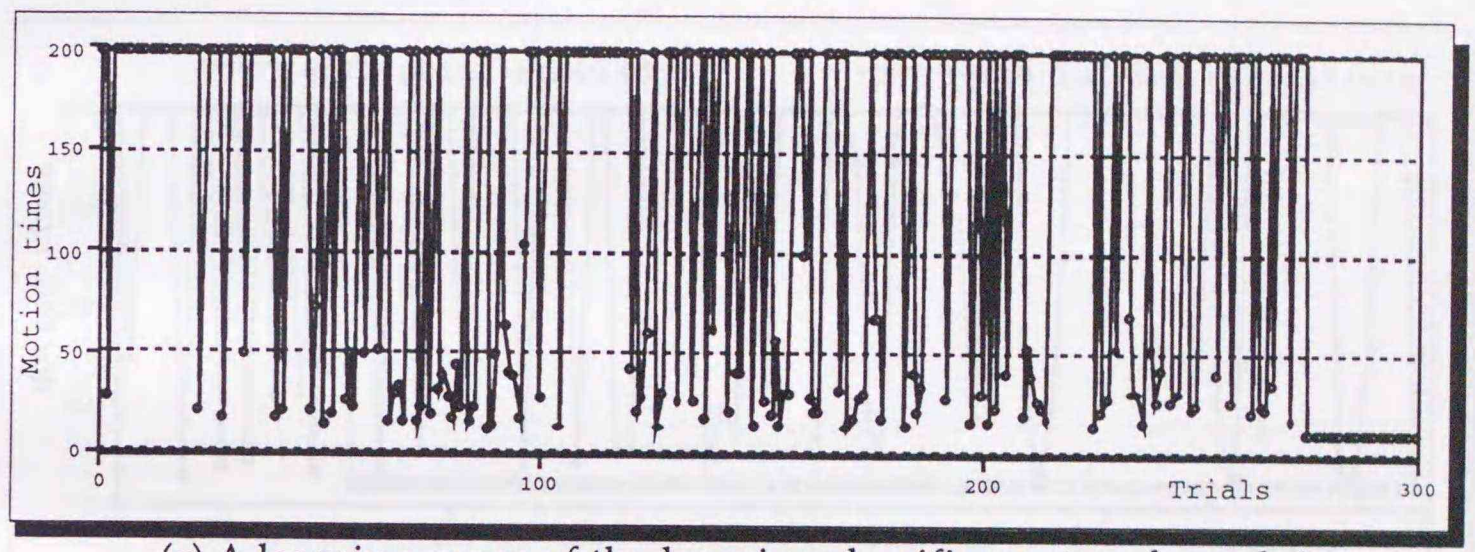
The range of displacement value of joint angle	$[-\pi/6, \pi/6]$
The number of classifiers	3500
Generating probability of symbol '#'	0.2
Initial strength value of classifiers	1000
The divided number of angle variable regions	8
The divided number of motion controllable regions	8
Discount coefficient in profit sharing method: α	0.1
Starting probability of rule discovery procedure: ρ	0.05
Crossover rate in a rule discovery procedure: χ	0.002
Mutation rate in a rule discovery procedure: μ	0.002



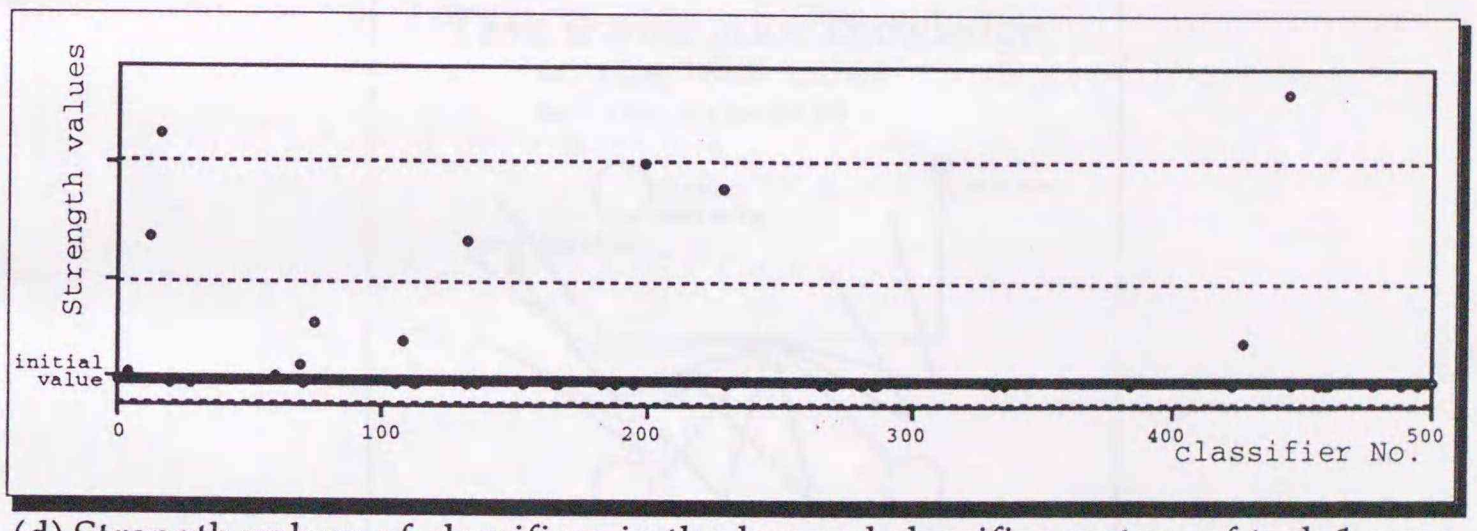
(a) A motion plan at the 78th trial.



(b) A motion plan at the 274th trial.



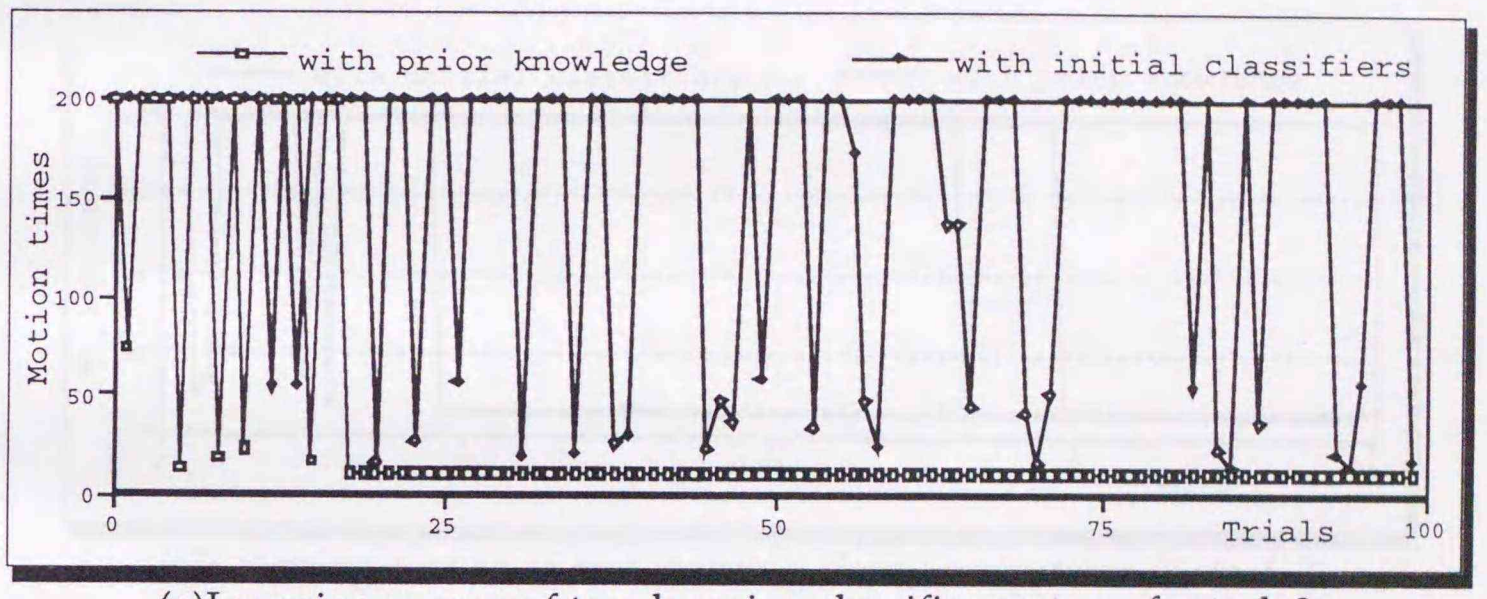
(c) A learning curve of the learning classifier system for task 1.



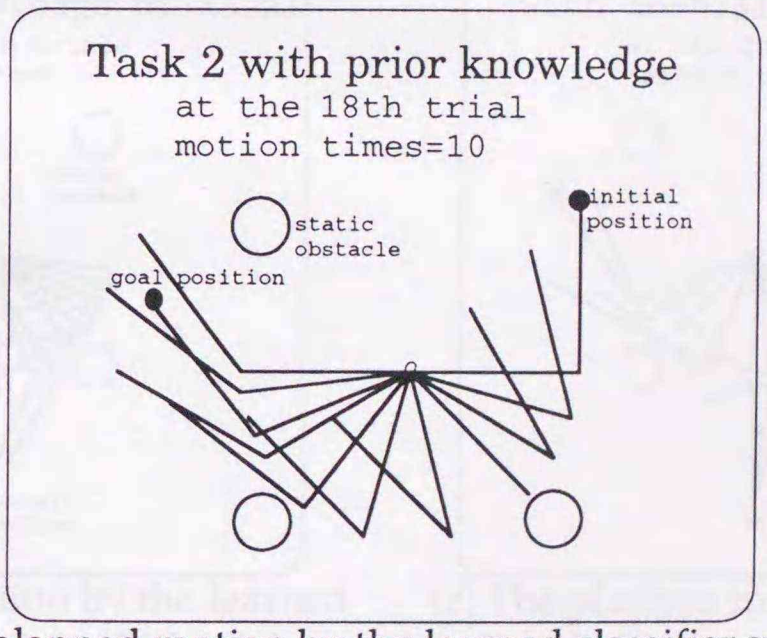
(d) Strength values of classifiers in the learned classifier system of task 1 over 300 trials.

Figure 4.45 Experimental results for task 1.

Next, I give the task 2 to examine the planning performance of already learned classifier systems for similar tasks. Task 2 has the same goal position as the task 1. Figure 4.46 (a) illustrates learning curves of two learning classifier systems for task 2. That is, one is a system with prior learned classifiers for task 1, and another is a system with randomly generated initial classifiers. The planned motion by the former classifier system at the 18th trial, is shown in figure 4.46 (b). These results indicate that a learning curve of the prior learned classifier system more quickly converges than one of the initial classifier system, and the learned classifier system searches a better motion plan. It is expected that the learning performance of classifier systems can be improved by accumulating the learned knowledge.



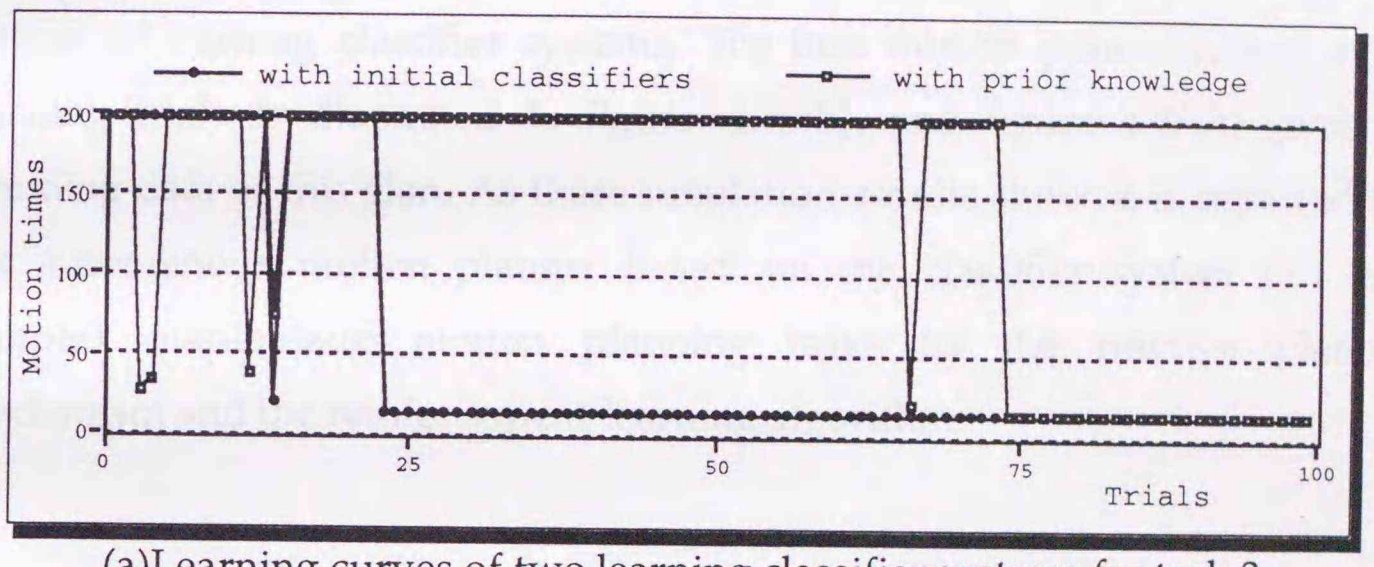
(a)Learning curves of two learning classifier systems for task 2.



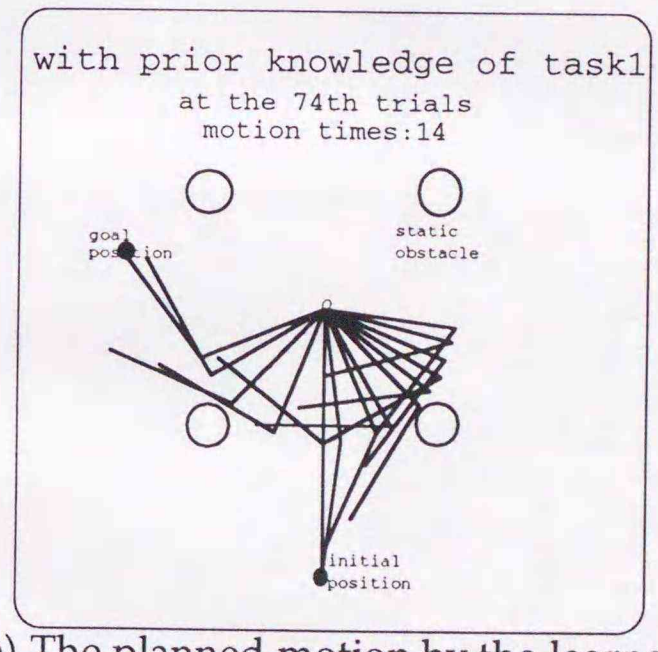
(b) The planned motion by the learned classifier system at the 18th trial.

Figure 4.46 Experimental results for task 2.

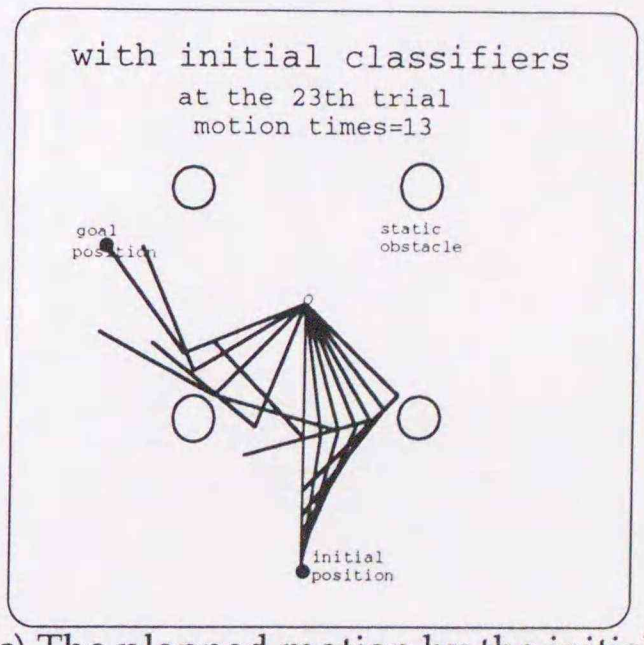
However, the learned classifier system does not always have higher learning performance than an initial classifier system. That is, learned knowledge might disturb the learning process for new tasks. Figure 4.47 represents experimental results for adding task 3. In this experiment, an initial classifier system and the same learned classifier systems as that used for task 2, are applied for task 3. As figure 4.47(a) shows, a learning curve of the prior learned classifier system slowly converges, and the learned classifier system cannot search any better solution than the initial system. This result seems to indicate that some important rules to solve task 3, have low strength values in the prior learned classifier system. Therefore, many trials are required to reinforce these important rules.



(a)Learning curves of two learning classifier systems for task 3.



(b) The planned motion by the learned classifier system at the 74th trial.

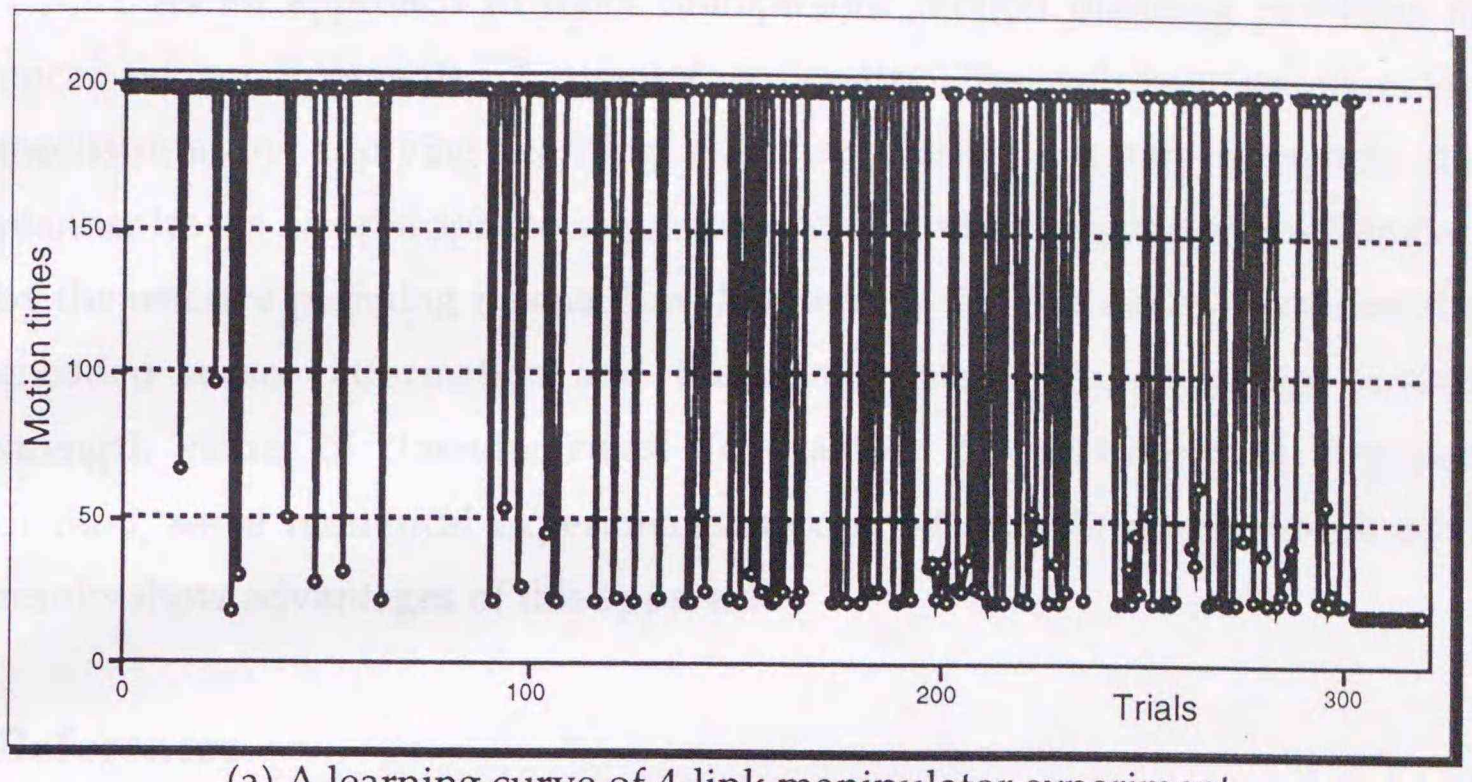


(c) The planned motion by the initial classifier system at the 23rd trial.

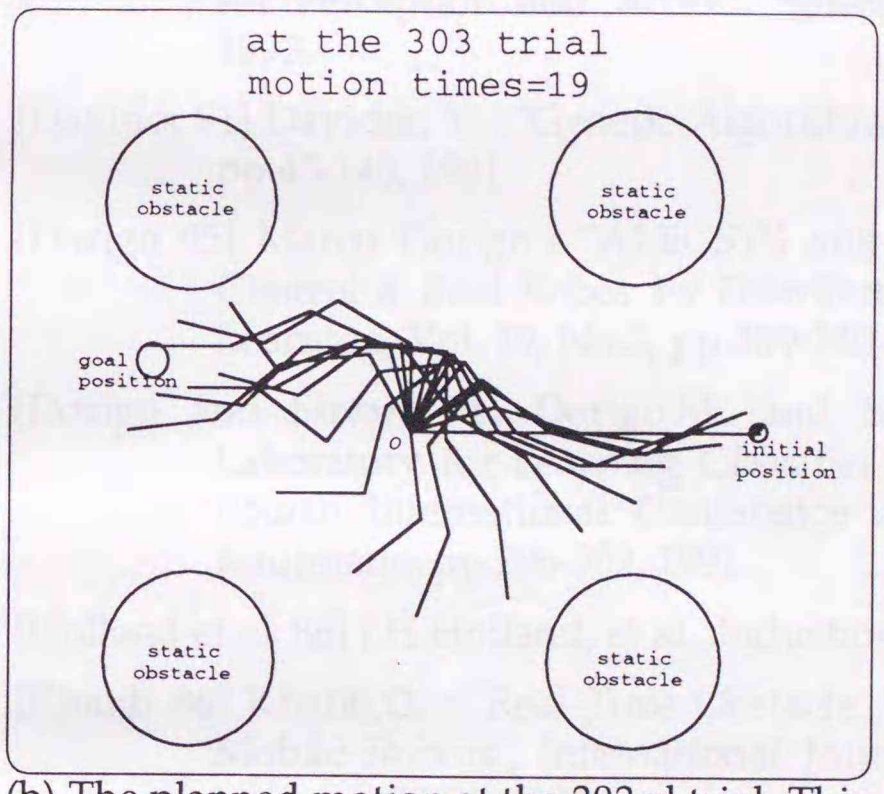
Figure 4.47 Experimental results for task 3.

4.6.4.2 Simulation 2: 4-link manipulator experiment

I also carry out a 4-link manipulator experiment to verify the learning performance of classifier system for more complex problems. Figure 4.48 gives experimental results of the learning simulation for a 4-link manipulator. In this simulation, the same experimental parameters excepting the number of classifier rules are applied as those used in simulation 1. The number of classifiers is 7000. Figure 4.48 (a) shows a learning curve of this simulation. The motion times of planned solutions are vibrating for a good while, because of influence of trial-and-error processes. Then the planned motion converges at an appropriate plan. The learning behavior of this simulation is similar to one identified in 2-link manipulator experiments. This seems to be the inherent feature of learning classifier systems. The best motion plan searched in 400 learning trials is illustrated in figure 4.48(b), and figure 4.48(c) gives the planning data of this plan. As these simulation results show, it is expected that the autonomous motion planner based on the classifier system can learn complex manipulator motion planning tasks by the reactive planning mechanism and the reinforcement learning algorithm.



(a) A learning curve of 4-link manipulator experiment.



(b) The planned motion at the 303rd trial. This plan is the best result during 400 learning trials.

Learned Motion Plan

cost	angle0	angle1	angle2	angle3
00	0.0	0.0	0.0	0.0
01	15.0	-30.0	7.5	22.5
02	30.0	-60.0	15.0	45.0
03	37.5	-75.0	22.5	22.5
04	52.5	-105.0	30.0	45.0
05	60.0	-120.0	37.5	22.5
06	67.5	-135.0	45.0	0.0
07	97.5	-150.0	22.5	-22.5
08	67.5	-157.5	37.5	-30.0
09	37.5	-165.0	52.5	-37.5
10	67.5	-180.0	30.0	-60.0
11	60.0	-202.5	37.5	-75.0
12	52.5	-225.0	45.0	-90.0
13	67.5	-255.0	30.0	-75.0
14	60.0	-247.5	45.0	-52.5
15	75.0	-255.0	30.0	-30.0
16	105.0	-285.0	45.0	-52.5
17	120.0	-292.5	30.0	-30.0
18	90.0	-300.0	60.0	-45.0
19	105.0	-307.5	45.0	-22.5

(c) The acquired planning data of the best plan.

Figure 4.48 Simulation results of 4-link manipulator experiment.

4.6.5 Conclusions

As an approach to robot manipulator motion planning problems in uncertain environments, I attempt to realize the autonomous planning mechanism by applying learning classifier systems. In this approach, the planner learns an appropriate sequence of displacement vectors of joint angles, by the reactive planning mechanism that decides suitable actions based on the encoded sensor information and the reinforcement algorithm that revises strength values of classifier rules. To examine the usefulness of proposed method, some numerical experiments are carried out. Obtained experimental results show advantages of this approach.

References

[Ahuactzin and Talbi 93] J.M. Ahuactzin, E. Talbi :Using Genetic Algorithms for Robot Motion Planning, in Laugire, C. (ed.) : Geometric Reasoning for Perception and Action, Springer Verlag LNCS 708, pp.84-93, 1993.

[Davidor 91] Davidor, Y. : "Genetic Algorithms and Robotics", World Scientific, pp.45-148, 1991.

[Dorigo 95] Marco Dorigo : "ALECSYS and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems", Machine Learning, Vol. 19, No.3, pp.209-240, 1995.

[Dorigo and Sirtori 91] Dorigo,M. and Sirtori,E. : "Alecsys: A Parallel Laboratory for Learning Classifier Systems", in Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, pp.296-302, 1991.

[Holland et al. 86] J.H.Holland, et al. :Induction, MIT, pp.102-150, 1986.

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

[Naruse and Kakazu 94] Naruse,K. and Kakazu,Y. : "Reactive Planning by Distributed Agents that Include Multiple Learning Modules, Proceeding of Japan-U.S.A. Symposium on Flexible Automation'94, pp.1461-1464, 1994.

[Noborio 91] Noborio,H. : "Current Researches of Robot Path-Planning Algorithms", Computrol, No.34, pp.47-56, 1991. (in Japanese)

[Patel and Dorigo 94] Mukesh J. Patel and Marco Dorigo : Adaptive Learning of a Robot Arm, Evolutionary Computing (T.C. Fogarty ed.), Springer Verlag LNCS 865, pp.180-194, 1994

CHAPTER 5
Summary

The aim of this thesis is to represent results of my works on constructing the genetics-based adaptive problem solver (GAPS) and on extending its mechanism. In this chapter, I summarize this thesis as follows;

In chapter 1, the background and the objective of this study are described as the introduction of this thesis. The outline of this thesis is also illustrated.

Characteristics of the GAPS are defined and formalized in chapter 2. For this purpose, I firstly, review traditional problem solving studies and approaches in artificial intelligence research area. It is pointed out that there are practical problems which can not be solved by conventional methods. Oppositely, in the approach of problem solving from nature, these difficult problems are expected to be solved by using interactions between problem solving systems and external environments. As the main implementation method in the evolutionary computation scheme, I focus on genetic algorithms to realize the adaptive problem solver that is inspired by the concept of problem solving from nature. Then I embed genetic algorithms in the proposed problem solver named GAPS, and formalize a structure of the GAPS. In the GAPS, I define the adaptive searcher and the behavior-based reinforcement

learner that commonly use the internal genotypic problem space to solve difficult problems such as combinatorial optimization problems or reactive planning problems in uncertain environments. To achieve the behavior-based reinforcement learning, learning classifier systems are also adopted in the GAPS. Since the learning classifier system is a machine learning system based on genetic algorithms, it can manipulate the same genotypic problem space.

In chapter 3, I propose extended mechanisms of simple classifier systems to develop the advanced GAPS, because simple genetic algorithms and simple classifier systems have some latent difficulties on implementation. So, firstly, I describe some difficulties on their implementations and review already reported theoretical extensive approaches. I especially focus on the mechanisms of classifier systems. To overcome pointed difficulties, I propose the following extending mechanisms;

1. Evolutionary synthesis of simple classifier system architectures by using genetic algorithms.
2. Extensive rule representation mechanism in classifier systems by defining the masking classifiers.

As the first difficulty, I focus on how the appropriate classifier system structure having high learning performance is to be designed. Section 3.3 shows the evolutionary architecture synthesis mechanism of simple classifier systems to generate an appropriate architecture for a given certain task. Thus, good learning parameters such as a reward function or resolution of sensors, must be suitably determined for given tasks. Although the parameter setting dominates a system architecture and largely affects learning performances of systems, it is not at all clear how this might be done. Before describing the evolutionary synthesis method, I formalize the simple classifier systems, and I define a set of parameters that dominates a system architecture. The representation method to encode a set of parameters on a chromosome, is also defined to apply genetic algorithms. As a fitness of each genetic chromosome, the evaluation criterion of corresponding simple classifier system architecture is described to adopt genetic operators. To examine the profit of proposed evolutionary method, I apply this method to the simple classifier system that learns two-link

manipulator motion planning tasks. From acquired simulation results, advantages of proposed approach are experimentally shown. That is, it is clear that an appropriate architecture of simple classifier system can be synthesized by genetic evolutionary mechanisms. Also, I clarify that the learning performances of classifier system architectures are largely affected by a combination of system parameters based the gotten experimental results.

As a second extensive mechanism, I focus on the representational difficulties of classifier rules. In section 3.4, the extensive approach of rule representations in classifier systems is provided. For this purpose, I propose and define a concept of new representational type of classifier, named masking classifiers. When the size of internally represented problem space is huge, it is difficult to learn a given task in a simple classifier system. Thus, the system must have a tremendous number of classifiers, and then it requires the great size of memory and the miserable amount of computational cost. Usually, to overcome this difficulty, a designer must consider an effective domain-dependent representational method. It is, however, hard to design a good encoding method. Therefore, the general approach for this difficulty is expected to be developed. For this expectation, I define masking classifiers that play a role of filtering schemata for general classifiers. The masking classifier extends the covering search space without increasing the number of classifiers. By this representational scheme, the profitable schema of environmental information is also automatically extracted and reinforced. The masking classifier system consists of two sets of classifier rules. One is a set of original classifiers, and another is a set of masking classifiers. A masking classifier consists of the 'don't care' symbol and the 'pass through' symbol. This masking classifier is not directly compared with the encoded environmental message. A reactive rule to be compared with an environmental message, is generated by combining the original classifier and the masking classifier. By combining a certain classifier and some variety of masking classifiers, some schemata can be examined for the classifier. As a result of this, a useful sub-space in a rule space is extracted for a problem state. That is, when only a partial information in a fully represented long string is needed to solve a problem, the benefit partial search space can be clarified by the effect of masking classifiers. I also define the

biding scheme and the credit assignment scheme of the masking classifier system to perform a sense-act behavior and a reinforcement mechanism. Then, the masking classifier system is applied to block stacking problem to verify above advantages. Experimental results show some features. That is, a masking classifier system having a smaller number of rules demonstrates the similar learning performance with that of a simple classifier system having a greater number of rules. From the result of revised connective strength values in the learned system, it is experimentally clear that the profit partial search space of a problem domain can be extracted by the masking effect.

In chapter 4, some engineering applications of the GAPS including proposed extensive mechanisms are presented. I describe applicabilities of the GAPS for a broader class of engineering problems by analyzing the problem solving performance of the GAPS based on the obtained experimental results of five practical problems.

(i) Firstly, 3-dimensional packing problems are evolutionarily solved as a complex combinatorial problem. 3-dimensional packing strategies are evolved by adaptive tuning systems in which the hybrid tuning method of GA-based search algorithms and heuristics is applied. That is, a packing procedure is expressed by heuristic rules that are represented as two evaluation functions. These heuristic rules are evolutionarily modified by the GA-based searcher in the GAPS. To exploit evolved heuristic rules for new 3-dimensional packing problems, I attempt to develop a 3-dimensional packing rule-base system. For the purpose of this, the similarity among 3-dimensional packing problems is defined as the difference of extracted feature information of problems. I also extend the problem setting, that is, I consider a situation where multiple containers exist. In multiple containers environments, hierarchical evolutionary mechanisms are also proposed.

(ii) As a second application, autonomous mobile robot navigation problems are treated by the GAPS. This problem is solved by classifier systems-based reinforcement learning. Multiagent environments, in which multiple mobile robots exist in one navigation area, are also treated. Each

agent induces an individual optimal solution for a given navigation task. The given navigation area is defined as a two-dimensional maze space that is suitable for representation of factories or offices.

(iii) Next, an approach to sequencing problems in process planning is described. This problem is settled as an ordering problem under geometrical constraints. In order to realize an autonomous sequencing mechanism, the two approaches are performed. One is the GA-based searching approach by which the problem is solved as a combinatorial optimization problem. As another approach, classifier systems solved the problem by reinforcement learning algorithms to obtain the further advantages.

(iv) As a fourth application, robot task planning problems in multiagent environments are handled by the GAPS. The block stacking problem is treated as the one of robot task planning problems. In multiagent systems, each robot has a classifier system as its task planner. Autonomous robots achieve given tasks and cooperate with each other to effectively solve the particular task that is difficult to be solved by single robot. From obtained simulation results, it is confirmed that a cooperative plan is learned without especial communications among agents for a simple task by the GAPS approach.

(v) Finally, an approach to reactive motion planning problems of robot manipulators is shown. I treat this problem under uncertain condition where unknown obstacles exist in a problem domain and reactive plans have to be made based on sensed environmental information. This reactive planning mechanism is realized by trial-and-error processes based on learning classifier systems. As simulation results show, it is expected that the autonomous motion planner based on the GAPS can learn complex manipulator motion planning tasks by the reactive planning mechanism and the reinforcement learning algorithm.

Consequently, the usage and its effectiveness of the GAPS approach for especially complex combinatorial optimization problems and practical problems in uncertain problem domains, are illustrated. Obviously, a problem

that is represented in the genotypic problem space, can be adaptively solved by the GAPS approaches. Moreover, as I present the improvement of problem solving performances of the GAPS by extensive mechanisms, it is expected to construct the more evolved GAPS. For example, all of already proposed extended mechanisms concerning to genetic algorithms and learning classifier systems, can be applied to this GAPS.

Acknowledgments

This study would never have been accomplished without the help of many people. The author would like to particularly appreciate Professor Yukinori Kakazu of Hokkaido University for his constant patient instruction and adequate advice on this work.

Deep thanks are also due to Professor Mitsuo Wada and Professor Eiichi Miyamoto of Hokkaido University.

He wishes to express his gratitude to Associate Professor Hiroshi Yokoi of Hokkaido University for their for his profitable discussions and suggestions.

He also wishes to thank to Instructor Keiji Suzuki of Hokkaido University for his valuable discussions.

He wishes to appreciate Associate Professor Sadayoshi Mikami of Hokkaido University for his beneficial discussions.

He really thanks to President Mikio Asai of Hokkaido Women's College for his support of setting up the author's research environment.

The author's thanks go to many staffs of Hokkaido Women's College, especially to Professor Susumu Aiuchi, Professor Masaaki Fujii, Lecturer Mikio Muramatsu, Lecturer Naoki Yuzawa and Lecturer Shin Oozeki for their great helpfulness and encouragement.

List of Results

Papers

1. Kawakami,T., Minagawa,M. and Kakazu,Y. : "Automatic Tuning of 3-D Packing Strategy and Rule-Base Construction using GA," Transactions of Information Processing Society of Japan, Vol.33, No.6, pp.761-768, 1992.(in Japanese)

2. Kawakami, T. and Kakaza,Y. : "A Study on an Autonomous Robot Navigation Problem using a Classifier System", Transactions of the Japan Society of Mechanical Engineers, C, Vol.59, No.564, pp.2339-2345, 1993. (in Japanese)

3. Kawakami, T. and Kakazu,Y. : "Strategy Acquisition of the 3-D Packing Problem in Multiagent Environmment (GA-based Hierarchical Tuning)," Transactions of the Japan Society of Mechanical Engineers, Vol.60, No.577,C, pp.3219-3225, 1994. (in Japanese)

4. Kawakami,T. and Kakazu,Y. : "A Genetic Based Machine Learning System for the Process Planning", Intelligent Engineering Systems Through Artificial Neural Networks, Vol.6, ASME Press, pp.311-318, 1995.

5. Kawakami,T. and Kakaza,Y. : "A Study on Evolutionary Synthesis of Classifier System Architectures by Genetic Algorithms", Transactions of the Japan Society of Mechanical Engineers, 1993. (submitted)

International Conference

1. Kawakami,T. Minagawa,M. and Kakazu,Y. : "Auto Tuning of 3-D Packing Rules Using Genetic Algorithms," in *Proceedings of IEEE International Workshop on Intelligent Robots and Systems '91*, 1991.

2. Kawakami,T. and Kakazu,Y. : "The Autonomous Robot navigator with the Classifier Mechanism", in *Proceedings of First Japan-France Congress of Mechatronics*, pp.163-168, 1992.

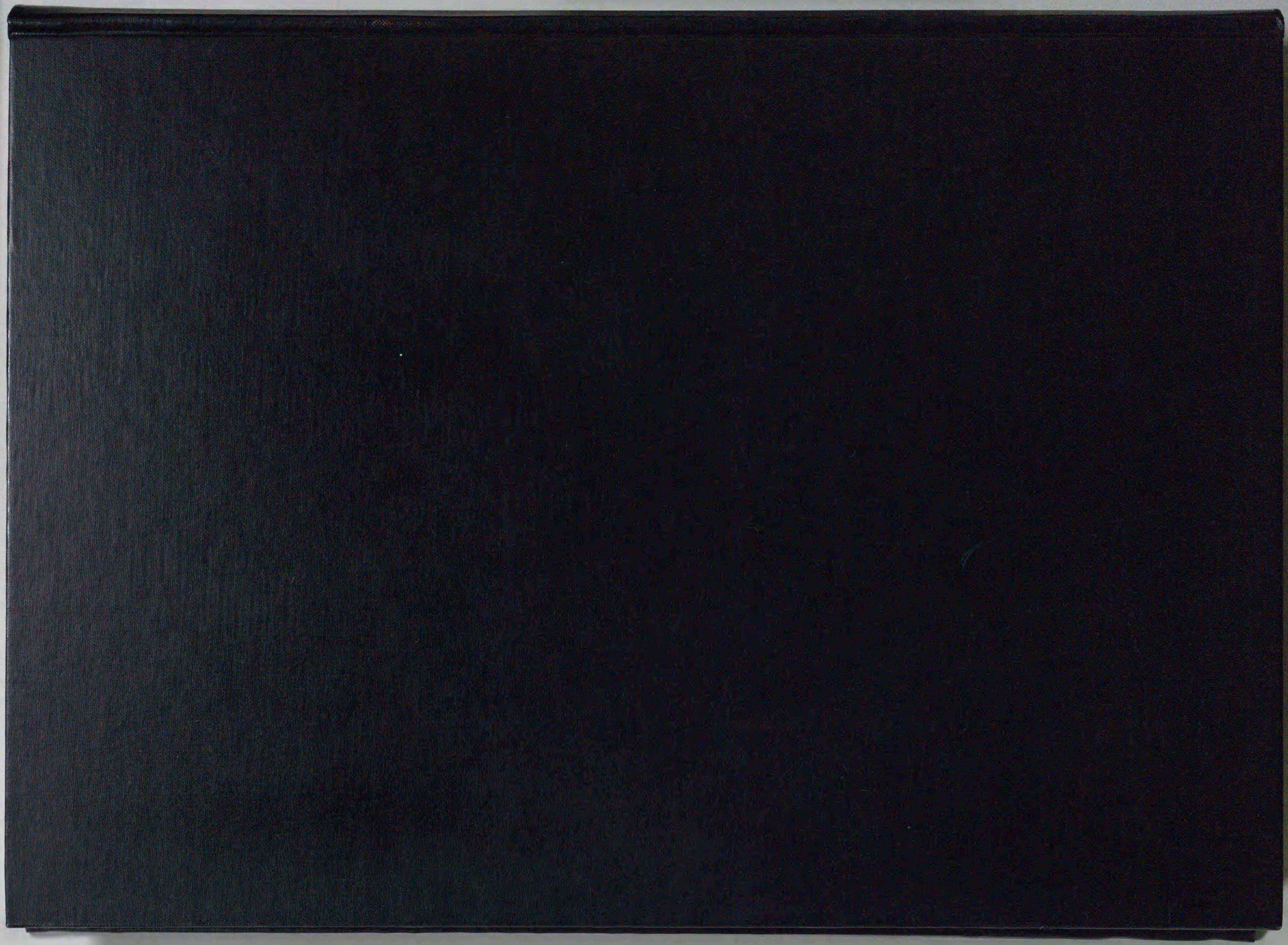
3. Kawakami,T. and Kakazu,Y.: "A GA-based Hierarchical Tuning of the 3-D Packing Strategy in a Multiagent Environment," in *Proceedings of 1994 Japan-U.S.A. Symposium on Flexible Automation*, pp.1319-1326, 1994.

4. Kawakami,T., and Kakazu, Y. : "A Study on Robot Task Planning Problems in Multiagent Environments", in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 2752-2757, 1995.

5. Kawakami, T. and Kakazu, Y.: "A study on Evolutionary Synthesis of Classifier System Architectures," in *Proceedings of IEEE International*

Conference on Evolutionary Computation'96, 1996.(submitted)

7. Kawakami, T. and Kakazu, Y.: "Reactive Motion Planning of Robot Manipulators by CS-based Reinforcement Learning," in *Proceedings of 1996 Japan-U.S.A. Symposium on Flexible Automation*, 1996. (submitted)



inches 1 2 3 4 5 6 7 8

cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Kodak Color Control Patches

© Kodak, 2007 TM: Kodak

Blue Cyan Green Yellow Red Magenta White 3/Color Black

Kodak Gray Scale

C Y M

© Kodak, 2007 TM: Kodak

A 1 2 3 4 5 6 **M** 8 9 10 11 12 13 14 15 **B** 17 18 19