Efficient Program Generation by Genetic Programming

by

Takuya Ito

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Associate Professor Satoshi Sato

School of Information Science
Japan Advanced Institute of Science and Technology

March 1999

Contents

Abstract

Genetic Programming (GP) can generate computer programs automatically without any explicit knowledge for target programs (solution programs). The solution programs are generated by means of selection and some genetic operators. However, GP has a difficulty, which it often takes too much time to generate solution programs. This may be a critical problem when GP generates large scale programs.

The goal of this work is to generate computer programs efficiently by means of the framework of GP. "Efficient" means to reduce the number of generations which is necessary to generate solution programs. To realize this goal, this work improves a genetic operator of GP. There are three genetic operators for GP, crossover, mutation and reproduction. Among these genetic operators, crossover mainly contributes to searching for solution programs. Therefore, this work improves crossover. The normal crossover selects a crossover point randomly so that it breaks building blocks (i.e., effective small program which contributes to improving fitness performance) due to its blind application. To solve this problem, this work proposes four new crossovers

The first crossover is a "depth-dependent crossover" and the second crossover is a "revised depth-dependent crossover". "Depth-dependent" means that node selection probability is determined by the depth of the tree structure. On these crossovers, shallower nodes are more often selected, and deeper nodes are selected rarely. The building blocks can be protected by swapping shallower nodes.

The third crossover is a "non-destructive depth-dependent crossover", which is a combination of the depth-dependent crossover and a "non-destructive crossover". "Non-destructive" means that offsprings of crossover are kept only if their fitness are better than fitness of their parents. This crossover is proposed to solve the program size problem of the depth-dependent crossover.

The fourth crossover is a "self-tuning depth-dependent crossover". On this crossover, each individual of the population has a different depth selection probability and depth selection probability of a selected individual is copied to the next generation. This crossover is proposed to enhance the applicability of the depth-dependent crossover for various GP problems.

This work compares GP performances (i.e., fitness value and the size of generated programs) of the normal crossover with performances of these four crossovers using standard GP problems and an original robot problem. These experimental results clarify that the superiority of the proposed crossovers to the normal crossover.

Furthermore, this work discusses the building block hypothesis, which explains how crossover searches solution programs with a survey of previous works and these experimental results.

Acknowledgments

I would like to thank Professor Satoshi Sato who is my supervisor. He advised me essential issues for research, e.g., how to write a quality paper and how to write a quality proposal.

I wish to thank Professor Masayuki Kimura. He was my supervisor when I was a master student. He advised me how to spend time and how to go ahead with the research. His advises had a great influence on me so as to process research.

I would like to thank Professor Hitoshi Iba for his helpful discussions. He explained me expertise of Genetic Programming to me. His comments were very useful for improving the quality of this thesis. He also gave me a chance to translate Melanie Mithcell's book "an introduction to genetic algorithms" from English to Japanese. This was a great experience for me.

I greatly acknowledge Dr. W. B. Langdon for his useful comments. He advised me expertise of Genetic Programming, especially, crossover.

I would like to thank Professor Nikolay I. Nikolaev. He explained me fitness landscape analysis at EuroGP'98.

I wish to thank Professor Ho Tu Bao. He revised our GP'96 paper and gave me a chance to write our paper of DS'98.

I especially thank Professor Seiji Yamada. He told me some topics of evolutionary robotics at EuroGP'98 and EvoRobot'98.

I would like to thank Dr. Una-May O'Reilly and Dr. Peter J. Angeline. They revised our paper of "Advances in Genetic Programming 3".

I wish to give special thank Mr. So Shisei. He gave me useful comments of hardware when I made a real robot in sub-theme research. Discussions with him about the real robot were so exciting for me.

Chapter 1

Introduction

There are two approaches to generate computer programs. First is a design approach. This means that a user writes computer programs. When we write computer programs, we should know how to generate them and explicit knowledge about target programs. Therefore, the accumulation and systematization of the knowledge are needed in the design approach.

Second is an automatic program synthesis approach. This approach does not require to know how to generate them and explicit knowledge about target programs.

There is Genetic Programming (GP) in automatic program synthesis approach. GP is based on the theory of evolution [Koza, 1992a]. GP treats computer programs directly as its genes and generates them by selection and some genetic operators (e.g., crossover, mutation and so on). First, we prepare (1) the input and the output of the program we want to obtain, and (2) the set of primitive functions (e.g., boolean functions, arithmetic functions and conditional functions) that are available. Then, GP can generate computer programs using selection and some genetic operators.

The algorithm of GP is as follows:

1. Generate randomly M individuals (programs) for an initial population (i.e., 0 generation).
2. Calculate each individual of the population.
3. Repeat the following steps and generate M individuals for a new generation.
   (a) Choose an individual from the present individual using selection (e.g., tournament selection, roulette wheel selection and so on).
   (b) Choose an operator from three genetic operators: crossover, mutation and reproduction.
   (c) If the selected genetic operator is crossover, choose one more individual using selection.
   (d) Apply the selected genetic operator to the selected individual (individuals) and generate a new individual (individuals).
   (e) Insert the generated individual (individuals) as an element (elements) of a new population.

4. If a terminate criterion is fulfilled, go to step 5. Otherwise, replace the present population with the new population (increment one generation) and go to step 2.

5. Show a structure of the best individual as an output of the algorithm.

If the number of generations has reached a maximum number or if the solution program has been found in the middle of the evolution, the terminal criterion is fulfilled.

A structure of the individual (program) is usually a tree structure (LISP S-Expression) [1]. Figure 1.1 shows an example of the tree structure. On the tree structure, leaves indicate variables or constants and other nodes mean functions. In GP, a leaf is called a terminal node and a non-leaf node is called a function node. Genetic operators are applied to the tree structure. There are three important genetic operators.

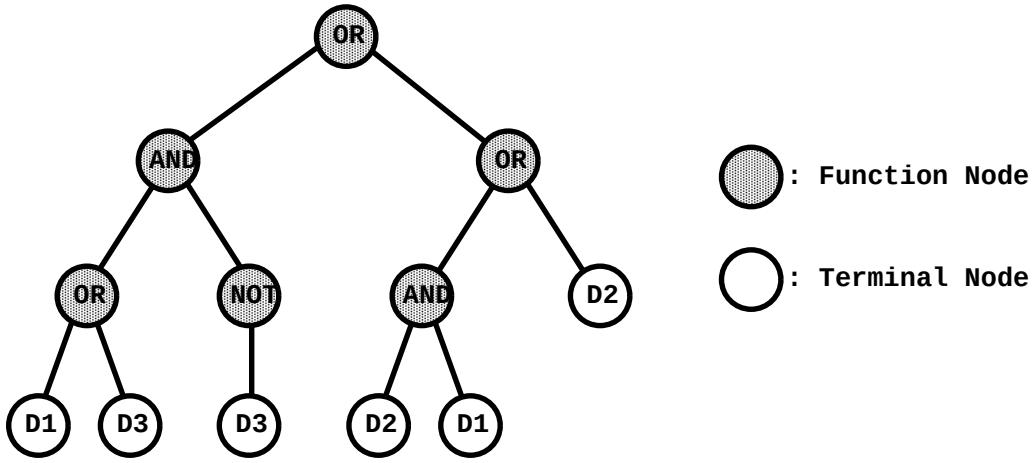

Figure 1.1: Tree Structure

Crossover is an operator which is applied to two individuals. It selects randomly a crossover point (i.e., node) in each parent and exchanges the selected subtrees between two parents to produce new children (Figure 1.2).

Mutation is an operator which is applied to an individual. It selects randomly a mutation point (i.e., node) and replaces the selected subtree with a random generated subtree (Figure 1.3).

Reproduction is an operator which is applied to an individual. It makes a copy of an individual which is chosen by selection without any modifications.

[1] Recently, the linear and the graph structure are also studied to treat as GP's genes [Banzhaf *et al.*, 1998, pp.239–276]

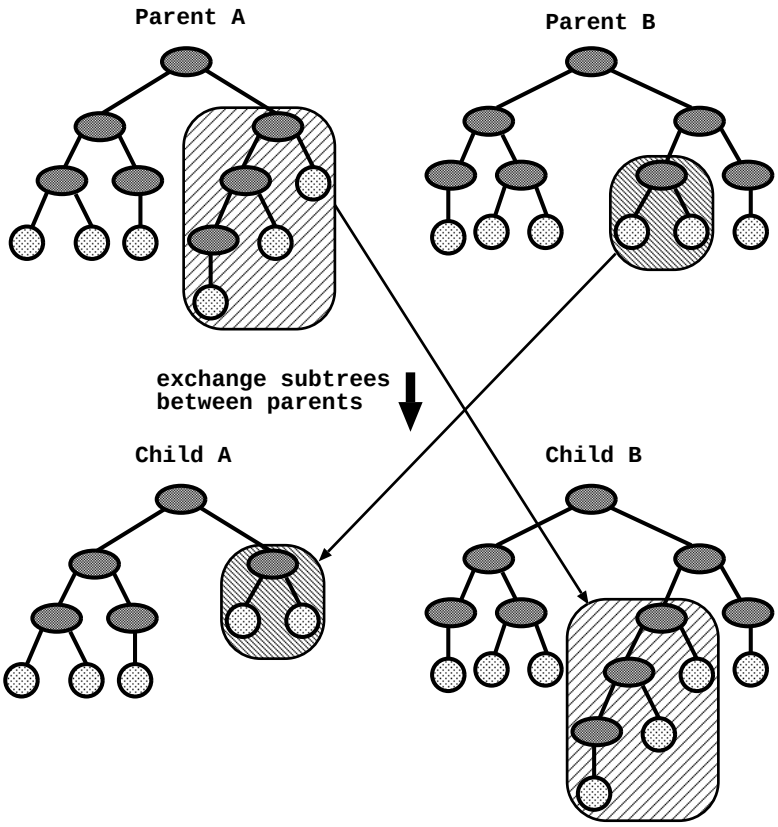

Figure 1.2: Crossover

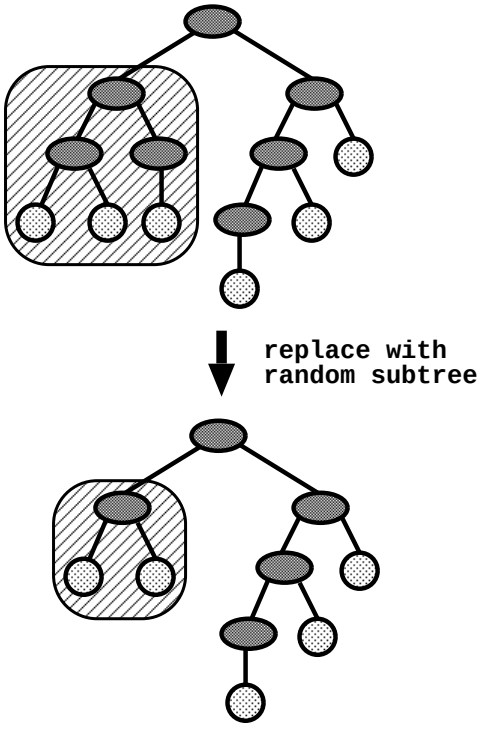

Figure 1.3: Mutation

A process of generation of target programs by GP is explained as follows. In GP, a program is regarded as an individual and is a target of selection. Selection chooses a better

program with higher probability. The better program means a program which has a good fitness. Fitness is an index how good each program is, i.e., how near to target programs [Banzhaf *et al.*, 1998, p. 126]. A better program prospers in the population (i.e., the better program is copied frequently at a change of generations) and an inferior program is reduced by selection and reproduction. However, new programs are not generated by using only selection and reproduction; crossover and mutation work, too. If new programs generated by these two genetic operators are superior, they prosper in the population. After iterating the above process, target programs are expected to be generated.

However, GP has a difficulty because it may take a long time to generate target programs. This may due to be the fact that it does not have explicit knowledge or procedure for target programs. This is a critical problem for GP when we use GP to generate a large scale program (e.g., programs of a word processor, a spread sheet and so on). The simplest solution to this problem is given by incorporating explicit knowledge into the automatic program synthesis approach. However, this solution is exactly the design approach.

The goal of this thesis is to generate computer programs efficiently using the frame work of GP. "Efficient" means that reduction of the number of generations required to generate target programs. For the sake of the above goal, this thesis improves a genetic operator. In GP, there are three important genetic operators, crossover, mutation and reproduction for GP. The role of crossover is to combine two effective parts (building blocks) and to generate a larger effective part. Mutation plays a second role in escaping a local optimum. Reproduction is to copy an individual into the next generation. Among these genetic operators, this thesis improves crossover so as to pursue the efficiency of the program generation. The reason why this thesis focuses on crossover is that crossover is a main genetic operator which searches for a search space [Koza, 1992a, p. 599].

This paper is organized as follows:

Chapter 2 mentions crossover and the building block hypothesis. This hypothesis mentions how crossover searches target programs. It also explains the normal crossover and its drawback.

Chapter 3 proposes two new crossovers to reduce the number of generations which are necessary to generate target programs. The first crossover is a "depth-dependent crossover" and the second crossover is a "revised depth-dependent crossover". "Depth-dependent" means that node selection probability is determined by the depth of the tree structure. On these crossovers, shallower nodes are more often selected, and deeper nodes are selected rarely. The building blocks can be protected by swapping shallower nodes. This chapter compares GP performances (i.e., fitness value and the size of generated programs) of the normal crossover with performances of these two crossovers using standard GP problems and an original robot problem.

Chapter 4 proposes a "non-destructive depth-dependent crossover", which is a combination of the depth-dependent crossover and a "non-destructive crossover". "Non-destructive" means that offsprings of crossover are kept only if their fitness are better than fitness of their parents. This crossover is proposed to solve the program size problem of the depth-dependent crossover. This chapter compares GP performances of the non-destructive depth-dependent crossover with the original depth-dependent crossover. Furthermore, this chapter discusses the growth problem of the tree structure (i.e., the bloating problem) on GP.

Chapter 5 proposes a "self-tuning depth-dependent crossover". On the self-tuning depth-dependent crossover, each individual of a population has a different depth selection probability and depth selection probability of a selected individual is copied to the next generation. This crossover is proposed to enhance the applicability of the depth-dependent crossover for various GP problems.

Chapter 6 discusses crossovers proposed above and the building block hypothesis.

Chapter 7 mentions some conclusions of this thesis and future directions of this work.

Chapter 2

Crossover

This chapter describes crossover which is a main genetic operator in order to search for target programs (solution programs). In GP, solution programs are expected to be generated by combining building blocks and to make larger building blocks.

Section 2.1 focuses on the building block and the building block hypothesis. This hypothesis mentions how crossover searches solution programs. Section 2.2 explains the normal crossover. Section 2.3 describes the approach of this works to improve of crossover. Section 2.4 mentions some related works as for the improvement of crossover.

2.1 Building Block Hypothesis

Some researchers have defined the building block and studied the building block hypothesis of GP. For example, Langdon discussed the following definition:

> **Building block** A pattern of genes in a contiguous section of a chromosome which, if present, confers a high fitness to the individual. According to the building block hypothesis, a complete solution can be constructed by crossover joining together in a single individual many building blocks which where originally spread throughout the population [Langdon, 1998, p. 238].

O'Reilly discussed the building block hypothesis by defining a GP schema [O'Reilly, 1995, pp. 118 – 137]. A schema is a similarity template describing a subset of strings with similarities at certain string positions [Goldberg, 1989, p. 19]. O'Reilly's schema is based on Koza's, which is a set of subtrees which contains specified trees. An example of Koza's schema is $H_1 = \{ (+ (+ 3 4)) \}$. O'Reilly extended Koza's schema by introducing a don't care symbol (or wild card):

> A GP-schema H is a set of pairs. Each pair is a unique S-expression tree or fragment (i.e., incomplete S-expression tree with some leaves as wild cards) and a corresponding integer that specifies how many instances of the S-expression tree or fragment comprise H [O'Reilly, 1995, p. 123].

The fragment is a tree which contains a don't care symbol (#) such as Holland's Genetic Algorithm (GA) schema theorem [Mitchell, 1995, p. 27]. An example of the fragment is $H_2 = \{ (\# (+ 3 4)) \}$. $(+ (+ 3 4))$ and $(- (+ 3 4))$ are the instances of the fragment H_2.

The building block is a kind of schema which contributes to giving a high fitness to the individual. She defined the building block and building block hypothesis for GP by using an analogy with GA's as follows [O'Reilly, 1995, p. 130]:

> **GP building blocks:** Low order, consistently compact GP schemas with consistently above average observed performance that are expected to be sampled at increasing or exponential rates in future generations.
>
> **GP Building Block Hypothesis (BBH):** The GPBBH states that GP combines building blocks, the low order, compact highly fit partial solutions of past samplings, to compose individuals which, over generations, improve in fitness.

According to their definitions, the building block is a part of program which contributes to improving fitness performance. The building block hypothesis is explained as follows. First of all, small building blocks are generated by a random program initialization. The fitness of an individual which has such building blocks is better than that of an individual which does not have building blocks. Thus, the individual which has building blocks can prosper gradually in the population by means of selection. These building blocks are combined and become larger gradually by crossover. Finally, the solution program is generated. This is exactly the building block hypothesis and an explanation why GP can generate the solution program.

2.2 The Normal Crossover

Crossover of GP plays an important role in terms of the generation of a new program. This means that crossover is a main genetic operator which searches for a search space [Koza, 1992a, p. 599]. Mutation, another genetic operator which generates a new program, searches completely blindly. On the contrary, crossover is a genetic operator which combines two building blocks and generates larger building blocks.

If crossover is applied to two individuals which have different building blocks, and then an individual which has both building blocks is generated, the fitness of the individual is supposed to be improved. Therefore, its parent is replaced by a new individual and the new individual prospers in the population. Large building blocks are made when crossover combines small building blocks. The solution program is finally found by iterating the above process.

A simple role of crossover is to combine two building blocks and to construct larger building blocks. When crossover works like this, it might as well be said that crossover works effectively. On the contrary, when crossover breaks building blocks, it works ineffectively and fails.

In this work, Koza's crossover is called a "normal crossover" [Koza, 1992a, pp. 101–105]. The normal crossover selects a node randomly for each individual. The probability of choosing a crossover point (the node selection probability) is set to be a constant value for every node. Therefore, it is likely that the normal crossover might work ineffectively because the building block can be broken by randomness of the normal crossover.

2.3 Improving the Normal Crossover

A solution to the problem of the normal crossover is that a larger structure is selected with a high probability. This idea is realized by making the node selection probability of a shallower node (i.e., a closer node to a root node) higher, and the probability of a deeper node (i.e., a distant node from a root node) lower. Swapping large structures may work well for the protection (encapsulation) of generating building blocks. At early generations, swapping large building blocks does not work advantageously to combine building blocks, because building blocks are small at early generations.

Fortunately, this problem can be solved easily. The node selection probability of a crossover point is dependent on an absolute depth (i.e., a number of nodes from a root node), not on a relative depth of the tree structure. Usually, the size of each individual (each tree structure) is small at early generations, and then each individual grows up and the size of each individual gets greater during the evolution [Langdon and Poli, 1997]. At early generations, an absolute depth of a swapped substructures is not so deep even if a shallower node is selected as a crossover point. Because the tree structure is shallow at early generation. As the evolution proceeds and the tree structures grow up, the absolute size of a swapped substructure becomes deep in proportion to the growth of the tree structure. This thesis calls this type of crossover a "depth-dependent crossover" [Ito *et al.*, 1998b].

2.4 Related Works

Some researches tackled to improve the normal crossover. Their studies are classified into two approaches: the syntax approach and the semantic approach. The depth-dependent crossover belongs to the syntax approach. The syntax approach is applicable without any apriori knowledge as to the solution program, so that it can be applied to various GP problems. On the contrary, the semantic approach is supposed to work effectively because it knows the meanings of the program structure [Iba and de Garis, 1996]. The syntax approaches have many examples, such as node selection approach [O'Reilly and Oppacher, 1994, Harries and Smith, 1997], an intron approach [Nordin *et al.*, 1996] and a brood recombination approach [Tackett, 1994]. The depth-dependent crossover is based on the node selection approach.

2.4.1 The Syntax Approach

O'Reilly and Harries aimed at the probability of the node selection. O'Reilly proposed a height-fair-crossover, in which a crossover point was chosen accordingly to the height of a tree. She divided nodes of a tree into some groups. The selection probability of the group was set to be higher as the number of the nodes in a group were smaller [O'Reilly and Oppacher, 1994]. Harries proposed five depth-based crossovers (SameDepths, DiffDepths, Std/Same, Std/Diff and Half&Half) [Harries and Smith, 1997]. Their experimental results showed that Std/Same outperformed the other GP operators. The operator performance was sensitive to the problem domain. Combinations of operators were more robust. These previous works have shown a certain success to extend crossover in GP. However, they did not aim at the accumulation of building blocks via the depth selection probability. The depth-dependent crossover protected break building blocks and

promoted to build larger building blocks by means of swapping shallower nodes.

Nordin introduced Explicitly Defined Introns (EDI's), which protected genes from destructive crossover [Nordin *et al.*, 1996]. Intron is a part of program which does not affect the fitness of an individual. In other words, intron is a useless part of a program. Thus, even if crossover is applied to such introns, building blocks can be protected. He showed improvement of the performances by means of EDI's in terms of the fitness transition and computational time. In case of the depth-dependent crossover, it protects building blocks by depth selection probability.

Tackett proposed a method to reduce the destructive effect of crossover by means of the blood recombination [Tackett, 1994]. He inferred the observed fact that many animal species (e.g., fish, bird, etc) produce more offsprings than are expected to live. In case of the brood recombination, crossover generates N individuals, and then these individuals are evaluated for the fitness. After sorting by the fitness, best two individuals are selected and other individuals are discarded. Brood recombination is similar to the (μ, λ) selection of Evolutionary Strategies (ES) [Banzhaf *et al.*, 1998, pp. 98–100]. In case of ES, selection has two deterministic parameters, i.e., μ and λ. μ means the number of present and also future parents. λ means the number of offspring. Therefore, both of the blood recombination and the (μ, λ) selection work as an over-production selection.

2.4.2 The Semantic Approach

In case of semantic approach, the operator knows where it should be applied to the program structure because it knows meanings of the program structure. Therefore, the semantic approach can search the solution program effectively. There are some studies of the semantic-based approach. For instance, Iba introduced a subtree value (S-value) and proposed a recombinative guidance by using the S-value [Iba and de Garis, 1996]. He tried to exploit already build structures by adaptive recombination, in which GP recombinative operation is guided by the S-value measure. However, the semantic-based approach might have difficulty. In general, there is no apriori knowledge of the solution program in GP. The semantic approach can be applied to only limited GP problems. For instance, a solution of the even-3-parity problem can be used as the apriori knowledge so as to solve the even-4parity problem. This work chooses the syntax approach because of the above argument against the semantic approach.

Chapter 3

Depth-Dependent Crossover

The standard Genetic Programming (GP) requires huge computational time to search a solution program. This is a critical problem if we apply GP to a real world problem. Therefore, this chapter improves a genetic operator to search effectively. There are three genetic operators, i.e., crossover, mutation and reproduction for GP. Among these genetic operators, crossover mainly contributes to searching for a solution program. Thus, this chapter improves crossover. The normal crossover selects a crossover point randomly, so it destroys building blocks due to its blind application. However, building blocks can be protected by swapping larger substructures. To realize this idea, this chapter introduces two new crossovers: a depth-dependent crossover which applies to a shallower node more often and a revised version of the depth-dependent crossover. This chapter compares GP performances the normal crossover with performances of these depth-dependent crossovers. These experimental results clarify that the superiority of the proposed crossovers to the normal crossover.

3.1 Introduction

Recently, Genetic Programming (GP) has been applied to various applications, e.g., a robot program [Koza, 1991], a multi-agent [Iba, 1997], an image recognition [Iba *et al.*, 1995] and so on. The standard GP requires huge computational time to search for the solution program. This is a critical problem when GP is applied to a large scale problem, such as the automatic generation of a robot control program. Therefore, an effective search method is required. In many previous studies, it is shown that crossover and selection mainly contribute to generating the solution in GP [Koza, 1992a, p. 599]. In crossover and selection, this chapter focuses on crossover and aims at generating computer programs efficiently by improving crossover.

A program structure of GP is generally a tree structure. The normal crossover selects randomly a crossover point (node) regardless its position of the tree structure. Thus, if the normal crossover destroys building blocks, it will degrade the search for the solution program.

A solution to this difficulty is to swap large structures. We can expect that building blocks are protected by swapping larger structures. This chapter realized the "depth-dependent crossover" based on this idea [Ito *et al.*, 1998b] (see Section 2.3). This method is aim for the protection of building blocks and the construction of larger building blocks by crossover which is applied in a shallower node of the tree structure.

This method is expected that number of generation, which is required to search a search space, is reduced. This chapter verifies the effectiveness of the depth-dependent crossover through some experiments of the boolean concept formation problems (i.e., the 11-multiplexor and 4-even parity problem), the ANT problem and a robot problem.

This chapter is organized as follows. Section 3.2 explains a mechanism of the depth-dependent crossover. Section 3.5 describes experiments in several tasks and compares GP performances of the depth-dependent crossover with performances of the normal crossover. Section 3.6 mentions some conclusions.

3.2 The Depth-Dependent Crossover

The idea mentioned in Section 2.3 is realized as the following algorithm:

STEP1. For parent1, determine the depth d for a selected tree.

STEP2. For parent1, select randomly a node of which depth is equal to d in STEP1.

STEP3. For parent2, determine the depth d for a selected tree.

STEP4. For parent2, select randomly a node of which depth is equal to d in STEP3.

STEP5. Swap the nodes chosen in STEP2 and STEP4.

This thesis calls this algorithm a "depth-dependent crossover" (see Figure 3.1).
The depth selection probability is derived by using the following equations:

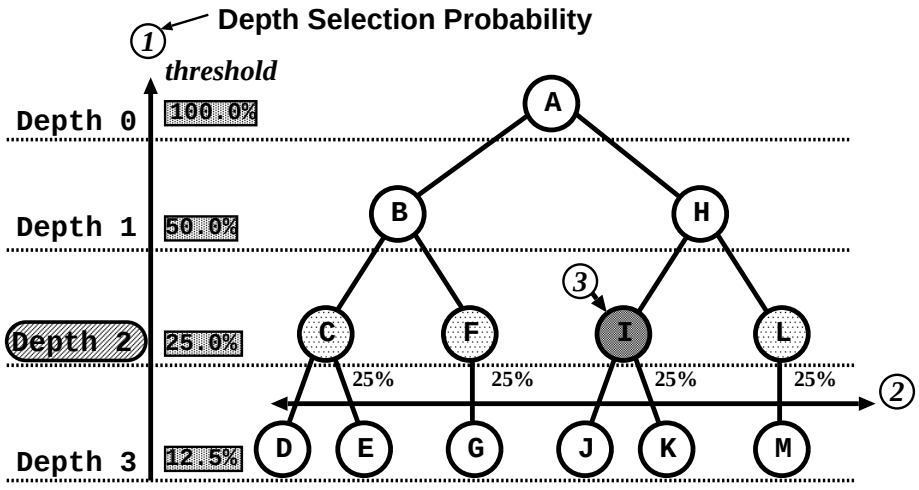

Figure 3.1: The Depth-Dependent Crossover

$$\begin{cases} threshold_i = 1/2^i, & \text{if } i = depth. \\ threshold_i = threshold_{i+1} \times 2, & \\ \qquad\qquad \text{otherwise } i = 0, 1, \cdots, depth - 1 & \end{cases} \tag{3.1}$$

where $threshold_i$ is the depth selection probability at the ith depth, and $depth$ is the depth of the tree. The above equations represent that the depth selection probability

of the root node is 1.0 and the depth selection probability of any other node is half of its parent node's probability (see Figure 3.1). In order to determine the depth d for the depth-dependent crossover in STEP1, a number i is chosen from between 0 and $depth$ in proportion to the $threshold_i$ value. This process is similar to the roulette wheel selection used in Genetic Algorithms.

3.3 The Revised Depth-Dependent Crossover

The principle of the depth-dependent crossover is to increase the node selection probability which swaps larger substructures. The depth-dependent crossover adopts the method which depends on the absolute depth of the tree structure. This method encourages to enlarge the size of the swapped substructures during the evolution.

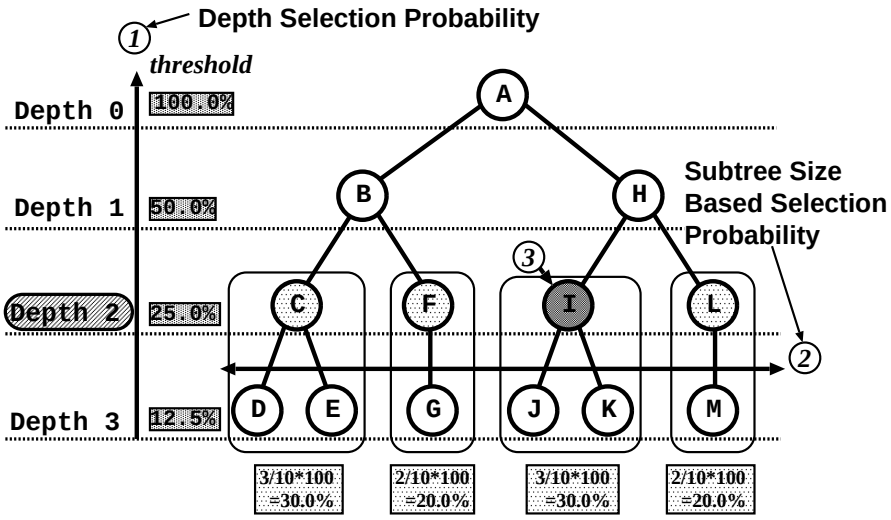

Figure 3.2: The Revised Version of Depth-Dependent Crossover

However, the selection probability of a node is not in proportion to the size of the swapped substructure in this method. For example, the selection probability of node C, F, I and L are the same regardless of their subtree sizes. If the node C is selected as a crossover point, the swapped subtree size is three. On the contrary, if the node L is selected, the size is two. Thus, crossover swaps larger subtrees when the node C is selected as the crossover point than the node L.

This inconsistency is solved by modifying STEP2 and STEP4 of the algorithm of the depth-dependent crossover:

STEP2'. For parent1, select a node in proportion to its subtree size. The depth of the root node of the subtree is equal to d in STEP1.

STEP4'. For parent2, select a node in proportion to its subtree size. The depth of the root node of the subtree is equal to d in STEP3.

This thesis calls this crossover the revised depth-dependent crossover (see Figure 3.2).

3.4 A Difference of the Node Selection Probability of Each Crossover

This section shows the difference of the node selection probability of each node among the normal crossover and the two types of depth-dependent crossover using the tree structure of Figure 3.1.

In case of the normal crossover, each node is selected with same probability. The tree structure has 13 nodes, so that the probability of the node I is $1.0/13 \times 100 = 7.7\%$.

In case of the depth-dependent crossover, the thresholds which are shown in Figure 3.1 are the accumulated values from the deepest node of the tree. Thus, they have to be changed to relative values. The relative probability of the depth 2 is $0.25/(0.125 + 0.25 + 0.50 + 1.00) \times 100 = 13.3\%$. At depth 2, there are 4 nodes which are selected equally, its chance of selection of the node I is $13.3\%/4 = 3.3\%$.

In case of the revised depth-dependent crossover, the selection probability of the node I is based on the size of the subtree. As can be seen in Figure 3.2, the subtree size based selection probability of the node I is $3/10 \times 100 = 30.0\%$. So, the selection probability of the node I is $0.133 \times 0.30 \times 100 = 4.0\%$

Figure 3.3 shows results of the above calculation for all nodes. According to this figure, the node selection probability of a deeper node of the depth-dependent crossover is lower than that of the normal crossover. On the contrary, the node selection probability of a shallower node of the depth-dependent crossover is higher than that of the normal crossover. In case of the revised depth-dependent crossover, the node selection probability of a larger subtree is higher than that of a smaller subtree.

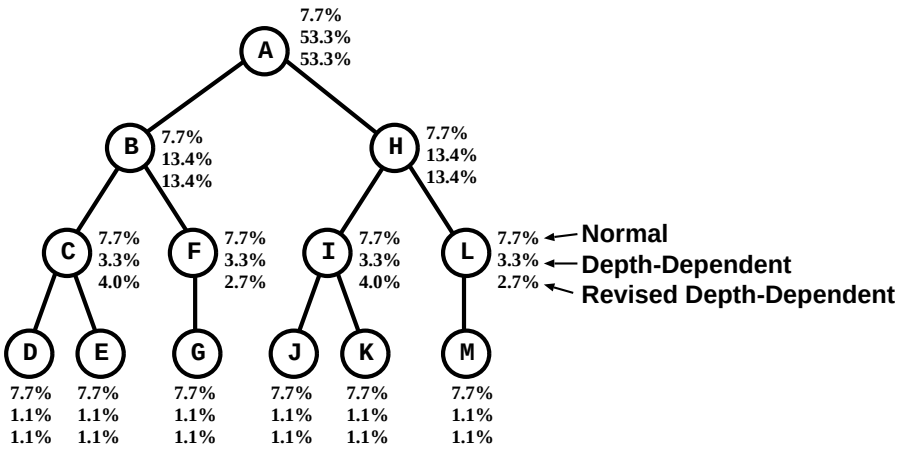

Figure 3.3: Node Selection Probability of Each Crossover

3.5 Experimental Results

Effectiveness of the depth-dependent crossover has been investigated for several problems. This section explains experiments in the Boolean concept formation problems (11MX and 4EVEN), the ANT and a robot problem. Table 3.1 shows the experimental set up. The **NORMAL**, the **DD** and the **RDD** means the normal, the depth-dependent, the revised depth-dependent crossover, respectively. All settings of mutation are random.

For the sake of comparison, all experiments were conducted until a final generation, even if a solution was founded during the evolution. Experimental results are based on the average over twenty runs.

Table 3.1: Experimental Set Up

Setting	Crossover	Mutation
NORMAL	Random	Random
DD	Depth-Dependent	Random
RDD	Revised Depth-Dependent	Random

3.5.1 11MX

The task of the 11MX (11-multiplexor) problem is to generate a boolean function which decodes an address encoded in binary and returns the binary data value of the register at that address. The 11-multiplexor function has three binary-valued address lines (a_0, a_1, a_2) and eight data registers of binary values $(d_0, d_1, \cdots, d_7)$ [Koza, 1992a, p. 170].

Figure 3.4 shows an example of the 11-multiplexor function. In this figure, the function returns the value of d_2 because the output of the address lines is 2 ($a_2 = 0, a_1 = 1, a_0 = 0$). Thus, the output of the function is 0 when d_2 is 0, and it is 1 when d_2 is 1. Equation 3.2 shows this function by boolean symbols.

$$\begin{aligned} f(a_2, \cdots, d_0) \quad = \quad & \bar{a}_2\bar{a}_1\bar{a}_0 d_0 \vee \bar{a}_2\bar{a}_1 a_0 d_1 \vee \bar{a}_2 a_1\bar{a}_0 d_2 \vee \bar{a}_2 a_1 a_0 d_3 \vee a_2\bar{a}_1\bar{a}_0 d_4 \\ & \vee a_2\bar{a}_1 a_0 d_5 \vee a_2 a_1\bar{a}_0 d_6 \vee a_2 a_1 a_0 d_7 \end{aligned} \tag{3.2}$$

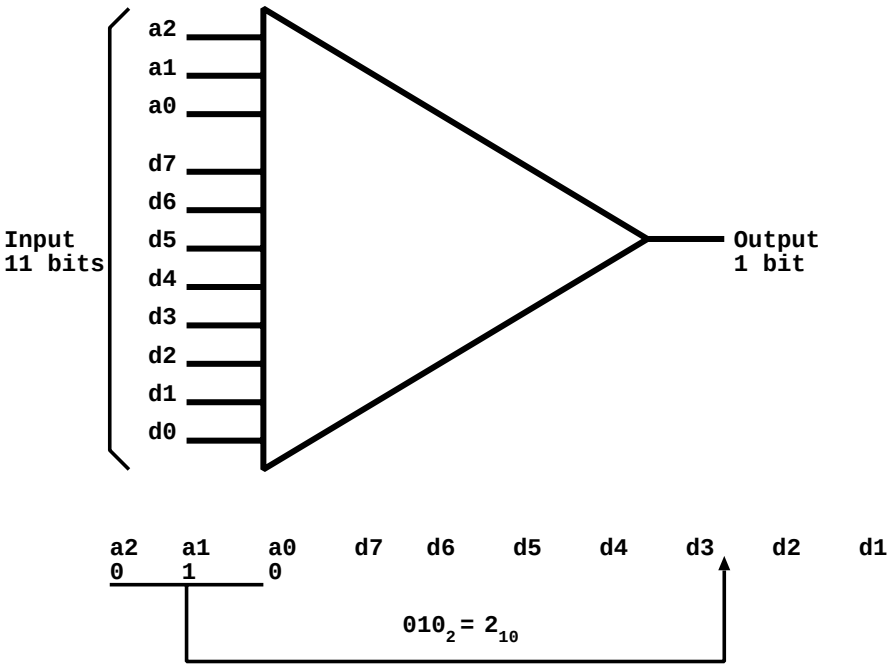

Figure 3.4: The Function of the 11MX Problem

The Fitness function of the 11MX problem is:

$$Fitness = 100.0 - \frac{sum}{numins} \times 100.0 \tag{3.3}$$

where $numins$ is the number of fitness cases (i.e., the number of states, $2^{11} = 2048$), sum is the number of total solutions. This fitness function definition represents an error rate for total inputs, so the smaller value means the better fitness.

Table 3.2 shows the used parameters. The terminal and the function set are indicated in Table 3.3 and 3.4, respectively. All experiments were conducted with these parameters to compare three kinds of crossover settings.

Table 3.2: Parameters for the 11MX Problem

Parameter	Value
Population size	2000
Maximum of generation	20
Maximum depth for new trees	10
Maximum depth after crossover	15
Maximum mutant depth	3
Tree initialization method	Grow
Selection method	tournament
Tournament size	5
Crossover function point fraction	0.1
Crossover any point fraction	0.6
Fitness proportionate reproduction fraction	0.2
Mutation fraction	0.1

Table 3.3: The Terminal Set for the 11MX Problem

ID	Name
0	A0
1	A1
2	A2
3	D0
...	...
10	D7

Table 3.4: The Function Set for the 11MX Problem

ID	Name	Number of Argument
0	AND	2
1	OR	2
2	NOT	2
3	IF	3

Figure 3.5 gives the best and the average fitness on the 11MX problem over twenty runs. The "best" means the fitness of the best individual and the "average" indicates

the average fitness of the population. According to this figure, the **RDD** gave the best performance for the best and the average fitness values. Table 3.5 shows averaged numbers of hits and its standard deviation values at the final generations. If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. As for the hits measure, the **RDD** was also the best among the three cases. These results are examined statistically with the paired t-test [Freund and Wilson, 1992]. Table 3.6 shows results of the t-test. This test confirmed that the **DD** is superior to the **NORMAL**, and the **RDD** is superior to the **NORMAL**. However, this test did not verify that the **RDD** is superior to the **DD** in terms of the best fitness values.

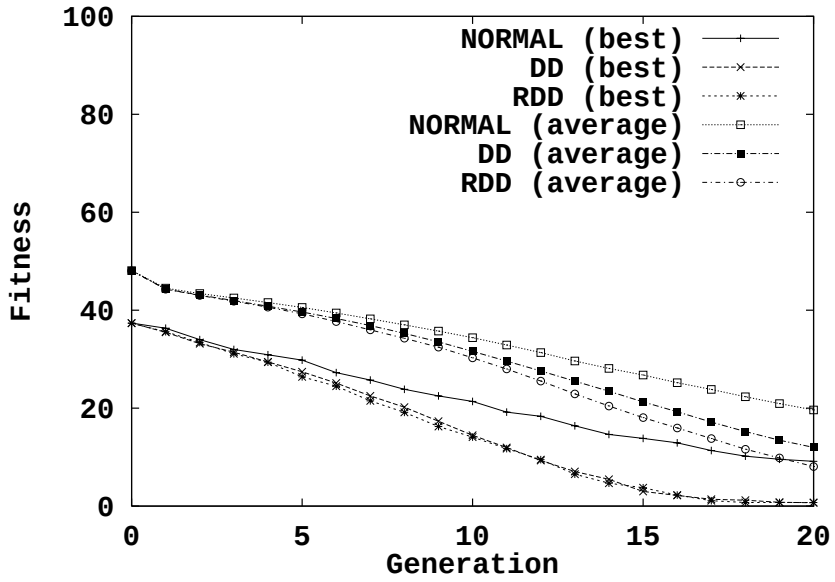

Figure 3.5: Experimental Results, means of twenty runs (11MX). The fitness of the 11MX is an error rate for total inputs. The "best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

Table 3.5: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (11MX). If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. Rank indicates a ranking of three crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.00 (0.00)	3
DD	0.70 (0.46)	2
RDD	0.85 (0.36)	1

Table 3.6: Statistic t for the Best and the Average Fitness Values at the Final Generation (11MX).

Setting	Best	Average
DD (against **NORMAL**)	12.01	10.27
RDD (against **NORMAL**)	9.28	15.69
RDD (against **DD**)	-0.11	6.78

The **DD** and the **RDD** have an effect which swaps deeper subtrees. This works to protect building blocks. It is considered that this effect relates the improvement of the fitness performance. Figure 3.6 shows the average absolute depth of swapped subtrees on the 11MX problem. This figure indicates that the absolute depth of swapped subtrees was shallow for all three crossover settings at the early generation. However, the **DD** (and also the **RDD**) gradually swapped deeper subtrees than the **NORMAL** during the evolution. These phenomena are also observed in the numbers of nodes of the swapped subtrees. Figure 3.7 shows the numbers of the swapped subtrees of the **NORMAL**, the **DD** and the **RDD** on the 11MX problem. This figure shows that the numbers of the swapped subtrees were small for all three crossover settings at the early generation. However, the **DD** (and also the **RDD**) gradually swapped larger subtrees than the **NORMAL** during the evolution.

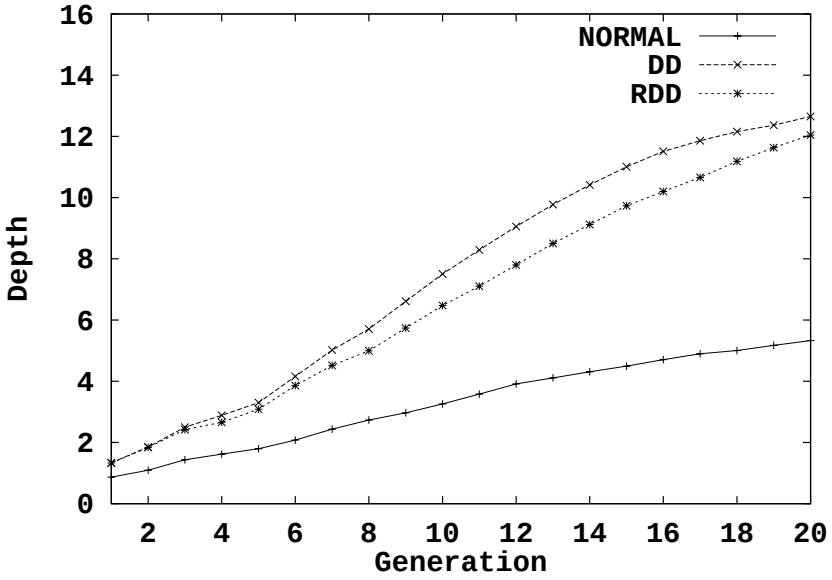

Figure 3.6: Average Absolute Depth of Swapped Tree Structure (11MX)

These phenomena ware related to the tree growth and the fixed depth selection probability (Equation 3.1). In GP, there is a phenomenon that the tree structure grows with the process of the generation. This phenomenon is called *bloat* [Langdon and Poli, 1997]. Angeline clarified that crossover promoted unnecessary bloat by several experimental results [Angeline, 1998]. The depth selection probability of the **DD** is depended on the absolute depth, not on the relative depth and its probability is fixed even if the generation proceeds. The depth of swapped subtrees becomes deep gradually with the growth

of the tree structure. Because a closer node to the root node is selected frequently even if the tree structure becomes deep. Thus, building blocks are rarely broken. As a result, the fitness performance is improved.

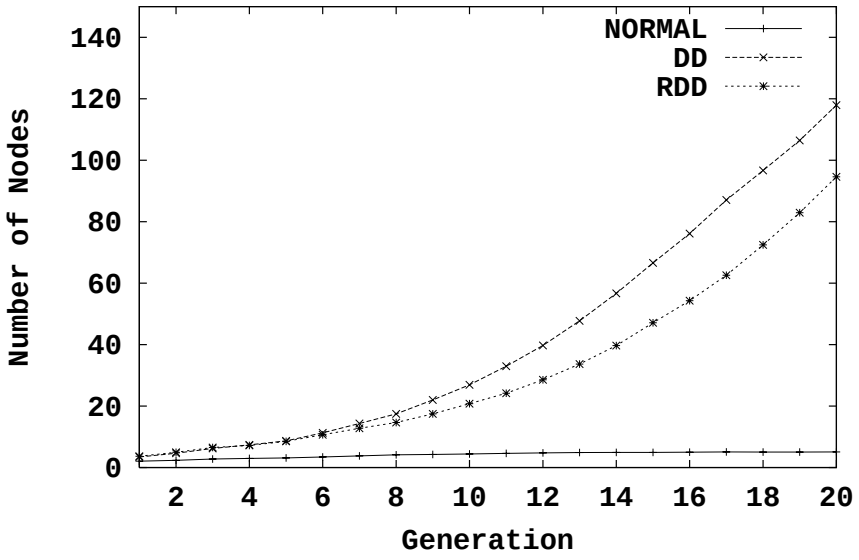

Figure 3.7: Average Number of Nodes of Swapped Tree Structure (11MX)

An advantage of the **RDD** is to suppress the growth of the tree structure. Figure 3.8, 3.9 and 3.10 plot the depth of the tree structure, the number of function node, and the number of terminal node with generations for the 11MX problem, respectively. The growth pattern of the tree depth of the **NORMAL** is shallower than that of the **DD**. The **DD** and the **RDD** show the same pattern (Figure 3.8). The number of the function node (and also the terminal node) of the **DD** (and also the **RDD**) is greater than that of the **NORMAL** (Figure 3.9 and 3.10). The algorithm of the **DD** (and also the **RDD**) selects a shallower node frequently as a crossover point. Therefore, it is frequent to occurred the phenomenon that a small tree on the shallower node is replaced with a large subtree. In this case, the fitness of an individual that received the large subtree (individual 1) is improved because there are large building blocks in the received subtrees. On the contrary, the fitness of an individual that received the small subtree (individual 2) is not improved because the size of building blocks in the received subtree are small. As a result, the size of the tree structure evolves to large because individual 1 prospers in the population and individual 2 is reduced. Thus, it is considered that the selection pressure (i.e., the pressure which makes the size of the tree structure become large) worked in case of the **DD** (and the **RDD**) [Soule *et al.*, 1996]. The **DD** and the **RDD** induced the bloating phenomenon easily by means of the above reason.

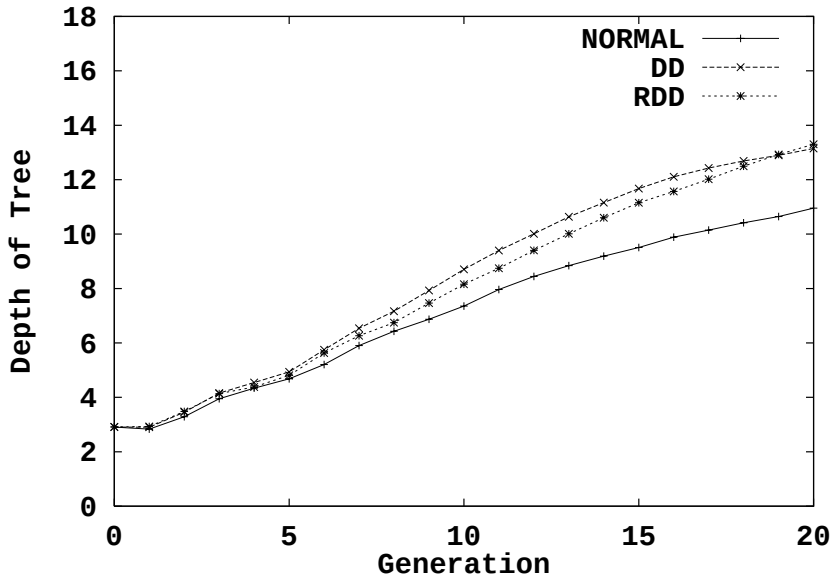

Figure 3.8: Depth of Tree (11MX)

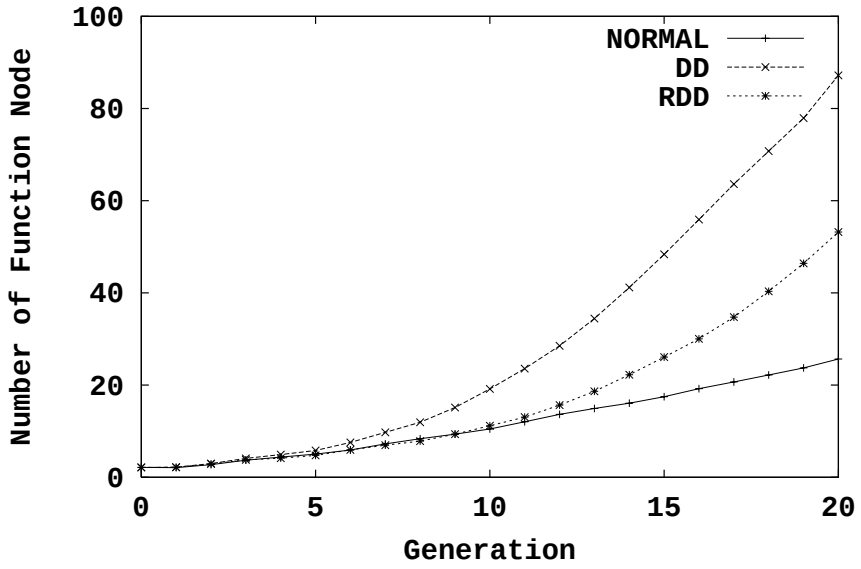

Figure 3.9: Number of Function Node (11MX)

Compared with the number of the function node (and also the terminal node) of the **DD**, that of the **RDD** was small. It is considered that this difference is brought from the modification of the STEP2 of the algorithm of the **DD**. In case of the algorithm of the **DD** (Figure 3.1), a node in the crossover depth d which is determined in the STEP1 is selected randomly so that a small subtree is replaced frequently with a large subtree. If the tree structure which received the large subtree is superior in terms of the fitness to the tree structure which received the small subtree, it can be considered that the size of the tree structure evolved to large size. In case of the **RDD**, the algorithm is made to swap large subtrees in proportion to the size of subtrees (Figure 3.2) and each parent swaps a large subtree together. Thus, it is rarely occurred that a small subtree is swapped by

a large subtree. Therefore, the **RDD** could generate small tree structures than the **DD** because the **RDD** controls accurately the size of the tree structure.

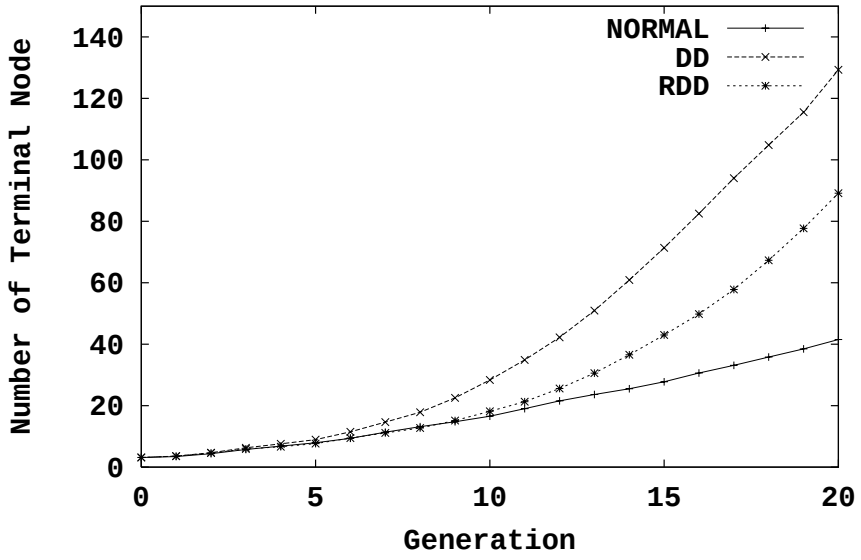

Figure 3.10: Number of Terminal Node (11MX)

3.5.2 4EVEN

Effectiveness of the depth-dependent crossover is also verified for the even-4parity (4EVEN) problem. The 4EVEN (even-4-parity) problem is to generate a Boolean function which returns `T` if an even number of its Boolean arguments are `T`, and otherwise returns `NIL` [Koza, 1994a, p. 157]. For example, Figure 3.11 shows the output is 1 (`T`) for inputs of 1 (`T`), 0 (`F`), 1 (`T`) and 0 (`F`).

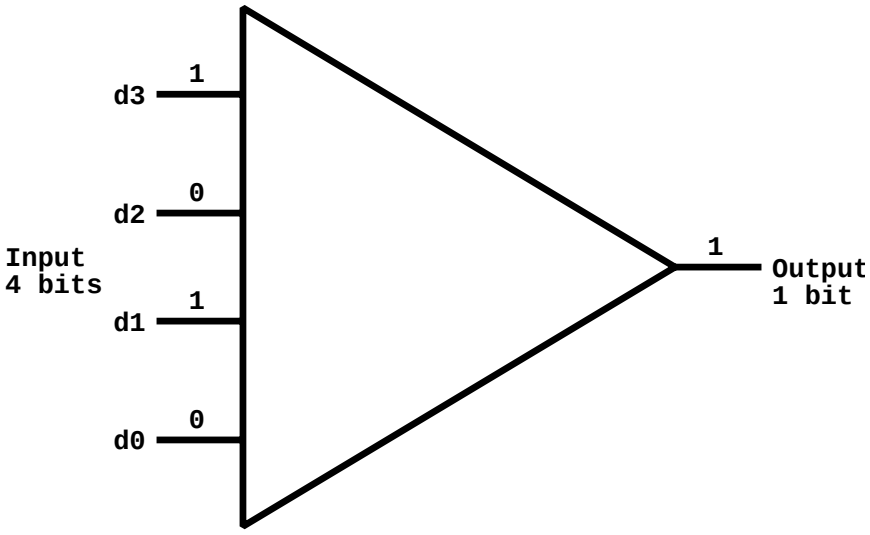

Figure 3.11: The Function of the Even-4-Parity

The fitness of the 4EVEN problem is also an error rate for inputs i.e., the same as that of the 11MX problem (Equation 3.3). Note, $numins$ is $2^4 = 16$.

The used parameters are shown in Table 3.7. For the sake of comparison of three kinds of crossovers, all experiments were conducted with these parameters. The terminal and the function set are indicated in Table 3.8 and 3.9, respectively.

Table 3.7: Parameters for the 4EVEN Problem

Parameter	Value
Population size	2000
Maximum of generation	24
Maximum depth for new trees	6
Maximum depth after crossover	12
Maximum mutant depth	4
Tree initialization method	Grow
Selection method	Tournament
Tournament size	5
Crossover function point fraction	0.1
Crossover any point fraction	0.6
Fitness proportionate reproduction fraction	0.2
Mutation fraction	0.1

Table 3.8: The Terminal Set for the 4EVEN Problem

ID	Name
0	`D0`
1	`D1`
2	`D2`
3	`D3`

Table 3.9: The Function Set for the 4EVEN Problem

ID	Name	Number of Argument
0	`AND`	2
1	`OR`	2
2	`NAND`	2
3	`NOR`	2

Figure 3.12 shows the best and the average fitness values on the 4EVEN problem. According to this figure, the **DD** gave the best performance on the best and the average fitness values. Table 3.10 shows average numbers of hits and its standard deviation at the final generations over twenty runs on the 4EVEN problem. By using the **DD**, the solution program was acquired for all twenty runs. On the contrary, the solution programs was not acquired in case of the **NORMAL**. Table 3.11 shows results of the paired t-test about the best and the average fitness values. According to this test, it has confirmed that the **DD** is better than the **NORMAL**, and the **RDD** is better than the **NORMAL** on the best and the average fitness values. However, it has not verified that the **RDD** is

superior to the **DD** in the average fitness values. The 4EVEN problem is a sort of the boolean function formation problem the same as 11MX problem. The 4EVEN problem is similar in character to the 11MX problem. The **DD** and the **RDD** are suitable for the boolean function formation problems based on the experimental results of the 11MX and the 4EVEN problem.

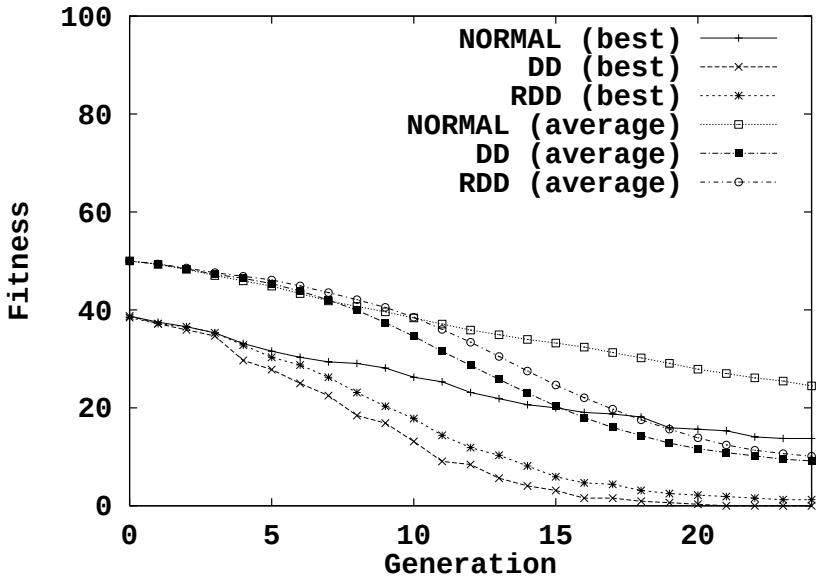

Figure 3.12: Experimental Results, means of twenty runs (4EVEN). The fitness of the 4EVEN is an error rate for total inputs. The "best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

Table 3.10: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (4EVEN). If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. Rank indicates a ranking of three crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.00 (0.00)	3
DD	1.00 (0.00)	1
RDD	0.80 (0.40)	2

Table 3.11: Statistic t for the Best and the Average Fitness Values at the Final Generation (4EVEN).

Setting	Best	Average
DD (against **NORMAL**)	11.00	19.29
RDD (against **NORMAL**)	8.31	12.72
RDD (against **DD**)	-2.18	-1.32

3.5.3 ANT

The depth-dependent crossover were experimented on the ANT problem which is one of a standard problem on GP. The ANT problem is the task of navigating an artificial ant so as to find all 89 foods lying along an irregular trail on 32×32 world (Figure 3.13) [Koza, 1992a, p. 54]. The ant's goal is to traverse the entire trail (thereby eating all of the foods) within a limited energy. The ANT can go ahead, turn to the right and the left. Each motion consumes one energy (total energy is 400). The ANT can detect a food which is in the front of the ANT.

The fitness of ANT defined as follows:

$$Fitness = (foods - rawfitness)/foods \tag{3.4}$$

where, $foods$ is total number of foods, 89 and $rawfitness$ is the total number of foods of which the ANT collected.
In other words, if the ant could not eat any foods, the fitness is 1.0 and if the ant could eat all foods within the limited energy, it is 0.0. The smaller value means the better fitness.

The used parameters are shown in Table 3.12. All experiments were conducted with these parameters to compare three kinds of crossovers. Table 3.13 and 3.14 show the terminal and the function set, respectively. The ANT commands (i.e., `RIGHT`, `LEFT` and `FORWARD`) are assigned to the terminal set. These commands are 0-argument functions.

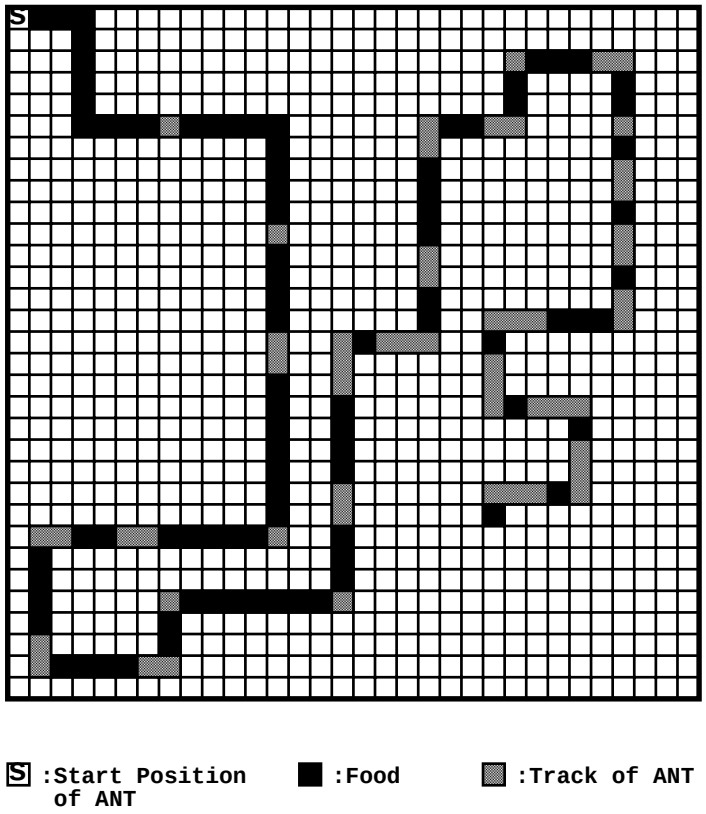

Figure 3.13: The ANT Problem

Table 3.12: Parameters for the ANT Problem

Parameter	Value
Population size	2000
Maximum of generation	49
Maximum depth for new trees	10
Maximum depth after crossover	17
Maximum mutant depth	3
Tree initialization method	Grow
Selection method	Tournament
Tournament size	5
Crossover function point fraction	0.1
Crossover any point fraction	0.6
Fitness proportionate reproduction fraction	0.2
Mutation fraction	0.1

Table 3.13: The Terminal Set for the ANT Problem

ID	Name	Function
0	`RIGHT`	Turns to the right.
1	`LEFT`	Turns to the left
2	`FORWARD`	Moves forward with the present ANT's direction

Table 3.14: The Function Set for the ANT Problem

ID	Name	Function
0	`(If-Food-Ahead p0 p1)`	Evaluates `p0` when a food is in frot of the ANT, otherwise evaluate `p1`
1	`(Prog2 p0 p1)`	Evaluates sequentially all the argument forms and returns the value of the second argument (`p1`).
2	`(Prog3 p0 p1 p2)`	Evaluates sequentially all the argument forms and returns the value of the third argument (`p2`).

Figure 3.14 shows the best and the average fitness for the generation on the ANT problem. As can be seen in this figure, the **NORMAL** gave the best performance on the best fitness value. There is no difference in the average fitness value among the **NORMAL**, the **DD** and the **RDD**. These experimental results are verified statistically. Table 3.16 shows results of t-test. According to this table, there was no difference in the fitness performance between the **DD** (also the **RDD**) and the **NORMAL**. Table 3.15 shows average numbers of hits and its standard deviation at the final generations over twenty runs on the ANT problem. According to this table, the **NORMAL** was superlative about the generation ability of solution programs among three crossover settings.

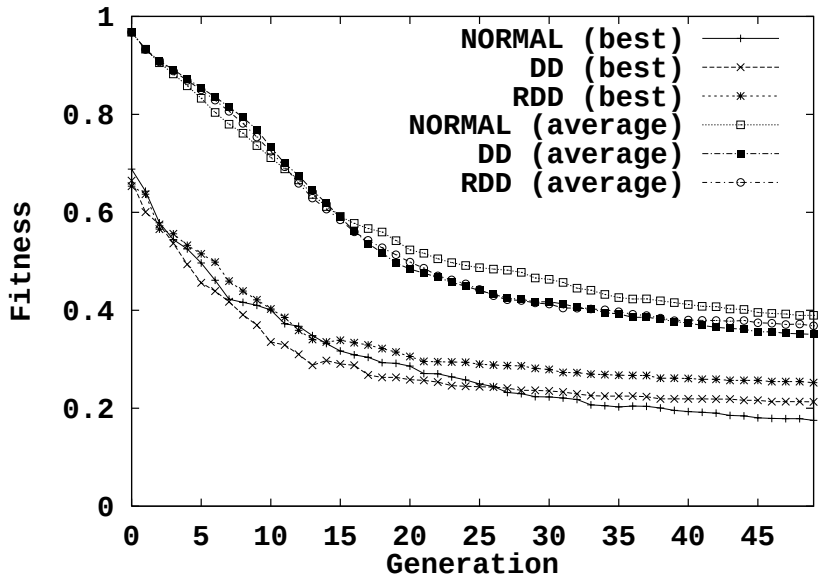

Figure 3.14: Experimental Results, means of twenty runs (ANT). The fitness of the ANT is a probability for which the ant could not eat 89 foods. The "best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

Table 3.15: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (ANT). If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. Rank indicates a ranking of three crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.15 (0.36)	1
DD	0.10 (0.30)	2
RDD	0.00 (0.00)	3

Table 3.16: Statistic t for the Best and the Average Fitness Values at the Final Generation (ANT).

Setting	Best	Average
DD (against **NORMAL**)	-0.96	1.14
RDD (against **NORMAL**)	-1.31	0.30
RDD (against **DD**)	-0.62	-0.40

3.5.4 Robot

This section verified effectiveness of the depth-dependent crossovers on a robot problem. This is to show the feasibility of the depth-dependent crossovers for a real-world task.

The Simulated Robot

An autonomous robot simulator have been constructed (Figure 3.15). The model of this robot is a behavior-based robot[Maes, 1993]. In this simulator, there are only five types

of objects, i.e., the robot, a target object, a station, wall and obstacles in the robot's work space (Figure 3.16).

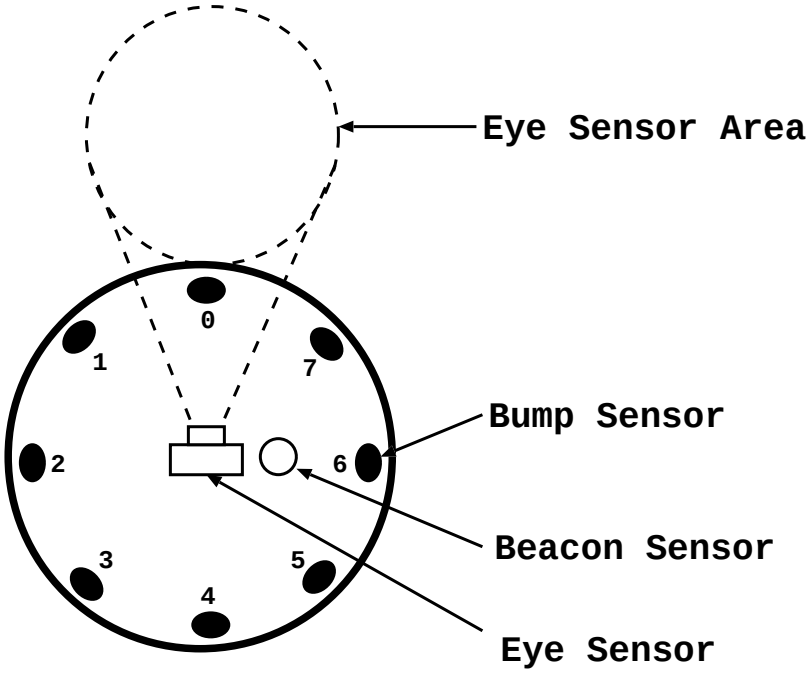

Figure 3.15: The Simulated Robot

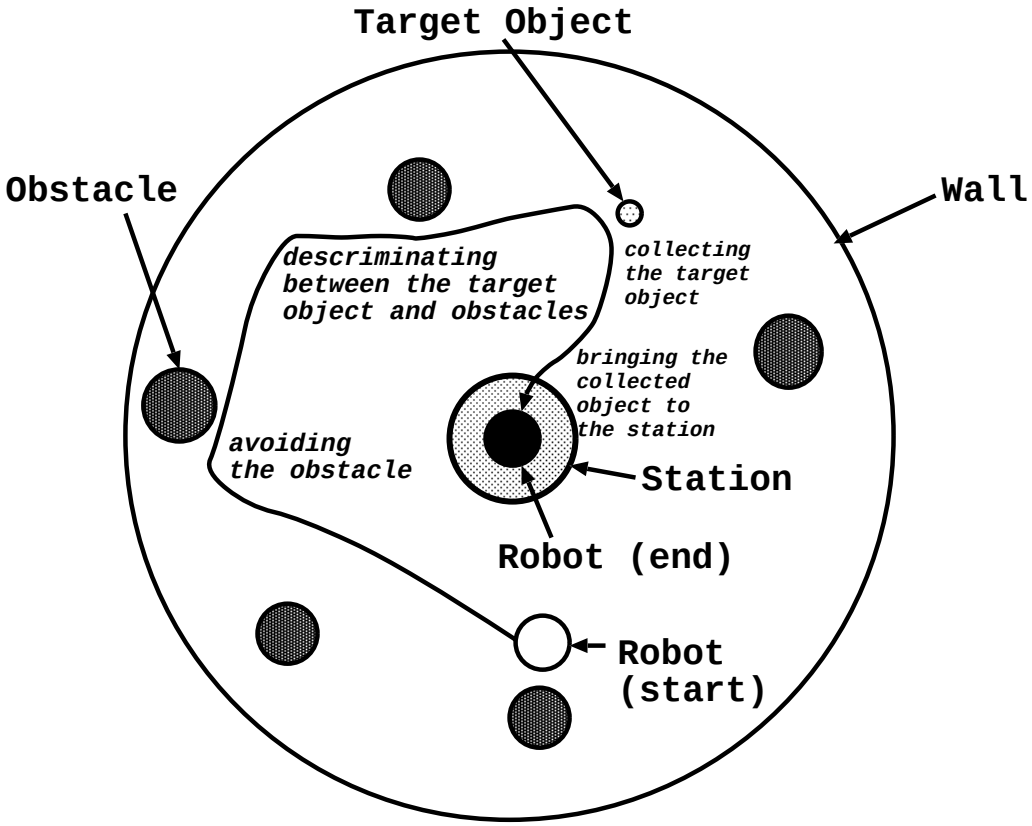

Figure 3.16: The Workspace of the Robot

The simulated robot has three types of sensors. They are eight bump sensors, an eye sensor and a beacon sensor (Table 3.17).

Table 3.17: Sensors of the Simulated Robot

Type	Number	Function
Bump Sensor	8	Reports whether the robot bumps an object or not.
Eye Sensor	1	Reports the IDs of objects within the eye-sensor area.
Beacon Sensor	1	Reports the distance between the robot and the station when the station is within the robot's sensor range.

The motivation of constructing the above robot simulator is derived from a following design principle: this robot simulator is constructed so as to close to a real world as much as possible. In this sense, actual experimental settings are:

1. The robot's coordinates are arbitrary floating point numbers, but not grid based.
2. The robot commands are taken from a real robot.

Task Overview

The robot task is to collect an object. The robot can discriminate between the target object and obstacles by means of their colors. This task consists of following subtasks:

1. avoiding obstacles
2. discriminating between the target object and obstacles
3. collecting the target object
4. bringing the collected target object to the station in the limited time

Terminal Set

As commonly used in most GP applications [Ito *et al.*, 1996b], the robot commands were considered as terminals, i.e., 0-argument function (Table 3.18). When each robot command is evaluated, one time-step is consumed and each command is evaluated until the total number of time-steps reaches a certain limitation, or the robot brings the target object to the station correctly. The maximum number of time-steps is set to be 300.

Table 3.18: Terminal Set (Robot)

ID	Type	Function
$0 \sim 5$	**GF**	Moves forward with the robot speed in its robot's direction. The robot speed is assumed to be constant.
6	**TR**	Makes the robot's body turn 10° to the right.
7	**TL**	Makes the robot's body turn 10° to the left.
8	**MCR**	Makes the eye sensor turn 10° to the right.
9	**MCL**	Makes the eye sensor turn 10° to the left.
10	**GP**	Grasps the target object within the robot's eye sensor area.

The robot does not move when **TR**, **TL**, **MCR**, **MCL** and **GP** commands are evaluated. If there is only one **GF** command in the terminal set, the robot will come to a stop too soon. This situation is called a deadlock. To prevent this deadlock situation, five **GF** commands are included in the terminal set.

Function Set

In order to make the robot respond to its sensor inputs, the 14 functional nodes were introduced (Table 3.19).

Table 3.19: Function Set for the Robot Problem

ID	Type	Function
0∼7	**(BS***ID* **p0 p1)**	Evaluates p0 if the bump*ID*-sensor reports collision, p1 otherwise.
8	**(AS p0 p1 p2)**	Evaluates p0 if the robot approaches the station, p1 if the robot goes away from the station, and p2 if the station is not within the beacon sensor range.
9	**(EWL p0 p1)**	Evaluates p0 if the wall within the eye-sensor area, p1 otherwise.
10	**(EBC p0 p1)**	Evaluates p0 if the obstacle is within the eye-sensor area, p1 otherwise.
11	**(ERC p0 p1)**	Evaluates p0 if the target object within the eye-sensor area, p1 otherwise.
12	**(ESN p0 p1)**	Evaluates p0 if the station is within the eye-sensor area, p1 otherwise.
13	**(PROG2 p0 p1)**	Evaluates sequentially two argument forms and returns the value of the second argument (p1).

Fitness Function

The following fitness function is used for this robot task:

$$Fitness = \begin{cases} dist(S, R_t) & \text{if the robot has the} \\ & \text{target object in hand} \\ dist(S, RC_t) + dist(R_t, RC_t) & \text{otherwise} \end{cases} \tag{3.5}$$

where S is the position of the station, R_t is the the position of the robot at t time-step, RC_t is the the position of the target object at t time-step, $dist(x, y)$ is the Euclidean distance between x and y.

Note, this fitness function definition does not necessarily represent the hardness of the problem. For instance, this definition does not care whether there is obstacles between the robot and the station or not. If there are not obstacles between the robot and the station, the robot searches only the station. On the contrary, if there is obstacles between the robot and the station, the robot have to avoid them and search the station. The latter situation is more difficult for the robot to accomplish the task than the former situation. However, the above fitness function cannot distinguish these two situations. When the

fitness function can distinguish the hardness of the problem, GP can evolve the robot programs easily, however, this can be a sort of heuristics. Less heuristics are better for realizing general automatic generation of programs. Thus, the above fitness function has been chosen.

Experimental Result on the Robot Problem

Figure 3.17 plots the best and the average fitness values for the robot problem. Table 3.20 depicts the used parameters.

Table 3.20: Parameters for the Robot Problem

Parameter	Value
Population size	500
Maximum of generation	49
Maximum depth for new trees	5
Maximum depth after crossover	15
Maximum mutant depth	3
Tree initialization method	Grow
Selection method	Tournament
Tournament size	5
Crossover function point fraction	0.1
Crossover any point fraction	0.6
Fitness proportionate reproduction fraction	0.2
Mutation fraction	0.1

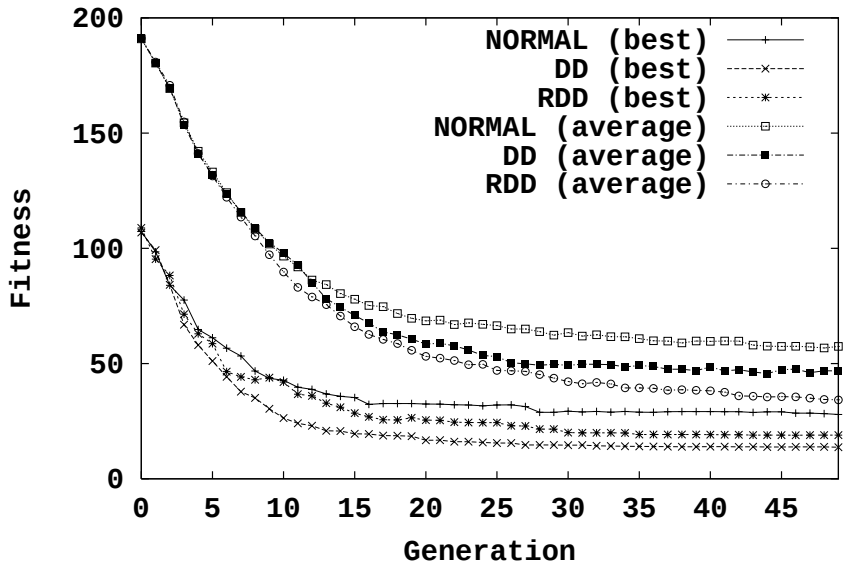

Figure 3.17: Experimental Results, means of twenty runs (Robot). The fitness of the Robot is derived from Equation 3.5. The "best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

According to Figure 3.17, the **DD** gave the best performance on the best fitness value. As for the average fitness value, the **RDD** gave the best performance. Table 3.21 shows the averaged numbers of hits at the final generations over twenty runs. By using the **RDD**, the solution program was acquired for all twenty runs.

Table 3.22 shows the result of t-test. It has confirmed that **DD** was superior to the **NORMAL** in terms of the best and the average fitness value. The **RDD** was better than the **NORMAL** on the average fitness value. However, it has not verified that the **RDD** was superior to the **NORMAL** in terms of the best fitness value.

Table 3.21: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (Robot). If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. Rank indicates a ranking of three crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.80 (0.40)	3
DD	0.95 (0.22)	2
RDD	1.00 (0.00)	1

Table 3.22: Statistic t for the Best and the Average Fitness Values at the Final Generation (Robot)

Setting	Best	Average
DD (against **NORMAL**)	3.34	2.40
RDD (against **NORMAL**)	1.41	4.33
RDD (against **DD**)	-1.53	2.90

3.6 Conclusion

This chapter proposed the depth-dependent crossover and its revised crossover to improve the ability about generation of solution programs. This chapter verified effectiveness of the depth-dependent crossovers by means of several experiments. According to the experiments, following points made clear:

1. For Boolean problems, two types of the depth-dependent crossovers gave better performance than the normal crossover.

2. Both of the depth-dependent crossovers and the revised one worked as a protection against the destructive crossover.

3. The revised depth-dependent crossover controlled the size of a GP tree.

Both of the depth-dependent crossovers and the revised one were superior to the normal crossover in terms of the fitness performance, however, they generated larger programs than that by the normal crossover. The next chapter tackled to solve the program size problem of the depth-dependent crossover by means of introducing non-destructive crossover [Ito *et al.*, 1998a].

Chapter 4

Non-Destructive Depth-Dependent Crossover

In the previous chapter (chapter 3), the depth-dependent crossover was proposed for GP. The purpose was to solve the difficulty of the blind application of the normal crossover, i.e., building blocks are broken unexpectedly. In case of depth-dependent crossover, the depth selection probability was varied according to the depth of a node. However, the depth-dependent crossover did not work very effectively as generated programs became larger. A large size program has three drawbacks: 1. the time necessary to measure their fitness often dominates total processing time, 2. huge memories are required to evolve programs, 3. analyzing generated programs is difficult. To overcome these difficulties, this chapter introduce a non-destructive depth-dependent crossover [Ito *et al.*, 1998a], in which each offspring is kept only if its fitness is better than that of its parent. This chapter compares GP performances of the original depth-dependent crossover with performances of the non-destructive depth-dependent crossover to show the effectiveness of the approach. These experimental results clarify that the non-destructive depth-dependent crossover produces smaller programs than the original depth-dependent crossover.

4.1 Introduction

The previous chapter has proposed the depth-dependent crossover to improve effectiveness of crossover for accumulating building blocks via the depth selection probability (Chapter 3). In case of depth-dependent crossover, the depth selection probability was varied according to the depth of a node. The depth-dependent crossover contributed to the reduction of the number of generations for the evolution on boolean concept formation problems. However, there seemed to be a difficulty that the depth-dependent crossover generated very large size programs. A large size program has following three drawbacks:

1. The time necessary to measure their fitness often dominates total processing time.
2. Huge memories are required to evolve programs.
3. Analyzing generated programs is difficult.

At first, a large program needs huge computational time to evaluate it's fitness. This is a critical problem for GP. In GP, evaluation of programs consumes computational time

fairly. Second, a large program needs huge memories. If GP generates a small program, we can save the expenditure of money on computer memories. Third, if a generated program is small, we can analyze it easily. Due to above reasons, a small size program is needed for GP.

This chapter tackles the program size problem of the depth-dependent crossover. In order to solve this problem, this chapter adopted the "non-destructive crossover (NDC)" to reduce the program size. By this method, each offspring, i.e., a new tree resulting from crossover, is kept only if its fitness is better than that of its parent [Soule and Foster, 1997].

There have been several related studies which tackled the problem of GP program size. Soule has considered that shorter programs tend to show better generalization performance than longer programs. Thus he has studied the selective pressure by penalizing longer programs [Soule *et al.*, 1996]. This method appeared to be effective in bounding the programs' size.

Kinnear added the inverse size of generated programs to the fitness measure [Kinnear, 1993]. However, its fitness measure had the size factor (sf) by which the size of the program was multiplied for addition. We cannot know in advance the exact value of the factors. On the contrary, the NDC requires neither semantic heuristics nor problem-dependent parameters. That means the NDC works purely syntactically.

Iba has introduced a method for controlling program (tree) growth, which used an MDL (Minimum Description Length) principle to define GP fitness functions [Iba *et al.*, 1994]. However, MDL-based fitness functions could not be applied to every kind of problem to be solved by GP. To use MDL-based fitness functions, trees had to have the two characteristics, i.e., "size-based performance" (the more the tree grows, the better its performance is) and "decomposition" (the fitness of a substructure is well-defined itself). However, the NDC can be applied every kind of problem.

This chapter is organized as follows. Section 4.2 described the non-destructive depth-dependent crossover. Section 4.3 shows several experimental results in several tasks and compares the performances of the original depth-dependent crossover with performances of the non-destructive depth-dependent crossover. Some conclusions are given in Section 4.4.

4.2 Non-Destructive Depth-Dependent Crossover

The previous chapter observed that the depth-dependent crossover as well as the revised depth-dependent crossover produced a much larger program than the normal crossover [Ito *et al.*, 1998b]. To solve this difficulty, this chapter has introduced the "non-destructive crossover" based on [Soule and Foster, 1997]. The NDC is a crossover, in which each offspring is kept only if its fitness is better than that of its parents. One benefit of using the NDC is that the program growth is reduced to only the growth necessary for improving program fitness [Soule and Foster, 1997, p. 314]. Thus, this chapter uses the non-destructive crossover to reduce the problem growth on the depth-dependent crossover. This chapter calls this type of crossover a "non-destructive depth-dependent crossover" [Ito *et al.*, 1998a]. The algorithm of the non-destructive depth-dependent crossover is given by adding the following procedure to the algorithm of the original depth-dependent crossover described in Section 3.2.

STEP6. Each offspring is kept only if its fitness is better than that of its parent.

4.3 Experimental Results

This section has investigated effectiveness of the non-destructive crossover and the original depth-dependent crossover for several problems. This section shows the experimental results in Boolean concept formation problems and the ANT problem.

Table 4.1 shows the experimental set up. In case of a **NORMAL**, crossover points are selected at random. The **DD** means using the "depth-dependent" crossover. The **RDD** represents the "revised depth-dependent" crossover. These three depth-dependent crossovers are original ones. The **ND-NORMAL** denotes the "non-destructive normal" crossover. The **ND-DD** means using the "non-destructive depth-dependent" crossover. The **ND-RDD** represents the non-destructive revised depth-dependent crossover. These three non-destructive crossovers are mentioned in Section 4.2. Mutation is randomly applied for all settings.

Table 4.1: Table 4.1: Experimental Set Up

Setting	Crossover	Mutation
NORMAL	Random	Random
DD	Depth-Dependent	Random
RDD	Revised Depth-Dependent	Random
ND-NORMAL	Random (non-destructive)	Random
ND-DD	Depth-Dependent (non-destructive)	Random
ND-RDD	Revised Depth-Dependent (non-destructive)	Random

For the sake of comparison, all experiments were conducted until a final generation, even if a solution was founded during the evolution. Experimental results are based on the average over twenty runs.

4.3.1 11MX

Figure 4.1, 4.2 and 4.3 plot the tree depth, the number of the function node, and the number of the terminal node with generations for the 11MX problem, respectively. The used parameters are shown in Table 3.2. Details of this task are written in Section 3.5.1.

According to Figure 4.1, the tree depth of the non-destructive crossovers (the **ND-NORMAL**, the **ND-DD** and the **ND-RDD**) is shallower than by the original ones (the **NORMAL**, the **DD** and the **RDD**). The number of the function node (also the number of the terminal node) by the non-destructive crossovers is also smaller than by the original crossovers (see Figure 4.2 and 4.3). Tree growth pattern of the original crossovers (the **NORMAL**, the **DD** and the **RDD**) is rising until the final generations. However, the non-destructive crossovers (the **ND-NORMAL**, the **ND-DD** and the **ND-RDD**) suppress the program growth in the middle of the evolution. These experimental results show that the applicability of the non-destructive crossover for combining other crossover techniques. The non-destructive crossover could generate smaller programs on the four GP problems. This suggests that the non-destructive crossover does not have problem-dependent characteristics.

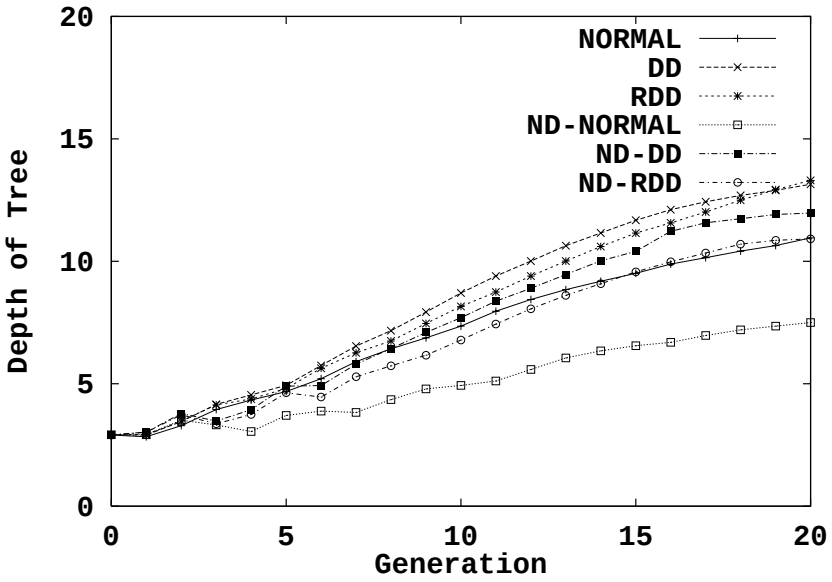

Figure 4.1: Depth of Tree (11MX)

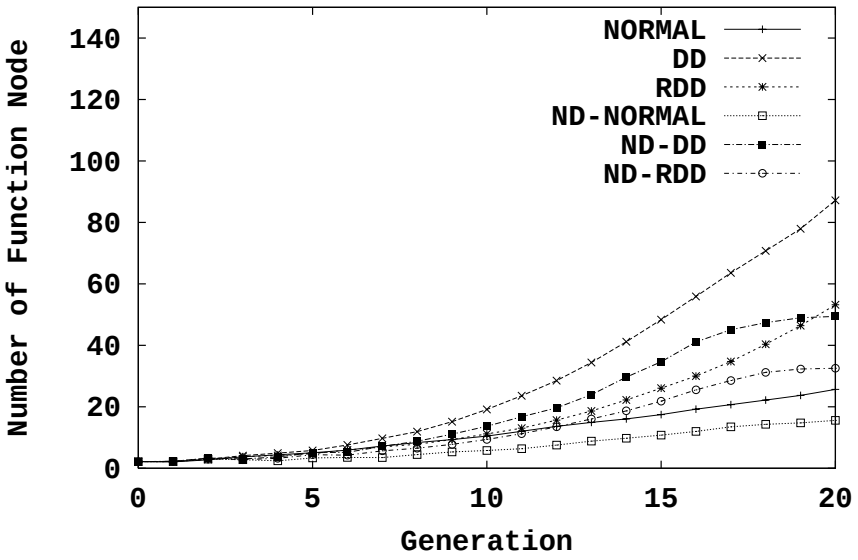

Figure 4.2: Number of Function Node (11MX)

Figure 4.4 and 4.5 give the best and the average fitness values for the 11MX problem, respectively. According to Figure 4.4, the **ND-DD** gave the best performance for the best fitness value. The evolution of the **ND-DD** is the fastest among six crossover settings. On the contrary, the evolution of the **NORMAL** is the slowest.

According to Figure 4.5, the **ND-DD** is superior to the **DD**. The **ND-RDD** is also surpass to the **RDD**. Each non-destructive crossover is superior to each original crossover. As for the hits at the final generations, the **ND-DD** was also the best among the six cases (see Table 4.2).

The experimental results of the non-destructive crossovers are compared with the results of the original crossovers using the the paired t-test [Freund and Wilson, 1992].

Table 4.3 shows the results of t-test. According to this test, it could not conclude that the **ND-DD** is superior to the **DD** in terms of the best fitness value. On the contrary, it has confirmed that the **RDD** is superior to the **ND-RDD** in terms of the best fitness value (5% of level of significance).

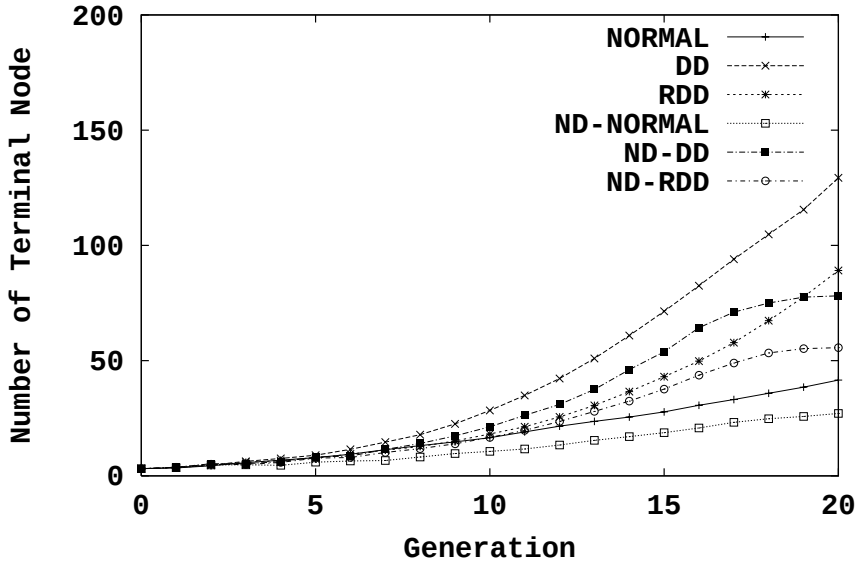

Figure 4.3: Number of Terminal Node (11MX)

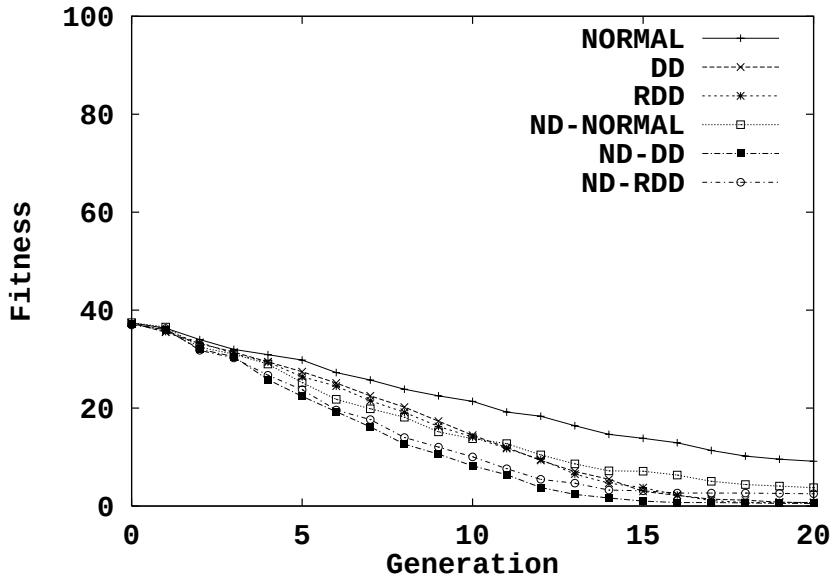

Figure 4.4: Best Fitness Value, means of twenty runs (11MX). The fitness of the 11MX is an error rate for total inputs. The "best" means the fitness of the best individual.

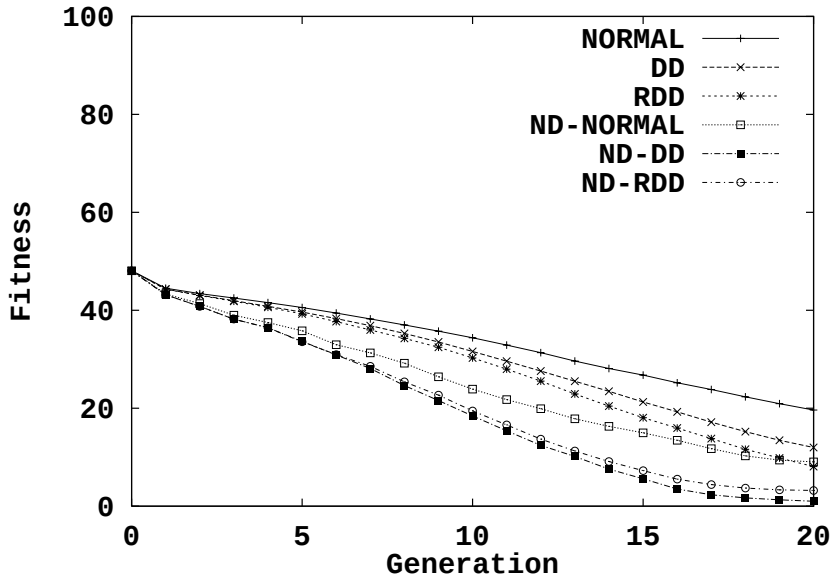

Figure 4.5: Average Fitness Value, means of twenty runs (11MX). The fitness of the 11MX is an error rate for total inputs. The "average" indicates the average fitness of the population.

Table 4.2: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (11MX). If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. Rank indicates a ranking of six crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.00 (0.00)	5
DD	0.70 (0.46)	2
RDD	0.85 (0.36)	1
ND-NORMAL	0.40 (0.49)	4
ND-DD	0.85 (0.36)	1
ND-RDD	0.65 (0.48)	3

Table 4.3: Statistic t for the Best and the Average Fitness Values at the Final Generation (11MX).

Setting	Best	Average
ND-NORMAL (against **NORMAL**)	5.14	12.36
ND-DD (against **DD**)	0.28	5.15
ND-RDD (against **RDD**)	-2.82	2.20

4.3.2 4EVEN

The purpose of the non-destructive crossover is to suppress the program growth. There are some advantages when the size of generated programs is reduced. For instance, a small program does not require huge computer memory. And the computational time of a small program is smaller than that of a huge program. As for the program size, the non-destructive crossover was successful for all four GP problems (i.e., the 11MX, the 4EVEN, the ANT and the robot problem). The following three sections (Section 4.3.2, 4.3.3 and 4.3.4) focuses on the best and the average fitness values, which are index of another program performance.

Figure 4.6 and 4.7 plot the best and the average fitness values for the 4EVEN problem, respectively. The used parameters are same as Table 3.7. Details of this task are written in Section 3.5.2. As for the best fitness value, the **DD**, the **RDD** and the **ND-DD** give good performance. However, the **NORMAL**, the **ND-NORMAL** and the **ND-RDD** fell into local optimum. This is because these three settings (i.e., the **NORMAL**, the **ND-NORMAL** and the **ND-RDD**) generated smaller GP trees than that by the former three settings (i.e., the **DD**, the **RDD** and the **ND-DD**). In terms of the average fitness value, the **ND-DD** is better than the **DD**, and the **ND-NORMAL** is superior to the **NORMAL**. However, the **ND-RDD** and the **RDD** shows same performance. According to the paired t statistical test, it has confirmed that the **DD** is superior to the **ND-DD**, and that the **ND-RDD** is superior to the **RDD** in terms of the best fitness values (Table 4.5). The solution program was acquired for all twenty runs by means of the **DD** (Table 4.4).

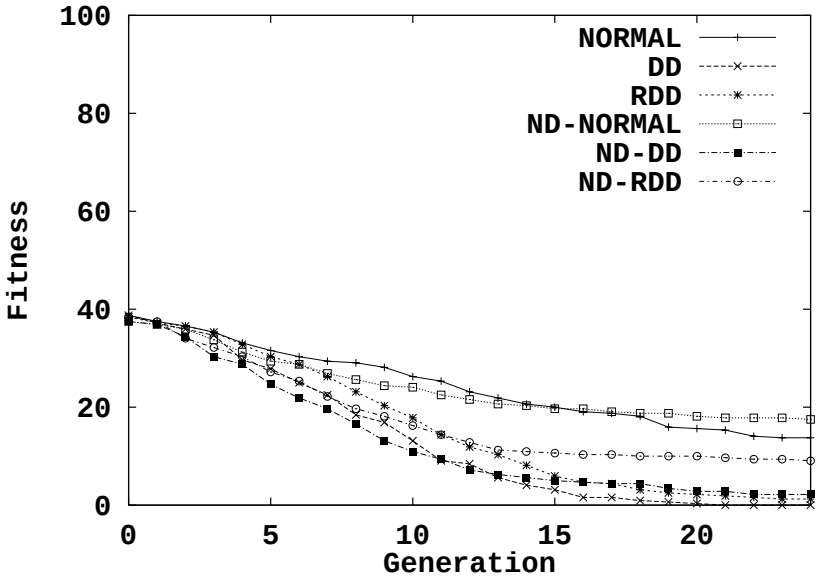

Figure 4.6: Best Fitness Value, means of twenty runs (4EVEN). The fitness of the 4EVEN is an error rate for total inputs. The "best" means the fitness of the best individual.

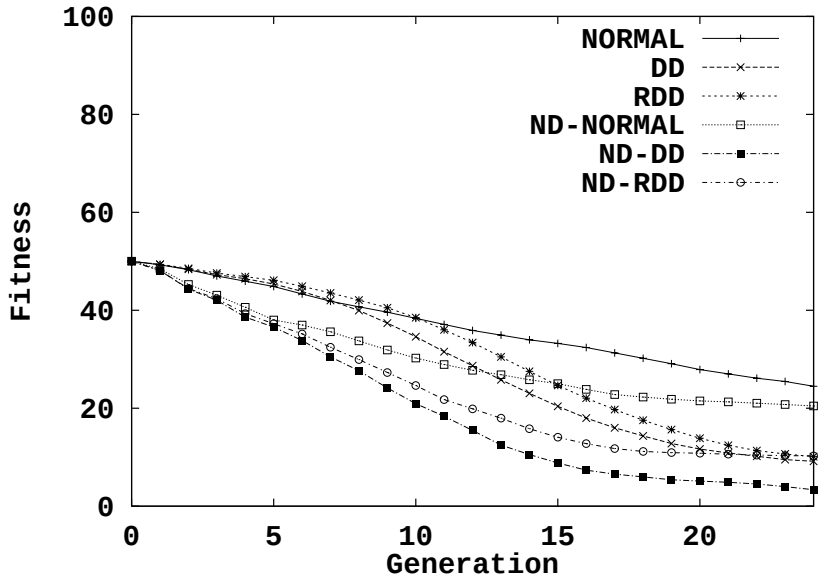

Figure 4.7: Average Fitness Value, means of twenty runs (4EVEN). The fitness of the 4EVEN is an error rate for total inputs. The "average" indicates the average fitness of the population.

Table 4.4: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (4EVEN). If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. Rank indicates a ranking of six crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.00 (0.00)	6
DD	1.00 (0.00)	1
RDD	0.80 (0.40)	2
ND-NORMAL	0.05 (0.22)	5
ND-DD	0.75 (0.43)	3
ND-RDD	0.15 (0.36)	4

Table 4.5: Statistic t for the Best and the Average Fitness Values at the Final Generation (4EVEN).

Setting	Best	Average
ND-NORMAL (against **NORMAL**)	-1.57	2.22
ND-DD (against **DD**)	-2.10	5.06
ND-RDD (against **RDD**)	-5.00	-0.10

4.3.3 ANT

Figure 4.8 plots the best fitness values of the ANT problem. The used parameters are shown in Table 3.12. Details of this task are written in Section 3.5.3. According to Figure 4.8, the performance of the **NORMAL** was the best in terms of the best fitness values. On the contrary, the **ND-RDD** was the worst in terms of the best fitness value. As can be seen in this figure, all six crossover settings suffered from the into local optimum. Figure 4.9 shows the average fitness values of the ANT problem. According to this figure, the **ND-DD** shows the best performance. However, a difference between the **ND-DD** and the other crossover settings are small. On the hits measure, the original crossover and the non-destructive crossover are the same performance on the ANT problem (see Table 4.6). Table 4.7 indicates that results of the paired t statistical test on the ANT problem. According to these results, the **RDD** is superior than the the **ND-RDD** on the best fitness value. However this test could not conclude that the **ND-DD** was superior to the **DD** in terms of the best and the average fitness values. There was no critical difference of the fitness between the original crossover and the non-destructive crossover.

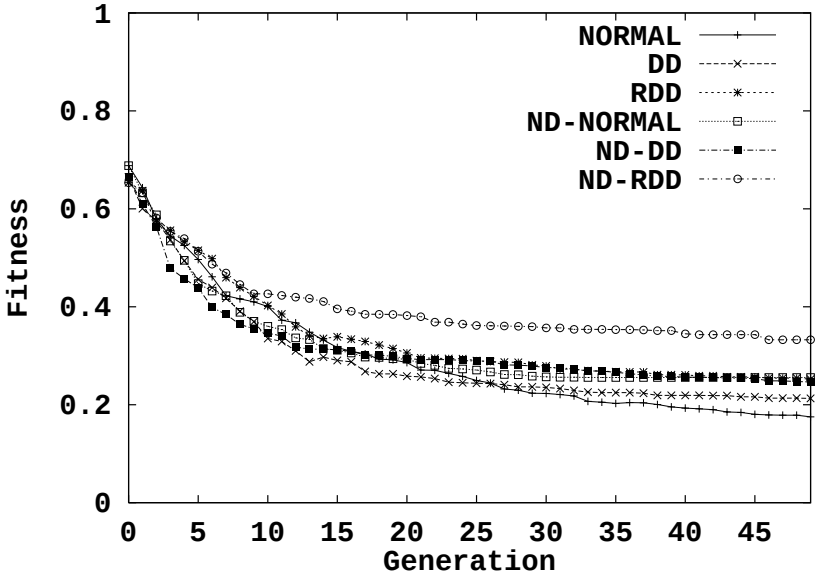

Figure 4.8: Best Fitness Value, means of twenty runs (ANT). The fitness of the ANT is a probability for which the ant could not eat 89 foods. The "best" means the best individual.

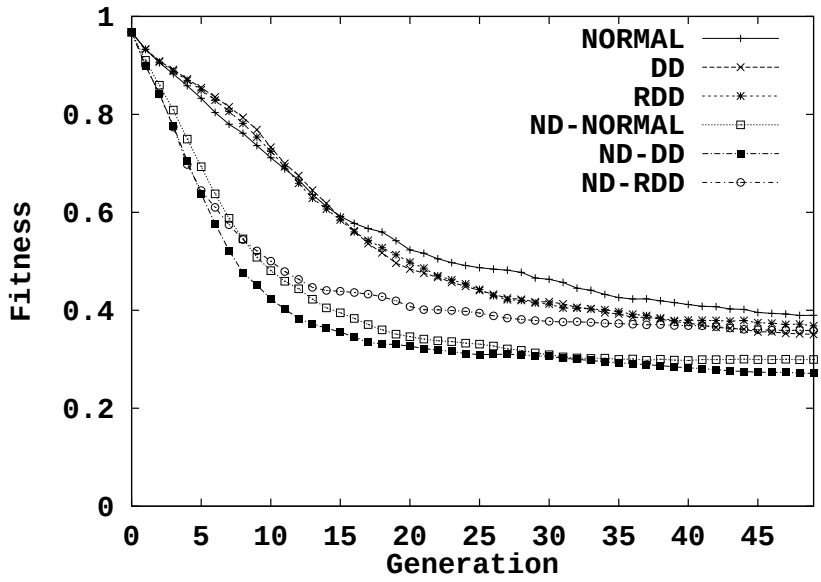

Figure 4.9: Average Fitness Value, means of twenty runs (ANT). The fitness of the ANT is a probability for which the ant could not eat 89 foods. The "average" indicates mean in the population.

Table 4.6: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (ANT). If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. Rank indicates a ranking of six crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.15 (0.36)	1
DD	0.10 (0.30)	2
RDD	0.00 (0.00)	4
ND-NORMAL	0.05 (0.22)	3
ND-DD	0.05 (0.22)	3
ND-RDD	0.05 (0.22)	3

Table 4.7: Statistic t for the Best and the Average Fitness Values at the Final Generation (ANT).

Setting	Best	Average
ND-NORMAL (against **NORMAL**)	-1.61	1.93
ND-DD (against **DD**)	-0.64	1.72
ND-RDD (against **RDD**)	-2.06	0.41

4.3.4 Robot

Figure 4.10 plots the best fitness of the robot problem. The used parameters are shown in Table 3.20. Details of this task are written in Section 3.5.4. As for the best fitness value, the original and the non-destructive depth-dependent crossovers (i.e, the **DD**, the

RDD, the **ND-DD** and the **ND-RDD**) are superior to the **NORMAL** and the **ND-NORMAL**. Figure 4.11 shows the average fitness of the robot problem. According to this figure, the **ND-DD** and the **ND-RDD** give the best performance among the all six crossover settings.

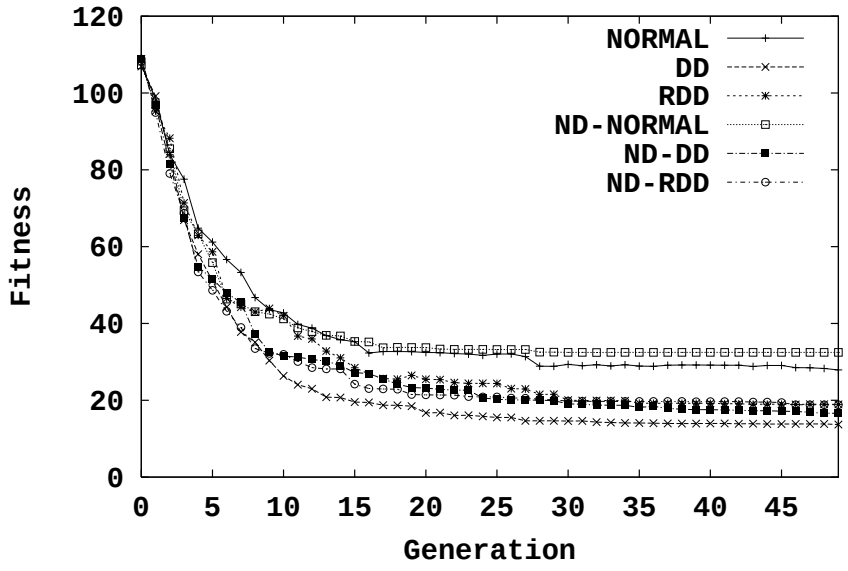

Figure 4.10: Best Fitness Value, means of twenty runs (Robot). The fitness of the Robot is derived from Equation 3.5. The "best" means the fitness of the best individual.

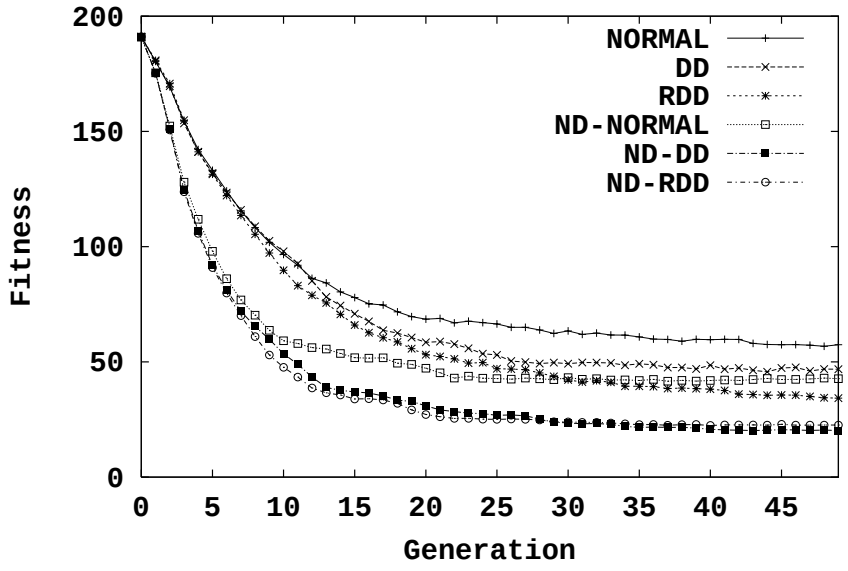

Figure 4.11: Average Fitness Value, means of twenty runs (Robot). The fitness of the Robot is derived from Equation 3.5. The "average" indicates the average fitness of the population.

In terms of the hits measure, the **RDD** and the **ND-DD** generated the solution program over all twenty runs (see Table 4.8). Table 4.9 shows the results of t-test which is comparison of the non-destructive and the original crossovers. In terms of the average

fitness value, the non-destructive crossovers (i.e., the **ND-NORMAL**, the **ND-DD** and the **ND-RDD**) are superior to the original crossovers (i.e., the **NORMAL**, the **DD** and the **RDD**). However, this test did not confirmed that the non-destructive crossovers are superior to the original crossovers.

Table 4.8: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (Robot). If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. Rank indicates a ranking of six crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.80 (0.40)	5
DD	0.95 (0.22)	3
RDD	1.00 (0.00)	1
ND-NORMAL	0.75 (0.43)	6
ND-DD	1.00 (0.00)	1
ND-RDD	0.95 (0.22)	3

Table 4.9: Statistic t for the Best and the Average Fitness Values at the Final Generation (Robot)

Setting	Best	Average
ND-NORMAL (against **NORMAL**)	-0.92	3.45
ND-DD (against **DD**)	-1.21	8.84
ND-RDD (against **RDD**)	0.03	4.03

4.4 Conclusion

This chapter introduced the non-destructive crossover for the depth-dependent crossover and verified the effectiveness in terms of generated program size and fitness values. As a result of experiments, the following points have been made clear:

1. For Boolean problems, the non-destructive depth-dependent crossover created smaller programs than the original depth-dependent crossover.

2. For the ANT problem, the performance for the non-destructive crossover was not always better.

Previous experimental results have shown that the non-destructive depth-dependent crossovers (i.e., **ND-DD** and **ND-RDD**) did not necessarily give better performance than the original depth-dependent crossovers (i.e., **DD** and **RDD**). However, the purpose of this chapter was to reduce the size of generated programs for GP. The non-destructive crossover generated smaller programs with a slight performance degradation.

Chapter 5

A Self-Tuning Mechanism for Depth-Dependent Crossover

The depth-dependent crossover protected building blocks and constructed larger building blocks easily by swapping higher nodes frequently (see Chapter 3). However, there was a problem-dependent characteristics on the depth-dependent crossover, because the depth selection probability was fixed for all nodes in a tree. To solve this difficulty, this chapter proposes a self-tuning mechanism for the depth selection probability. This chapter calls this type of crossover a "self-tuning depth-dependent crossover". This chapter compares GP performances of the self-tuning depth-dependent crossover with performances of the original depth-dependent crossover. Our experimental results clarify the superiority of the self-tuning depth-dependent crossover.

5.1 Introduction

The normal crossover selects randomly a crossover point (node) regardless of its position within the tree structure. Thus, if the normal crossover destroys building blocks, it will result in the degradation of the search for a solution program. Swapping a larger structure is one way to solve the difficulty. Because building blocks are protected by swapping larger structures. Chapter 3 proposed a "depth-dependent crossover", in which a crossover point is determined by a depth selection probability. The depth selection probability is the probability of selecting a depth to which is applied crossover. The depth selection probability is higher for a node closer to a root node. The depth-dependent crossover protected building blocks and constructed larger building blocks easily by swapping higher nodes frequently. Through the use of the depth-dependent crossover, the number of generation was expected to decrease for the boolean concept formation problems. However, there is a problem-dependent characteristic for the depth-dependent crossover, because the depth selection probability was fixed and given as a user-defined parameter. This explains why the depth-dependent crossover could not show any advantage for the fitness performance for the ANT problem (Chapter 3).

This chapter proposes a self-tuning mechanism for the depth selection probability to avoid the difficulty mentioned above. This type of crossover is called a "self-tuning depth-dependent crossover" [Ito *et al.*, 1999]. In case of the self-tuning depth-dependent crossover, each individual has a different depth selection probability. The depth selection probability of a selected individual is copied to the next generation. By using the

self-tuning depth-dependent crossover, it is not required to set up beforehand the depth selection probability for a particular GP task.

The self-tuning is not necessarily the best strategy. If an optimal depth selection is known for one GP problem in advance, setting the optimal depth selection would be better than using the self-tuning mechanism. However, we do not know the optimal depth selection probability for each GP problem in general. Thus, it is required to design the self-tuning mechanism for the depth selection probability.

Angeline argued that adaptive abilities can be separated into three basic classes depending on the association between the adaptive parameters and the evolutionary process [Angeline, 1996].

Population-level adaptations modify population-wide parameters, often include updating the global frequency of operator application and dynamically adjusting the interpretation of the representation.

Individual-level adaptations associate parameters with each individual that determine how the algorithm manipulates the individual.

Component-level adaptations associate adaptive parameters with each component of an evolving individual that determine how each component is modified during reproduction.

The self-tuning depth-dependent crossover belongs to the individual-level adaptations. Because each depth selection probability is changed to suit to each individual during the evolution. There have been other studies of the individual-level adaptations. For instance, Angeline studied an adaptive crossover [Angeline, 1996], in which a self-adaptive individual composed of a program tree and a parameter tree. A parameter tree was identical to its program tree in size and shape. It determined the probability of crossing the tree at the corresponding position. In case of the self-adaptive crossovers, the probability of each node of crossover is determined independently regardless of the depth of the tree. On the contrary, the self-tuning depth-dependent crossover is based on the depth of the tree, and it protects building blocks from a destructive crossover by which a higher node is selected as a crossover point frequently.

This chapter is organized as follows. Section 5.2 explains the self-tuning mechanism for the depth-dependent crossover. Section 5.3 describes several experimental results in several tasks and compares GP performances of the original depth-dependent crossover with performances of the self-tuning depth-dependent crossover. Section 5.4 mentions some conclusions.

5.2 A Self-Tuning Mechanism for Depth-Dependent Crossover

In case of the depth-dependent crossover [1], the depth selection probability is fixed (half of its parent node's probability, see Figure 3.1). It is a user-defined parameter. If the depth selection probability is not set up correctly, crossover may not work well.

[1] Detailed description of the depth-dependent crossover is given in Section 3.2.

To solve this difficulty, this chapter proposes a self-tuning mechanism for the depth selection probability, in which each tree has a different depth selection probability. Then, each crossover point is determined by its depth selection probability. If an individual has a high depth selection probability, it is more highly that crossover will select a shallower node. Therefore, on average, the selected subtrees will be bigger. On the contrary, if an individual has a low depth selection probability, crossover will select a deeper node. Therefore, selected subtrees will be smaller.

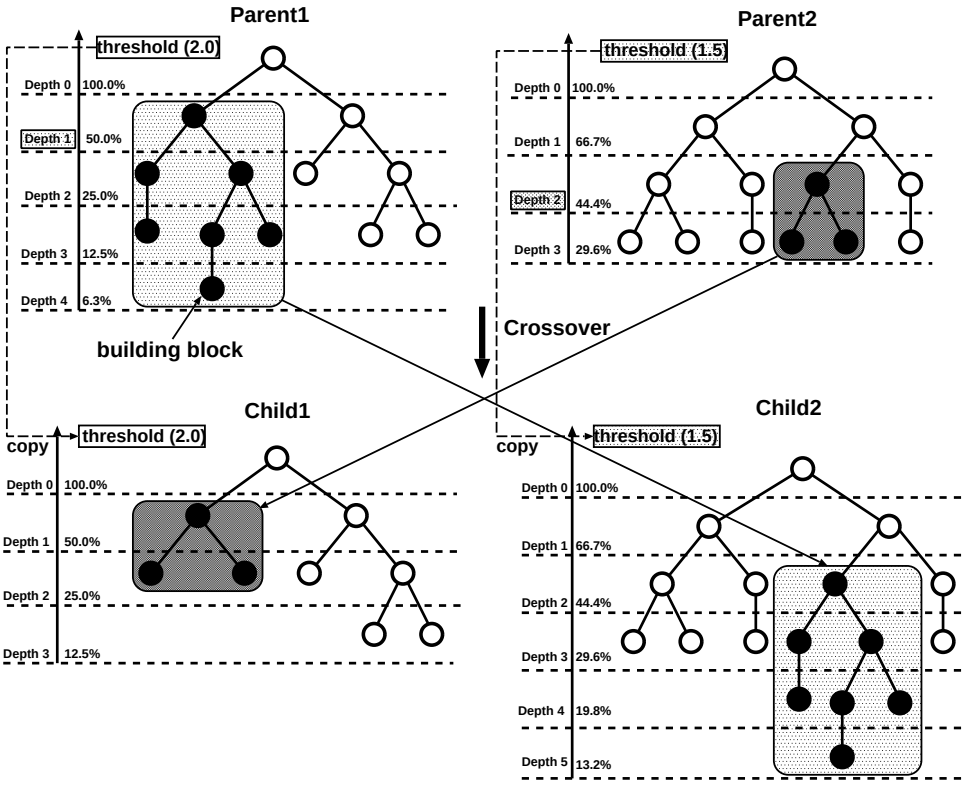

Figure 5.1: Self Tuning without Parameter Crossover

It is difficult to evaluate directly whether each depth selection probability is suitable for each tree structure or not. Because we do not have any evaluation criteria about the depth selection probability. A high depth selection probability may be good setting for one tree structure, whereas, a low depth selection probability may work well for another tree structure. In this self-tuning mechanism, this chapter hypothesized that if the depth selection probability is effectively assigned to the tree structure, the fitness of the tree structure is improved. According to this hypothesis, the depth selection probability of a desirable tree will be copied to the next generation. This means that the depth selection probability of the tree structure, which is selected by the fitness selection, is also selected and inherited to the next generation. This advantage is to evaluate both the depth selection probability and the program fitness (i.e., program performance) by means of only the fitness selection alone.

This chapter introduces two methods for the above purpose. One is to copy the depth selection probability of each parent to each child (Figure 5.1). Another method is to swap the depth selection probability (Figure 5.2). In Figure 5.1, black nodes mean building blocks. The crossover point of each tree is determined by each depth selection probability.

As a result of crossover, two children with the depth selection probability as their parents are generated. In case of the child 2, if the depth selection probability is not swapped, it is easy to select a node of building blocks (i.e., black nodes) and to break building blocks. Because the depth selection probability of nodes within building blocks is higher (i.e., the probabilities of the depth 3, 4 and 5 is 29.6%, 19.8% and 13.2, respectively). The same is true of child 1 (i.e., the probability of the depth 2 is 25.0%).

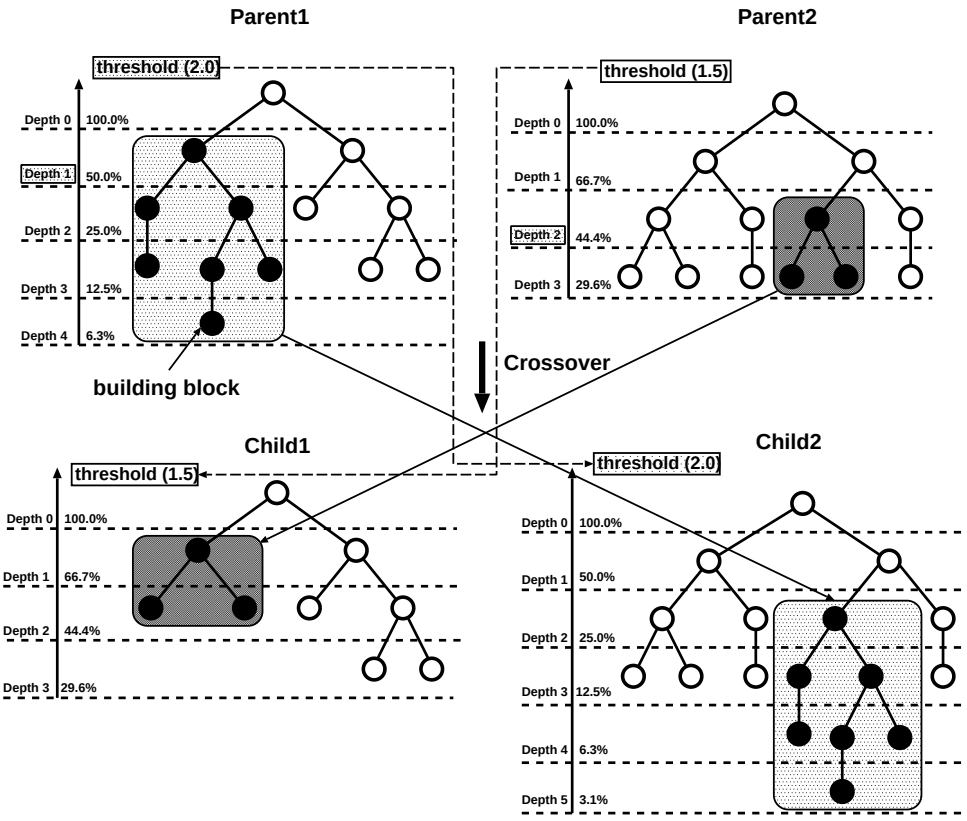

Figure 5.2: Self Tuning with Parameter Crossover

In case of Figure 5.2, the depth selection probability of each parent is also swapped when the selected subtrees are swapped. This method may work effectively for large building blocks. However, it may not be a good strategy for small building blocks. For instance, in case of Figure 5.2, it is easy to protect building blocks of child 2 by swapping the depth selection probability. Because the depth selection probability of nodes within building blocks is low (i.e., the probabilities of the depth 3, 4 and 5 are 12.5%, 6.3% and 3.1%, respectively). However, in case of child 1, the depth selection probability of the node within building blocks is high (i.e., the probability of depth 2 is 44.4%). Large Building blocks of child 2 are protected easily, whereas small building blocks of child 1 are broken. Therefore, the method of Figure 5.2 is a greedy method. Next section clarifies differences of these methods (i.e, Figure 5.1 and 5.2).

The above idea of the self-tuning style depth-dependent crossover is realized as following algorithm.

Pre-processing Assign a random depth selection probability between *min-probability* and *max-probability* for the tree initialization.

Crossover When applying crossover, the following process is executed:

STEP 1. For parent1, determine the depth d for a selected tree using the depth selection probability of parent1.

STEP 2. For parent1, select randomly a node whose depth is equal to d in STEP1.

STEP 3. For parent2, determine the depth d for a selected tree using the depth selection probability of parent2.

STEP 4. For parent2, select randomly a node whose depth is equal to d in STEP3.

STEP 5. Swap the nodes chosen in STEP2 and STEP4.

STEP 6. Swap the depth selection probability (if the self tuning mechanism conducts parameter crossover, i.e., Figure 5.2).

Mutation When applying mutation, the following process is executed:

STEP 1. Mutate a selected tree randomly.

STEP 2. Mutate the depth selection probability of the selected tree between *min-probability* and *max-probability*.

This chapter assigns *min-probability* to 1.5 and *max-probability* to 2.0 (i.e., the depth selection probability of each tree ranges from 1.5 to 2.0). The self-tuning mechanism searches better depth selection probability for each tree between *min-probability* and *max-probability*. Mutation of the depth selection probability is to escape a local optimum for the depth selection probability.

The probability of crossover (and mutation) of the depth selection probability is same as the probability of crossover (and mutation) of the tree structure. This means that the depth selection probability is also swapped when tree structures are swapped with same probability (the depth selection probability is also mutated when the tree structure is mutated).

5.3 Experimental Results

This section has investigated effectiveness of the self-tuning depth-dependent crossover for several problems. These problems are Boolean concept formation problems (the 11MX and the 4EVEN), the ANT problem and the robot problem. Table 5.1 shows the experimental set up. The **NORMAL**, the **DD**, the **SDD** and the **SDD-XO** mean the normal crossover, the depth-depended crossover (Figure 3.1) and the self-tuning depth-dependent crossover in which the value of the depth selection probability are not swapped (Figure 5.1), and the self-tuning depth-dependent crossover in which the value of selection probability are swapped (Figure 5.2), respectively. For the sake of comparison, all experiments were conducted until a final generation, even if a solution was found during the evolution. Experimental results are shown on the average over twenty runs.

Table 5.1: Experimental Set Up

Setting	Crossover	Mutation
NORMAL	Random	Random
DD	Depth-Dependent (Figure 3.1)	Random
SDD	Self-Tuning Depth-Dependent without Parameter Crossover (Figure 5.1)	Random
SDD-XO	Self-Tuning Depth-Dependent with Parameter Crossover (Figure 5.2)	Random

5.3.1 11MX

Figure 5.3 shows the best and the average fitness. Note that the smaller, the better the fitness is. The used parameters are shown in Table 3.2. Details of this task are written in Section 3.5.1.

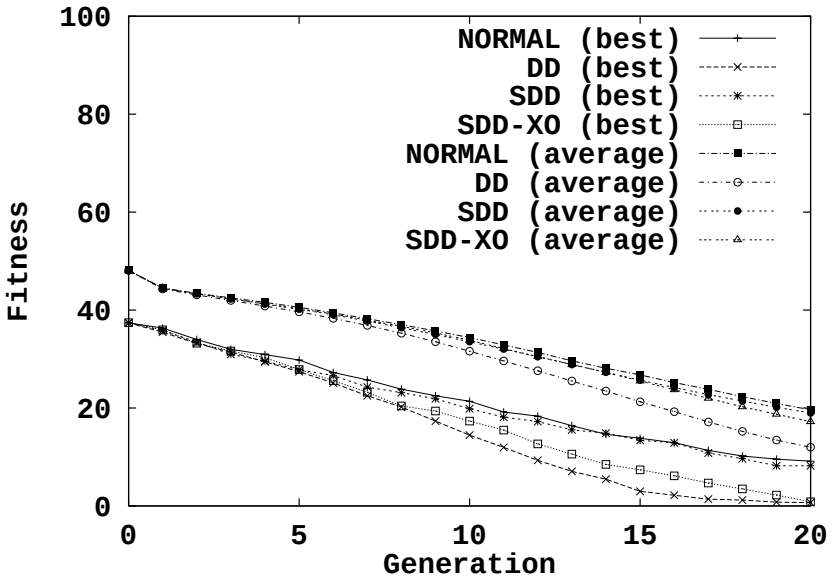

Figure 5.3: Experimental Results, means of twenty runs (11MX). The fitness of the 11MX is an error rate for total inputs. The "best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

According to the best fitness performance curve, the **DD** shows an ability of which it searches the solution program quickly. The **SDD-XO** is slower than the **DD**, however, the **SDD-XO** shows the same fitness performance to the **DD** at the final generation. There was no difference between the **NORMAL** and the **SDD** on the best fitness. As for the average fitness, the **DD** gives the best performance among all crossover settings. Other three crossover settings show almost the same ability.

This section statistically examined the best and the average fitness performances at the final generations with the paired t-test. Table 5.2 shows the result of t-test. According to this test, it has confirmed that the **DD** was superior to the **NORMAL**, and that the **SDD-XO** was superior to the **NORMAL** in terms of the best and the average fitness

values. However, it has not verified that the **SDD** was superior to the **NORMAL** in terms of the best and the average fitness values. Table 5.3 shows average numbers of hits and its standard deviation at the final generations over twenty runs. Note, if the solution program is acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program is not acquired over twenty runs at all, the value is 0.0. According to this table, both of the **DD** and the **SDD-XO** gave best performance among all crossover settings.

Table 5.2: Statistic t for the Best and the Average Fitness Values at the Final Generation (11MX)

Setting	Best	Average
DD (against **NORMAL**)	12.01	10.27
SDD (against **NORMAL**)	0.91	0.77
SDD-XO (against **NORMAL**)	9.91	3.21
SDD (against **DD**)	-11.74	-10.36
SDD-XO (against **DD**)	-0.42	-6.42
SDD-XO (against **SDD**)	12.22	2.62

Table 5.3: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (11MX). Rank indicates a ranking of four crossover settings

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.00 (0.00)	3
DD	0.70 (0.46)	1
SDD	0.00 (0.00)	3
SDD-XO	0.70 (0.46)	1

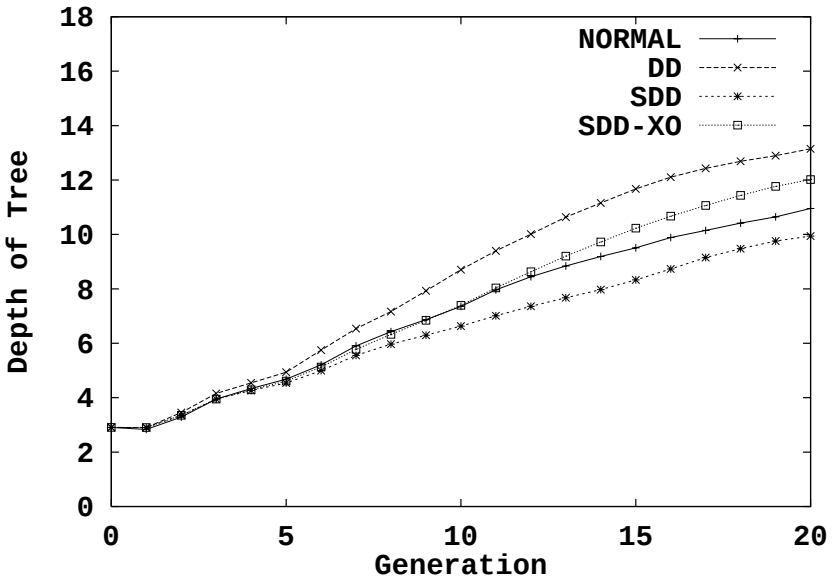

Figure 5.4: Depth of Tree (11MX)

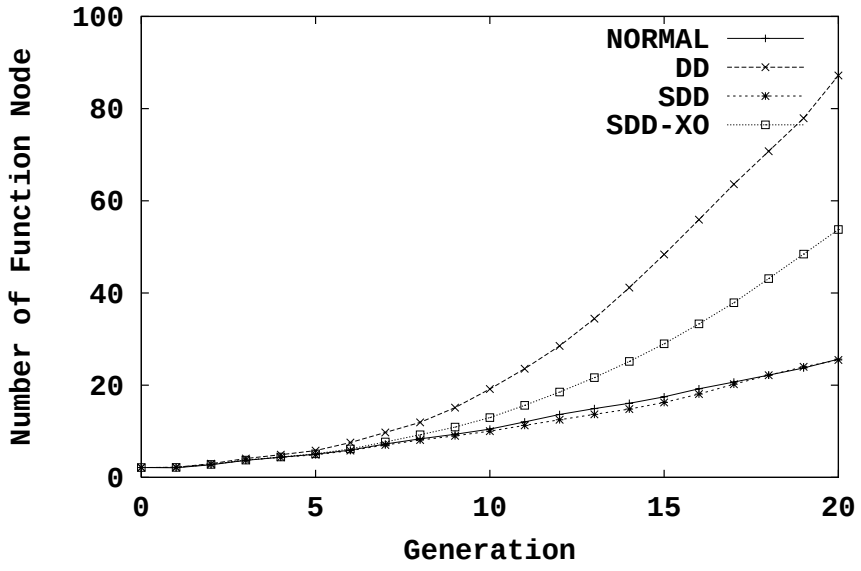

Figure 5.5: Number of Function Nodes (11MX)

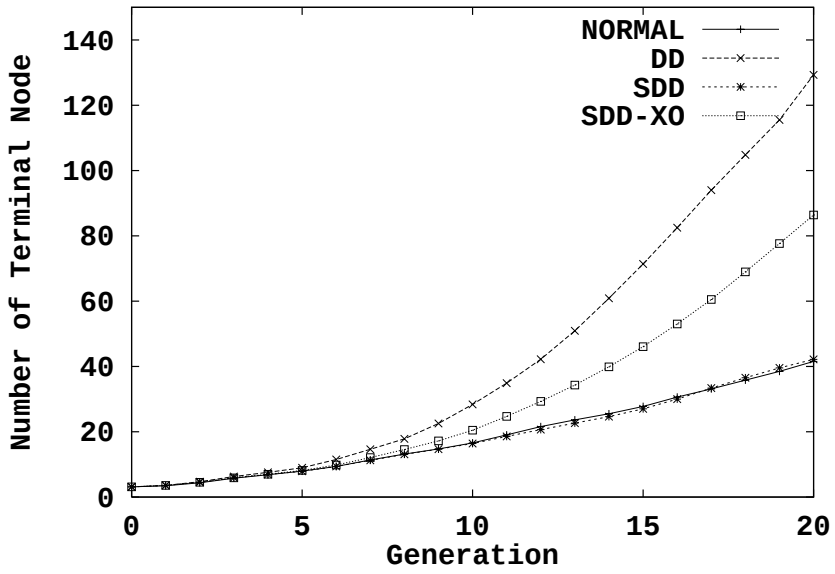

Figure 5.6: Number of Terminal Nodes (11MX)

There was no difference between the **SDD-XO** and the **DD** in terms of the best fitness value as mentioned above. An advantage of the **SDD-XO** against the **DD** is to suppress the size of generated programs (i.e., trees). Figure 5.4, 5.5 and 5.6 plot the depth of the tree, the number of function nodes and the number of terminal nodes with generations for the 11MX problem, respectively. The growth pattern of the **DD** shows the deepest among all crossover settings. On the contrary, the pattern of the **SDD-XO** shows the shallower than that of the **DD** (Figure 5.4). The number of function nodes (and terminal nodes) was much larger by the **DD** than that by the **SDD-XO** (Figure 5.5 and 5.6). By using the **DD**, a shallower node of the tree structure was selected frequently. It frequently occurred that a small subtree in the shallower node of the tree structure was replaced with a large subtree.

The original depth-dependent and the self-tuning depth-dependent crossover have an effect that deeper subtrees are swapped more often. This will result in the protection of building blocks. The fitness performance may be improved because of this effect on the 11MX problem. Figure 5.7 shows average absolute depth of swapped tree structures for the 11MX problem. According to this figure, average absolute depth of swapped subtrees for all crossover settings was shallow at the early generations. However, the **DD** swaps deeper subtrees as the evolution proceeds. The **SDD-XO** swapped shallower subtrees that the **DD**. On the contrary, there was no difference between the **SDD** and the **NORMAL**. These phenomena are also observed in the numbers of nodes of the swapped subtrees (see Figure 5.8).

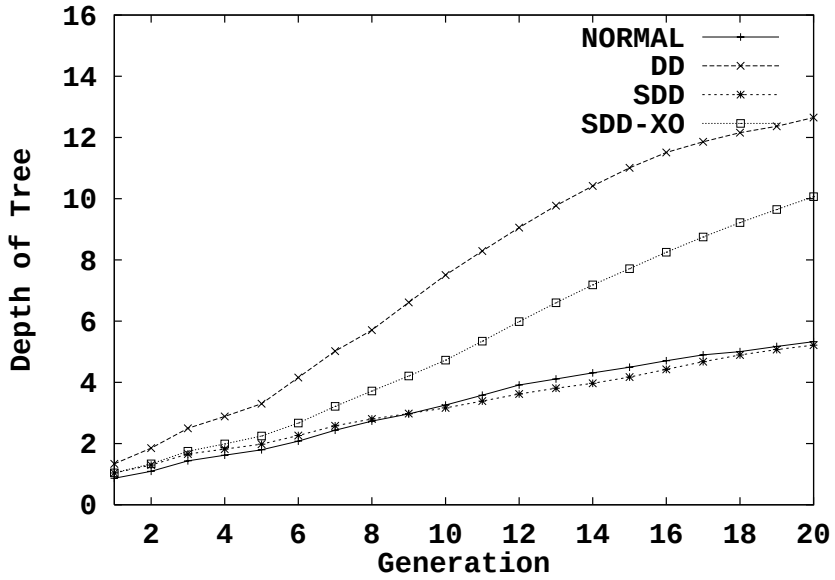

Figure 5.7: Average Absolute Depth of Swapped Tree Structure (11MX)

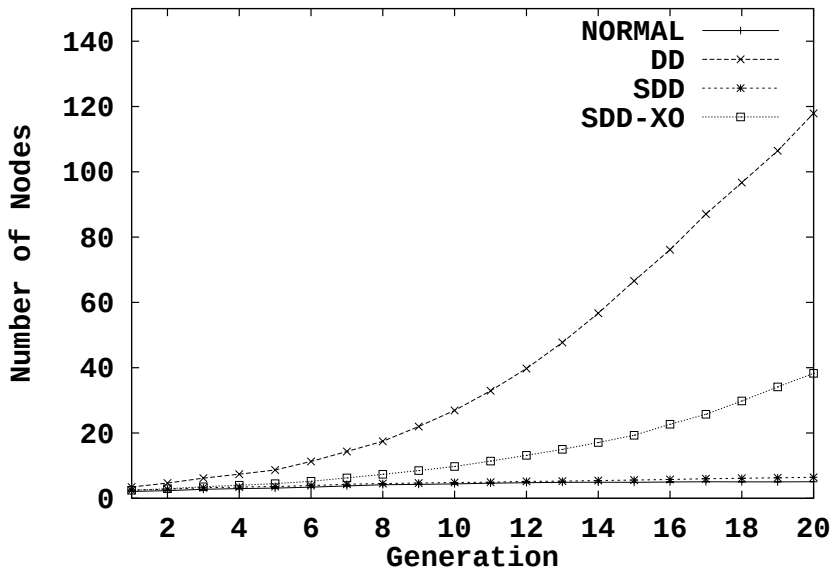

Figure 5.8: Average Number of Nodes of Swapped Tree Structure (11MX)

There was a difference between the **SDD** and the **SDD-XO** in terms of the size of the tree structure. This difference was derived from the depth difference of the selection probability of each individual. Figure 5.9 and 5.10 show transitions of the depth selection probability during the evolution. The numbers of individuals (z axis) are accumulated values over twenty runs. For both cases, i.e., the **SDD** and the **SDD-XO**, the total depth selection probability at the initial generation (i.e., generation 0) was nearly same value (approximately 4000). This was because the depth selection probability of each individual was generated randomly. However, the number of individuals of lower depth selection probability increased during the evolution in case of the **SDD** (Figure 5.9). This means that crossover points were selected at deeper nodes more often. On the contrary, the depth selection probability became higher during the evolution by the **SDD-XO** (Figure 5.10). This means that crossover points are often selected at shallower nodes. These phenomena occurred on the other GP problems, i.e., the 4EVEN, the ANT, and the robot problem.

When we compare two particular crossovers, we have to consider two factors, i.e., the fitness performance and the number of nodes. Even if the fitness is a good performance, we cannot insist that one crossover is better if tree structures become large. A large structure requires huge computer memory and computational time [Ito *et al.*, 1998a]. The **DD** gave good performance in terms of the fitness performance, however, it induced the bloat and enlarged the tree structures. On the other hand, the **SDD-XO** did not only improve the fitness performance, but also suppressed the bloat. Thus, the **SDD-XO** is superior than the **DD** on the 11MX problem.

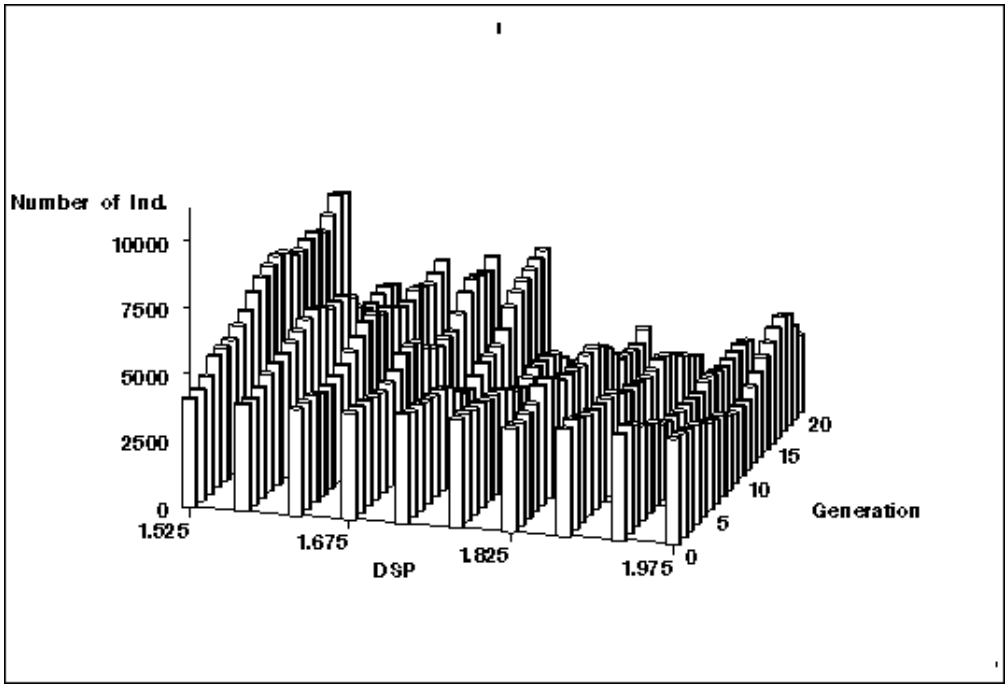

Figure 5.9: Transitions of the Depth Selection Probability during the Evolution (11MX, **SDD**). The numbers of individuals (z axis) are accumulated values over twenty runs.

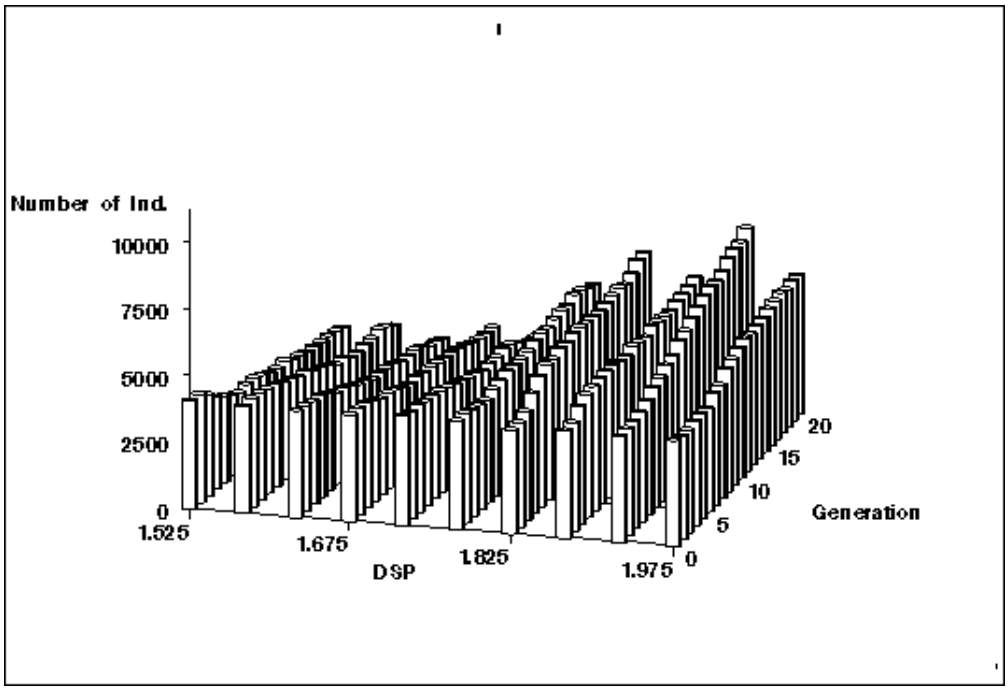

Figure 5.10: Transitions of the Depth Selection Probability during the Evolution (11MX, **SDD-XO**). The numbers of individuals (z axis) are accumulated values over twenty runs.

5.3.2 4EVEN

This section verified effectiveness of the self-tuning depth-dependent crossover for the even-4-parity (4EVEN) problem. The used parameters are shown in Table 3.7. Details of this task are described in Section 3.5.2.

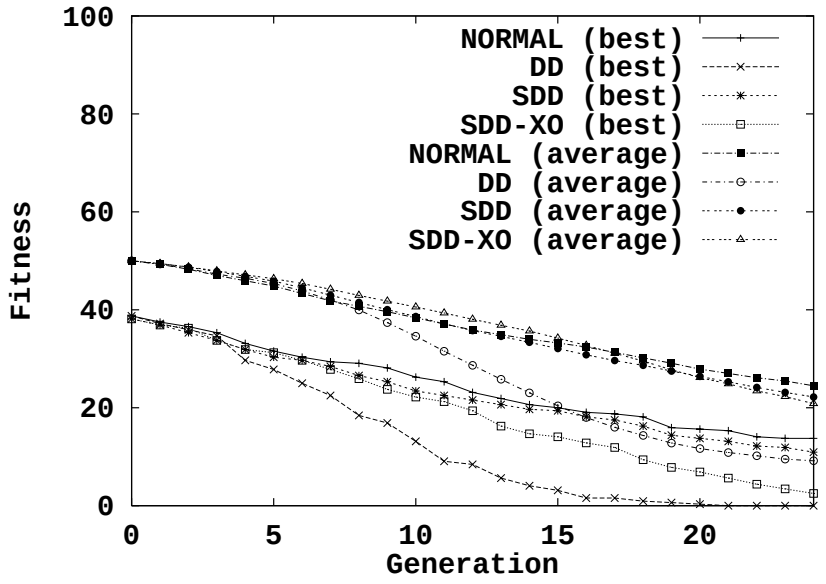

Figure 5.11: Experimental Results, means of twenty runs (4EVEN). The fitness of the 4EVEN is an error rate for total inputs. The "best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

Figure 5.11 plots the best and the average fitness values on the 4EVEN problem. According to this figure, the **DD** gave the best performance on the best and the average fitness values. The **SDD-XO** was better than the **NORMAL**, however, the **SDD-XO** was not better than the **DD** on the best fitness. On the average fitness, there was no difference among the **NORMAL**, the **SDD** and the **SDD-XO**.

Table 5.4 shows results of the paired t-test about the best and the average fitness values. According to this test, it has confirmed that the **DD** is better than the **NORMAL**, and the **SDD-XO** is better than the **NORMAL** on the best and the average fitness values. It has also confirmed that the **DD** is better than the **SDD-XO** on the best and the average fitness values. Table 5.5 shows average numbers of hits and its standard deviation at the final generations over twenty runs. By using the **DD**, a solution program was acquired for all twenty runs.

Table 5.4: Statistic t for the Best and the Average Fitness Values at the Final Generation (4EVEN)

Setting	Best	Average
DD (against **NORMAL**)	11.00	19.29
SDD (against **NORMAL**)	2.13	2.29
SDD-XO (against **NORMAL**)	8.46	3.53
SDD (against **DD**)	-9.19	-13.57
SDD-XO (against **DD**)	-3.55	-11.51
SDD-XO (against **SDD**)	6.47	1.84

Table 5.5: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (4EVEN). Rank indicates a ranking of four crossover settings

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.00 (0.00)	4
DD	1.00 (0.00)	1
SDD	0.10 (0.30)	3
SDD-XO	0.60 (0.49)	2

5.3.3 ANT

Next, this sections show the experimental result on the ANT problem. The used parameters are shown in Table 3.12. Details of this task are written in Section 3.5.3.

It is reported that this problem appears to be difficult because of the large number of sub-optimal peaks in the fitness landscape [Langdon and Poli, 1998a] and the depth-dependent crossover was not effective on this problem (Section 3.5.3).

This section conducted two types of experiments. In the first experiment, the maximum depth for a new tree is 10. In the second experiment, the maximum depth for a new tree is 5 (Table 5.6). This section investigated how these different maximum depth values affect the performance of the **DD**, the **SDD** and the **SDD-XO**.

Table 5.6: Two Types of the Experiment on the ANT Problem

Experiment 1	maximum depth for a new tree is 10
Experiment 2	maximum depth for a new tree is 5

Figure 5.12 plots the best and the average fitness values for the ANT problem of experiment 1 (i.e., the maximum depth for a new tree was 10). According to this figure, the **SDD-XO** gave the best performance on the best and the average fitness values. On the contrary, there was no difference among the **NORMAL**, the **DD** and the **SDD** on the best fitness value. On the average fitness, there was no difference among all crossover settings. These results were examined using paired t-test and it has confirmed that the **SDD-XO** is superior to the **DD** on the best value (Figure 5.7). It has also confirmed that the **SDD-XO** is superior to the **SDD** on the best fitness value. On the hits measure, the **SDD-XO** give the best performance among all crossover settings (Table 5.10).

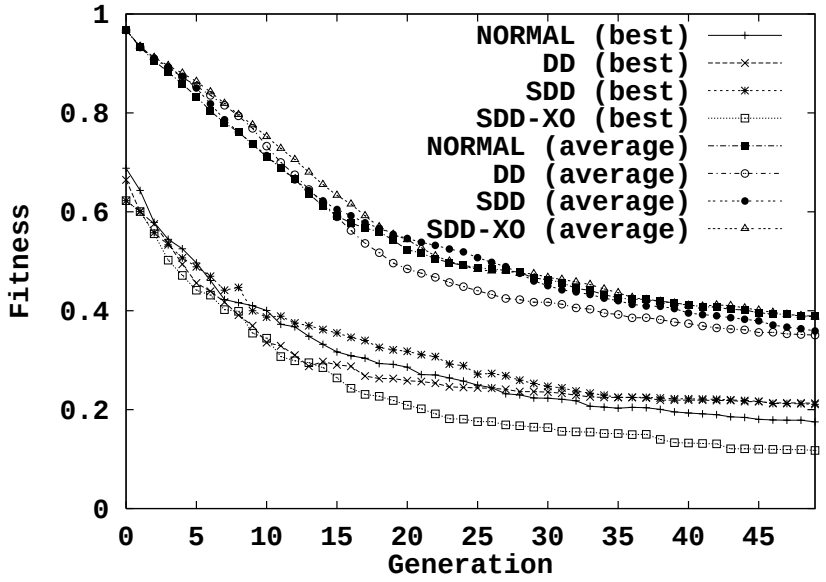

Figure 5.12: Experimental Results, means of twenty runs (ANT, Experiment 1). The fitness of the ANT is a probability for which the ant could not eat 89 foods. The "best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

Table 5.7: Statistic t for the Best and the Average Fitness Values at the Final Generation (ANT, Experiment 1)

Setting	Best	Average
DD (against **NORMAL**)	-0.96	1.14
SDD (against **NORMAL**)	-0.82	-0.63
SDD-XO (against **NORMAL**)	1.34	0.06
SDD (against **DD**)	-0.03	-0.16
SDD-XO (against **DD**)	2.53	-0.89
SDD-XO (against **SDD**)	2.47	-0.90

Table 5.8: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (ANT, Experiment 1). Rank indicates a ranking of four crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.15 (0.36)	2
DD	0.10 (0.30)	3
SDD	0.10 (0.30)	3
SDD-XO	0.35 (0.48)	1

Figure 5.13 plots the best and the average fitness values for the ANT problem of experiment 2 (i.e., the maximum depth for a new tree was 5). According to this figure, the **SDD-XO** also gave the best performance on the best fitness values. There was no difference between the **NORMAL** and the **DD** on the best fitness value. On the average fitness, the **SDD** gave the best performance among all crossover settings. Table 5.10 shows results of paired t-test concerned about the best and the average fitness. It has confirmed that the **SDD-XO** was better than the **DD**. On the hits measure, the **SDD-XO** gave the best performance among all crossover settings.

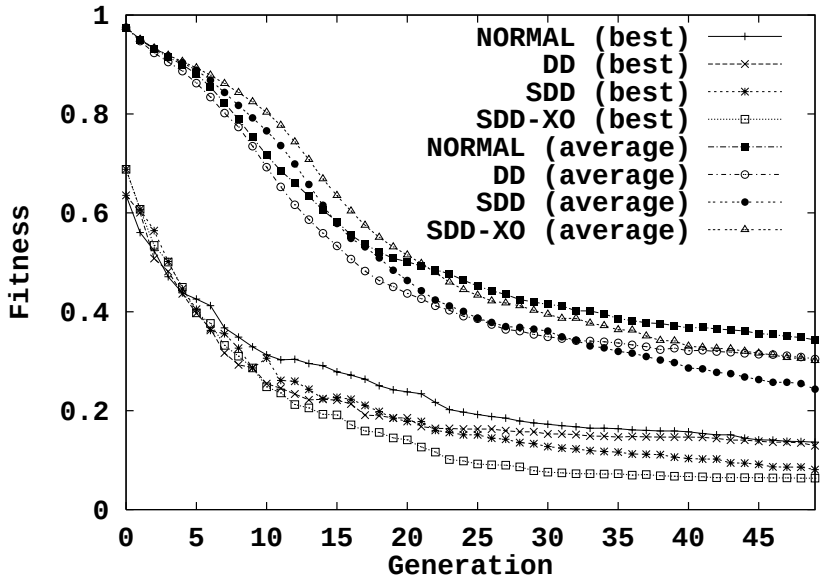

Figure 5.13: Experimental Results, means of twenty runs (ANT, Experiment 2). The fitness of the ANT is a probability for which the ant could not eat 89 foods. The "best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

Table 5.9: Statistic t for the Best and the Average Fitness Values at the Final Generation (ANT, Experiment 2)

Setting	Best	Average
DD (against **NORMAL**)	0.29	1.56
SDD (against **NORMAL**)	1.35	2.61
SDD-XO (against **NORMAL**)	1.69	1.26
SDD (against **DD**)	1.15	1.55
SDD-XO (against **DD**)	1.78	0.09
SDD-XO (against **SDD**)	0.59	-2.46

Table 5.10: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (ANT, Experiment 2). Rank indicates a ranking of four crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.20 (0.40)	4
DD	0.25 (0.43)	3
SDD	0.45 (0.50)	2
SDD-XO	0.60 (0.49)	1

As mentioned above, in case of **NORMAL**, the number of hits in experiment 2 (i.e., the maximum max depth for a new tree is 5) was improved over the number of hits in experiment 1 (i.e., the maximum max depth for a new tree is 10). However, these differences were small. In case of the other crossover settings (i.e., the **DD**, the **SDD** and the **SDD-XO**), the number of the hits in experiment 2 was superior to that in experiment 2 (Table 5.8 and 5.9).

These phenomena are related to the growth of the depth of the tree. Figure 5.14 and 5.15 show the depth of the tree of experiment 1 and that of experiment 2, respectively. According to these figure, the depth of the tree of experiment 1 was shallower than that of experiment 2 in the early generations. In case of experiment 1, the depth of tree grows quickly as the evolution proceeds. On the contrary, in case of experiment 2, the depth of tree became deep slowly during the evolution. At the last generation, the depth of tree was almost same value (about 15) for both experiments. In case of experiments 2, crossover generated building blocks effectively because the depth of the tree was shallow at early generations. On the contrary, crossover was not so effective for experiment 1 because the initial tree depth was deep. This is a reason that the performances of the **DD**, the **SDD** and the **SDD-XO** are better for experiment 2.

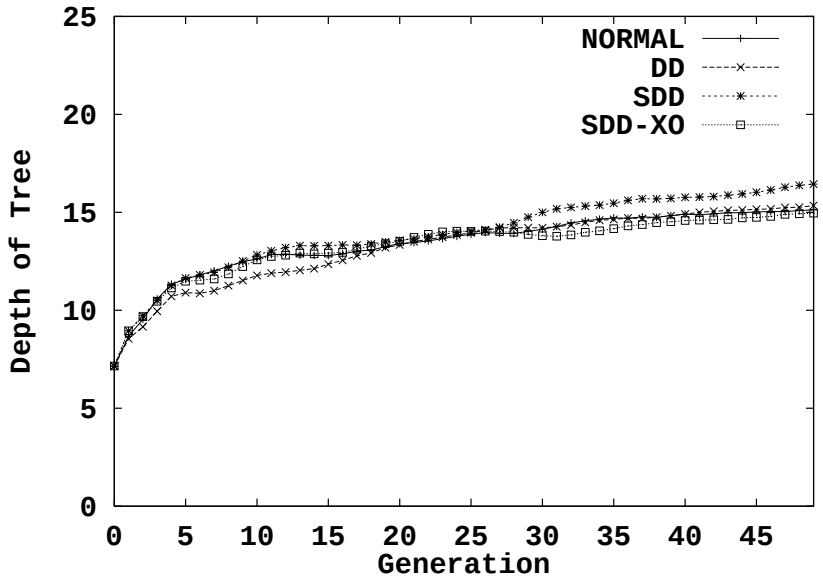

Figure 5.14: Depth of Tree (ANT, Experiment 1)

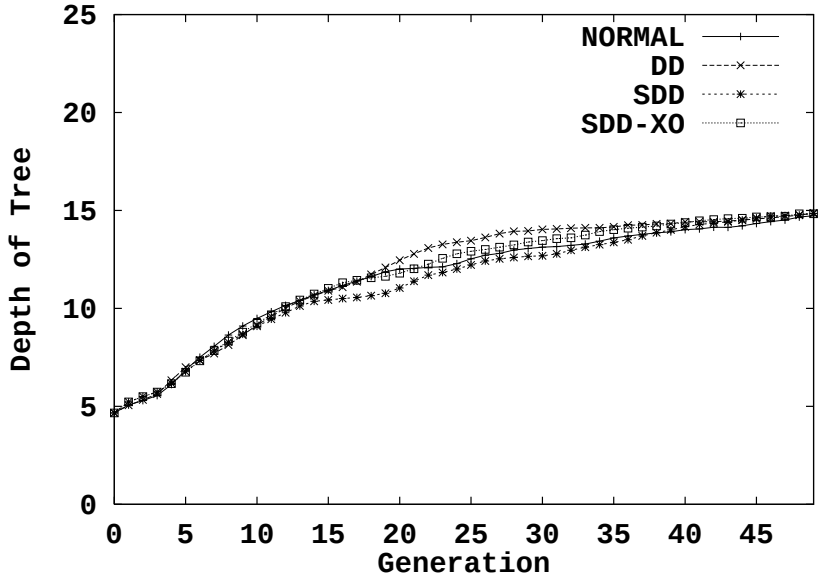

Figure 5.15: Depth of Tree (ANT, Experiment 2)

5.3.4 Robot

Figure 5.16 plots the best and the average fitness values for the robot problem. The used parameters are shown in Table 3.20. Details of this task are written in Section 3.5.4. According to Figure 5.16, the **DD** and the **SDD-XO** gave the best performance on the best fitness value. As for the average fitness value, the **DD** gave the best performance.

Table 5.11 shows the result of t-test. It has confirmed that the **DD** and the **SDD-XO** were superior to the **NORMAL** in terms of the best fitness value. On the average fitness,

the **DD** and the **SDD** were better than the **NORMAL**. However, it has not verified that the **SDD-XO** was superior to the **NORMAL**.

Table 5.12 shows the averaged numbers of his and its standard deviation at the final generations over twenty runs. By using the **SDD-XO**, the solution program was acquired for all twenty runs.

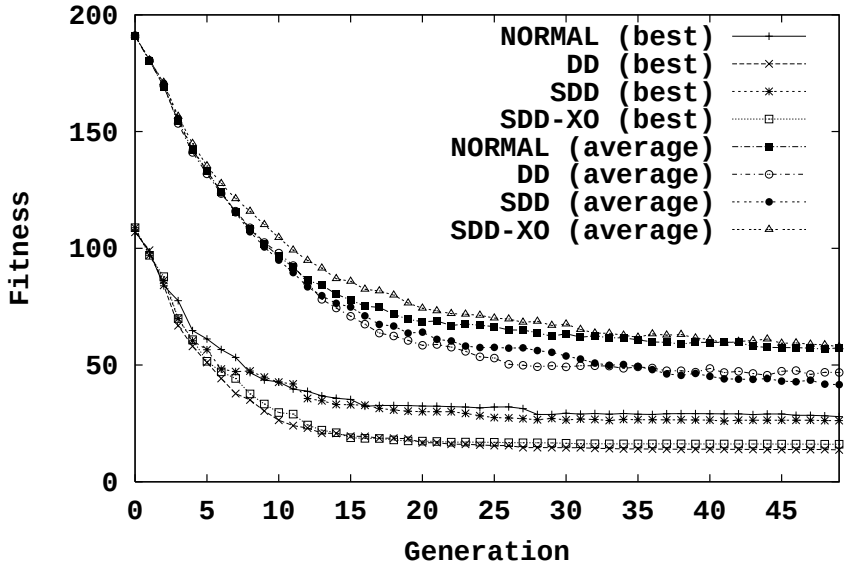

Figure 5.16: Experimental Results, means of twenty runs (Robot). The fitness of the Robot is derived from Equation 3.5. The "best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

Table 5.11: Statistic t for the Best and the Average Fitness Values at the Final Generation (Robot).

Setting	Best	Average
DD (against **NORMAL**)	3.34	2.40
SDD (against **NORMAL**)	0.28	3.15
SDD-XO (against **NORMAL**)	2.50	-0.09
SDD (against **DD**)	-3.26	1.31
SDD-XO (against **DD**)	-0.87	-2.44
SDD-XO (against **SDD**)	2.57	-3.43

Table 5.12: Average Numbers of Hits and its Standard Deviation at the Final Generations over Twenty Runs (Robot). If the solution program was acquired over twenty runs, the hits value is 1.0. On the contrary, if the solution program was not acquired over twenty runs at all, the value is 0.0. Rank indicates a ranking of four crossover settings.

Setting	Hits (Standard Deviation)	Rank
NORMAL	0.80 (0.40)	4
DD	0.95 (0.22)	2
SDD	0.95 (0.22)	2
SDD-XO	1.00 (0.00)	1

5.4 Conclusion

This chapter proposed the self-tuning depth-dependent crossover and examined its performance empirically. As a result of experiments, the following points have been made clear:

1. The self-tuning depth-dependent crossover worked effectively for the four GP benchmark problems.

2. The self-tuning depth-dependent crossover suppressed the growth of the tree structure.

This chapter realized the individual-level adaptation by means of the self-tuning mechanism for the depth selection probability of crossover. Besides the individual adaptations, there are the population-level adaptations and the component-level adaptations as mentioned in the introduction section. Future work will realize these adaptations to scale up GP performance [Bao *et al.*, 1998].

Chapter 6

Discussion

This chapter discusses previous experimental results of the depth-dependent crossover (Chapter 3), the non-destructive depth-dependent crossover (Chapter 4) and the self-tuning depth-dependent crossover (Chapter 5). This chapter also give some discussions as to the building block hypothesis based on survey of former works and the previous experimental results. This hypothesis mentions how crossover searches for the solution program (Section 2.1). This chapter is organized as follows. Section 6.1, 6.2 and 6.3 discusses the depth-dependent crossover, the non-destructive depth-dependent crossover and the self-tuning depth-dependent crossover, respectively. Section 6.4 discusses the building block hypothesis.

6.1 Depth-Dependent Crossover

The tree structure for the ANT problem is evaluated from left side to right, and a program including prog2 or prog3 functions conducts a sequential planning. The second argument of the prog2 statement has to be evaluated effectively only after the first argument is executed. In other words, the evaluation order is essential for this task. As we have seen in the previous experiments (Section 3.5.3), the depth-dependent crossover generally promotes the growth of GP trees during the evolution. A large tree includes many redundant structures, which all contribute to the result of the evaluation. This explains the failure of the depth-dependent crossover for the ANT problem. On the other hand, in case of Boolean functions, the evaluation order is not significant. Moreover, even if a tree structure becomes large and redundant, such a redundancy will work well for solving Boolean problems. This is because a large solution program include several small substructures for the Boolean problems.

6.2 Non-destructive Depth-Dependent Crossover

The previous experimental results (Section 4.3) showed that the non-destructive depth-dependent crossover was not always superior to the original depth-dependent crossover in terms of the best fitness value. However, the non-destructive depth-dependent crossover gave better performance than the original depth-dependent crossover in terms of the average fitness value. This is because most of the individuals "encapsulate junk subprograms" as a result of the original depth-dependent crossover. "Encapsulation" means

the protection of programs from the destruction by the crossover. The term "junk" means a sub-program which does not contribute to generating a solution. If the encapsulation works well, the fitness value of a program is improved. A small number of individuals get some benefits of improving its fitness from the original depth-dependent crossover. Thus, the original depth-dependent crossover is a greedy method. A greedy method is a search method in which most of the individuals are sacrificed to generate a few elites, as seen by the original depth-dependent crossover. On the contrary, the non-destructive depth-dependent crossover is not a greedy method.

This claim is clarified by the autocorrelation analysis described below. The autocorrelation evaluates the mutual role of the fitness function and the genetic operator in the search process on the landscape [Slavov and Nikolaev, 1997]. The experiment of the autocorrelation is to generate random programs and then to apply crossover to these programs. This experiment included a repetition of twenty runs of generating random 10,000 individuals to measure the autocorrelation. Used parameters are the same as the former experiments. The autocorrelation ranges from -1 to 1. The closer to 1, the more highly correlated the fitness and the operator are, which means a desirable situation for the adaptive search. According to this analysis, the non-destructive depth-dependent crossover shows a higher correlation than the original depth-dependent crossover for the three problems (see Table 6.1). This result supports the hypothesis, i.e., most of the individuals "encapsulate junk sub-programs" by means of the **DD**.

Table 6.1: Autocorrelation

Problem	The original crossover			The non-destructive crossover		
	NORMAL	**DD**	**RDD**	**ND-NORMAL**	**ND-DD**	**ND-RDD**
11MX	0.38	0.40	0.40	0.83	0.75	0.80
4EVEN	0.47	0.38	0.34	0.93	0.89	0.89
ANT	0.48	0.41	0.41	0.91	0.84	0.83

Crossover is liable to promote the program growth in size [Soule and Foster, 1997, Angeline, 1998]. The number by the non-destructive depth-dependent crossover is smaller than that by the original depth-dependent crossover. This is because the parent is copied to the next generation if its fitness is not worse than that of the offspring in case of the non-destructive depth-dependent crossover. Thus, the non-destructive depth-dependent crossover is expected to generate smaller programs than the original depth-dependent crossover due to the reduction of the number of crossover.

In case of the Boolean concept formation, the relationship between the fitness and the program size is strong, so that the large the size, the better the fitness is. One reason why the original depth-dependent crossover was successful in the Boolean concept formation seems to be the generation of large programs. In case of the non-destructive depth-dependent crossover, in which only a better offspring is kept for the next generation, the program growth in size is prevented so that GP may be easily trapped in a local optimum.

As for the ANT problem, the performance difference could be explained not by the program size problem, but by the fixed depth selection probability (Equation 3.1). The fixed depth selection probability is suitable for one GP task. However, it may not be suitable for another GP task.

On the robot problem, the original and the non-destructive depth-dependent crossover were superior to the normal crossover. This indicates that the depth-dependent crossover can be applicable to various GP problems except the Boolean concept formation problems.

6.3 Self-Tuning Depth-Dependent Crossover

Section 5.2 has hypothesized that, via a self tuning mechanism, if the depth selection probability is assigned to the tree structure, the fitness value of the tree structure is improved. The self-tuning mechanism has been designed to achieve this hypothesis. Previous experimental results (Section 5.3) have shown that the self-tuning depth-dependent crossover with parameter crossover (**SDD-XO**) gave good performance for all the four GP problems. For these problems, each individual has various kinds of depth selection probability in early generations because the probability was assigned randomly. However, the number of individuals of high depth selection probability increased gradually as the evolution proceeded (Figure 5.10). As a result, the fitness performance was improved. According to Figure 5.10, the self-tuning mechanism works so as to search for building blocks globally at early generations and to protect completed building blocks at later generations. The self-tuning depth-dependent crossover will work for a variety of GP problems due to this effect (adaptability).

6.4 Building Block Hypothesis

Some researchers argued against the building block hypothesis. Angeline discussed that the building block hypothesis was not descriptive of crossover [Angeline, 1997]. He compared SHCC (Strong Headless Chicken Crossover) with WHCC (Weak Headless Chicken Crossover). SHCC works as follows. A random tree was generated for the mate which was chosen by selection. After crossover generated new trees, the modified mate was selected to be the offspring of the mate. WHCC works follows. A random tree was generated for the mate which was chosen by selection. After crossover, the modified mate and the modified random generated tree were selected with equal probability to be the offspring. His experimental results showed that the normal crossover gave better results than two types of HCC for the 6-bit even parity problem. However, there was no significant difference between the three types for the spiral problem. On the contrary, two types of HCC were superior to the normal crossover for the sunspot prediction problem. Angeline concluded that:

> The best class of problems to look for potential building blocks would be the boolean functions. Complex boolean functions often contain sub-functions that can be composed into the larger solutions[Angeline, 1997, p. 16].

Luke and Spector investigated experimentally the existence of building blocks. They compared the GP performance of 90% crossover and 10% reproduction with performances of 90% mutation and 10% reproduction using GP standard problems (i.e., the 6-multiplexor, the lawnmower, the symbolic regression and the ANT problem)
[Luke and Spector, 1997]. Based on their experimental results, they concluded that:

> Crossover does often yield better results than subtree mutation, however, the difference between the two is usually not very significant [Luke and Spector, 1997, p. 243].

Their conclusion is consistent with Koza's claim. Koza usually uses only crossover with large population which consists of variety individuals, in spite of mutation [Koza, 1992a, p. 106]. They also conducted a revised version of statistical comparison with these two genetic operators and confirmed that:

> Crossover does have some advantage over mutation given the right parameter settings (primarily larger population sizes), though the difference between the two surprisingly small [Luke and Spector, 1998, p. 208].

Experimental results of the depth-dependent crossover (Section 3.5) was successful for the boolean problems (i.e., the 11MX and the 4EVEN). These experimental results confirmed Angeline's claim, i.e., the depth-dependent crossover was suitable especially for the problems that can be decomposed into sub-problems such as boolean problems. The depth-dependent crossover was also tested for the ANT problem and was not successful. As shown in Luke's experimental results, the difference between crossover and mutation is insignificant [Luke and Spector, 1998]. It is difficult that crossover to pile up building blocks for the ANT problem, because there are many local optimum [Langdon and Poli, 1998a]. This is a reason which the depth-dependent crossover was not successful for the ANT problem.

One drawback of the depth-dependent crossover is derived from the fixed depth selection probability. Although the size of building blocks of each individual is not always the same. The same depth selection probability is applied to every selected individual.

To solve this difficulty, the self-tuning mechanism was proposed for the depth-dependent crossover (Chapter 5). In case of the self-tuning mechanism, its individual is assigned a different depth selection probability. Then, each crossover point is determined by each depth selection probability. If an individual has a high depth selection probability, it is more highly that crossover will select a deeper node. On the contrary, if an individual has a low probability, crossover will select a shallower node more node. The self-tuning depth-dependent crossover was not successful for the boolean problems, but was effective for the ANT problem. According to the transitions of the depth selection probability, its probability increased during the evolution and thus GP performances were improved on the **SDD-XO** (Figure 5.10). This phenomenon indicates that building blocks became large through the successful evolution.

The depth-dependent crossover (including its various revised versions) was proposed in order to combine building blocks effectively. In case of the normal crossover, it selects a node randomly for each individual. Therefore, it is possible that the normal crossover is liable to break building blocks. To protect building blocks, the depth-dependent crossover was proposed. It selected a larger structure with a high probability by setting the node selection probability of a shallower node (a closer node to a root node) to be higher, and the probability of a deeper node (a distant node from a root node) lower. The depth-dependent crossover was proposed by the following consideration. At the initial generation (generation 0), small building blocks are generated by the tree initialization method. The fitness of individual with such building blocks is better than that of the individual without building blocks. Therefore, the individuals that have building blocks

prosper in a population. As this section discussed above, the applicability of all four types of the depth-dependent crossovers is based on the building block hypothesis. In other words, if the building block hypothesis does not hold good, the depth-dependent crossovers will not be successful. Previous experimental results of four types of the depth-dependent crossovers clearly show that the building block hypothesis is true of GP as well.

Chapter 7

Conclusion

7.1 Conclusions

The goal of this thesis is to improve the efficiency in the program generation. "Efficiency" means to reduce the number of generations required to generate the solution program. For this goal, this thesis proposes four new crossovers for GP.

The first and the second crossovers, i.e., the depth-dependent crossover and the revised one, have contributed to reducing the number of generations for Boolean problems (Chapter3). This indicated that building blocks were able to be protected so that larger building blocks were constructed by means of crossover, which was applied to a shallower node of the tree structure. Compared with the depth-dependent crossover, the revised depth-dependent crossover have suppressed the size of the tree structure effectively.

The third crossover, i.e., the non-destructive depth-dependent crossover was proposed to solve the program size problem of the depth-dependent crossover (Chapter 4). By using the non-destructive depth-dependent crossover, small programs have been generated for all four GP problems (i.e., the 11MX, the 4EVEN, the ANT and the robot problem). These experimental results indicated that non-destructive crossover was independent from the problem domain.

The depth-dependent crossover had the problem dependent characteristic, i.e., it was successful for Boolean problems but not successful for the ANT problem. To apply the depth-dependent crossover to various GP problems, the fourth crossover, i.e., the self-tuning depth-dependent crossover was proposed (Chapter 5). By using the self-tuning depth-dependent crossover, all four GP have been solved effectively. This self-tuning depth-dependent crossover was also successful in reducing the program size because of using the different depth selection probability.

Furthermore, this thesis discussed the building block hypothesis which explains how crossover searches for the solution program with the survey of previous works and these experimental results (Section 6.4).

7.2 Directions for the Future Research

This work adopted simple GP which had a single gene. The Future research will include investigation of effectiveness of the depth-dependent crossovers for other types of GP, e.g., GP with ADFs (Automatically Defined Functions) [Koza, 1994a]. GP with ADFs consists of some function-defining branches which contain the definition of functions and

the result-producing branch which returns the fitness value by using the function-defining branches. GP with ADFs generates a program similar to the one coded by human beings. Usually, we write programs which contain a main function and several subroutines for a large scale program. Therefore, GP with ADFs is suitable for the large scale problem such as a read-world problem. It is uncertain that how much the depth-dependent crossovers generate computer programs efficiently for GP with ADFs. This investigation is essential for the future research.

Program structures of GP problems used for this work's experiments (i.e., the 11MX, the 4EVEN, the ANT and the robot problem) do not include recursive and iterative functions (e.g., `do until`). If there are such functions in the function set, GP will generate more complicated programs. Especially, if recursive functions are included, the generated programs will be more compact. However, it is difficult for GP to generate programs which contain such functions. Because if a program structure has recursive and iterative functions, the program falls into infinite or near-infinite loops easily. The depth-dependent crossover may not work well for programs which have recursive and iterative functions. However, this claim is not necessarily clear.

Some researchers tackled to treat recursive functions on GP. For instance, Brave experimented in GP with recursive ADFs [Brave, 1996]. He extended ADFs which were allowed to call themselves (i.e., ADF1 included ADF1 in its function set and ADF2 included ADF2). According to his experimental results, GP with recursive ADFs showed the best performance among simple GP, GP with ADFs and GP with recursive ADFs. The future research also investigates GP performances of the depth-dependent crossover for using recursive functions.

In this work, a gene structure of GP was a tree. Besides the tree structure, there are a linear and a graph structure [Banzhaf *et al.*, 1998, pp.239–276]. An example of the linear structure is machine language (assembly language program) [Crepeau, 1995, Nordin and Banzhaf, 1995a]. Crepeau proposed the GEMS (Genetic Evolution Machine-language Software) system to evolve the machine language using the framework of GP. The GEMS system could interpreted the Z80 microprocessor using only 660 instructions (the Z80 microprocessor has 691 unique instructions) [Crepeau, 1995].

An example of the graph structure is PADO (Parallel Algorithm Discovery and Orchestration) system [Teller, 1996]. In case of PADO, each program consisted of a main program (MAIN) and private ADFs. Each PADO program was regarded as a directed graph of N nodes. Each node could have as many as N outgoing arcs. Teller claimed that the PADO system could classify signals better than simple GP with the tree structure.

These structures are quite different from the tree structure. Therefore, effectiveness of the depth-dependent crossovers is uncertain. The future research will also include verification of the depth-dependent crossovers for these structures.

Bibliography

[Aho *et al.*, 1974] Aho, A., Hopcroft, J. and Ullman, J. The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974

[Angeline, 1996] Angeline, J. P. Two Self-Adaptive Crossover Operators for Genetic Programming, In Angeline, P. and Kinnear, Jr. K. editors, *Advances in Genetic Programming 2*, pages 89–110, MIT Press, 1996

[Angeline, 1997] Angeline, J. P. Subtree Crossover: Building Block Engine or Macromutation?, In Koza, J., Deb, K., Dorigo, M., Fogel, D. Garzon, M., Iba, H. and Riolo, R. editors, *Proceedings of the Second Annual Conference Genetic Programming 1997 (GP97)*, pages 9–17, MIT Press, 1997

[Angeline, 1998] Angeline, J. Subtree Crossover Causes Bloat In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K. Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R. L. editors, *Proceedings of the Second Annual Conference Genetic Programming 1997 (GP98)*, pages 745–752, MIT Press, 1998

[Banzhaf *et al.*, 1998] Banzhaf, W., Nordin, P., Keller, R. and Francone, F. Genetic Programming An Introduction, Morgan Kaufmann Publishers, Inc., 1998

[Bao *et al.*, 1998] Bao, H. T., Nguyen, T. D., Nguyen, N. B., and Ito, T. Development of Some Methods and Tools for Discovering Conceptual Knowledge In *Proceedings of the Discovery Science 1998 The First International Conference on Discovery Science (DS'98)*, Springer-Verlag, 1998

[Brave, 1996] Brave, S. Evolving Recursive Programs for Tree Search, In Angeline, P. and Kinnear, Jr. K. editors, *Advances in Genetic Programming 2*, pages 203–219, MIT Press, 1996

[Crepeau, 1995] Crepeau, R. Genetic Evolution of Machine Language Software In Rosca, J. editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 121–134, University of Rochester, Technical Report 95.2, 1995

[Freund and Wilson, 1992] Freund, R. and Wilson, W. Statistical Methods, Academic Press, Inc. 1992

[Goldberg, 1989] Goldberg, D. Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989

[Holland, 1995] Holland, J. H. Hidden Order, Addison-Wesley, 1995

[Handley, 1994] Handley, S. G., The Automatic Generation of Plans for a Mobile Robot via Genetic Programming with Automatically Defined Functions, In *Advances in Genetic Programming*, MIT Press, 1994

[Harvey, 1995] Harvey, I. *The Artificial Evolution of Adaptive Behavior*, PhD Thesis, The University of Sussex, 1993

[Haynes, 1998] Haynes, T. D. *Collective Adaptation: The Sharing of Building Blocks*, PhD thesis, The University of Tulsa, 1998

[Haynes and Wainwright, 1995] Haynes, T. D. and Wainwright R., A Simulation of Adaptive Agents in a Hostile Environment, In *Proceedings of the 1995 ACM Symposium on Applied Computing*, ACM Press, 1995

[Harries and Smith, 1997] Harries, K. and Smith, P. Exploring Alternative Operators and Search Strategies in Genetic Programming, In Koza, J., Deb, K., Dorigo, M., Fogel, D. Garzon, M., Iba, H. and Riolo, R. editors, *Proceedings of the Second Annual Conference Genetic Programming 1997 (GP97)*, pages 147–155, MIT Press, 1997

[Iba, 1997] Iba, H. Multiple-Agent Learning for a Robot Navigation Task by Genetic Programming, In Koza, J., Deb, K., Dorigo, M., Fogel, D. Garzon, M., Iba, H. and Riolo, R. editors, *Proceedings of the Second Annual Conference Genetic Programming 1997 (GP97)*, pages 9–17, MIT Press, 1997

[Iba and de Garis, 1996] Iba, H. and de Garis, H. Extending Genetic Programming with Recombinative Guidance, In Angeline, P. and Kinnear, Jr. K. editors, *Advances in Genetic Programming 2*, pages 69–88, MIT Press, 1996

[Iba *et al.*, 1995] Iba, H., de Garis, H. and Sato, T. Recombination Guidance for Numerical Genetic Programming, In *Proceedings of the 1995 IEEE International Conference of Evolutionary Computation*, IEEE Press, 1995

[Iba *et al.*, 1994] Iba, H., deGaris, H. and Sato, T. *Genetic Programming using a Minimum Description Length Principle*, In Kinnear, Jr. K. editor, *Advances in Genetic Programming* , pages 265–284, MIT Press, 1994

[Ito *et al.*, 1996b] Ito, T., Iba, H. and Kimura, M. Robustness of Robot Programs Generated by Genetic Programming, In Koza, J., Goldberg, D., Fogel, D. and Riolo, R. editors, *Proceedings of the First Annual Conference Genetic Programming 1996 (GP96)*, pages 321–326, MIT Press, 1996

[Ito *et al.*, 1998a] Ito, T. and Iba, H. and Sato, S. Non-Destructive Depth-Dependent Crossover for Genetic Programming, In *Proceedings of the First European Workshop on Genetic Programming (EuroGP'98)*, pages 71–82, Springer-Verlag, 1998

[Ito *et al.*, 1998b] Ito, T., Iba, H. and Sato, S. Depth-Dependent Crossover for Genetic Programming, In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*, pages 775–780, IEEE Press, 1998

[Ito *et al.*, 1999] Ito, T., Iba, H. and Sato, S. A Self-Tuning Mechanism for Depth-Dependent Crossover, In *Advances in Genetic Programming 3*, MIT Press, 1999

[Kinnear, 1993] Kinnear, Jr. K. *Generality and Difficulty in Genetic Programming: Evolving a Sort*, In *Proceedings of 5th International Joint Conference on Genetic Algorithms* MIT press, 1993

[Koza, 1991] Koza, J. R. Evolution of Subsumption using Genetic Programming, In Varela, F. and Bourgine, P. editors, *Proceedings of the First European Conference on Artificial Life. Towards a Practice of Autonomous Systems (ECAL-91)*, MIT press, 1991

[Koza, 1992a] Koza, J. R. Genetic Programming: On the Programming of Computers by Natural Selection, MIT press, 1992

[Koza, 1994a] Koza, J. R. Genetic Programming II: Automatic Discovery of Reusable Programs, MIT press, 1994

[Koza, 1994b] Koza, J. R. Evolution of a Subsumption Architecture that Performs a Wall Following Task for an Autonomous Mobile Robot via Genetic Programming, In Petshe T. editor, *Computational Learning Theory and Natural Learning Systems*, pages 321-346, MIT Press 1994

[Koza and Rice, 1992] Koza, J. R. and Rice, J. P., Automatic Programming of Robots using Genetic Programming, In *Proceedings of Tenth National Conference on Artificial Intelligence*, AAAI Press / MIT Press, 1992

[Langdon, 1998] Langdon, W. B. Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming! Kluwer Academic Publishers, 1998

[Langdon and Poli, 1997] Langdon, W. B. and Poli, R. Fitness Causes Bloat, In *2nd Online World Conference on Soft Computing in Engineering Design and Manufacturing (WSC2)*, 1997

[Langdon and Poli, 1998a] Langdon, W. B. and Poli, R. Why Ants are Hard, In Koza, J., Banzhaf, W., Chellapilla *et al.* editors, *Proceedings of the Third Annual Conference Genetic Programming 1998 (GP98)*, pages 193–201, MIT Press, 1998

[Luke and Spector, 1997] Luke, S. and Spector, L. A Comparison of Crossover and Mutation in Genetic Programming, In Koza, J., Deb, K., Dorigo, M., Fogel, D. Garzon, M., Iba, H. and Riolo, R. editors, *Proceedings of the Second Annual Conference Genetic Programming 1997 (GP97)*, pages 240–248, MIT Press, 1997

[Luke and Spector, 1998] Luke, S. and Spector, L. A Revised Comparison of Crossover and Mutation in Genetic Programming, In Koza, J., Banzhaf, W., Chellapilla *et al.* editors, *Proceedings of the Third Annual Conference Genetic Programming 1996 (GP98)*, pages 208–213, MIT Press, 1998

[Maes, 1993] Maes, P., Behavior-Based Artificial Intelligence, In *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB-2)*, MIT Press, 1993

[Mitchell, 1995] Mitchell, M. An Introduction to Genetic Algorithms, MIT press, 1995

[Nordin and Banzhaf, 1995a] Nordin, P. and Banzhaf, W., Genetic Programming Controlling a Miniature Robot, In *Working Notes for the AAAI Symposium on Genetic Programming*, 1995

[Nordin *et al.*, 1995b] Nordin, P. Francone, F. and Banzhaf, W., Explicitly Defined Introns and Destructive Crossover in Genetic Programming, In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, Technical Report of University of Rochester, 1995

[Nordin *et al.*, 1996] Nordin, P., Francone, F. and Banzhaf, W. Explicitly Defined Introns and Destructive Crossover in Genetic Programming, In Angeline, P. and Kinnear, Jr. K. editors, *Advances in Genetic Programming 2*, pages 111–134 MIT Press, 1996

[O'Reilly, 1995] O'Reilly, U.-M. *An Analysis of Genetic Programming*, PhD Thesis, Carleton University, 1995

[O'Reilly and Oppacher, 1994] O'Reilly, U.-M. and Oppacher, F. Program Search with a Hierarchical Variable Length Representation: Genetic Programming, Simulated Annealing and Hill Climbing, In *Parallel Problem Solving from Nature (PPSN III)*, pages 397–406, Springer-Verlag, 1994

[Reynolds, 1992] Reynolds, C. W., An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion, In Meyer and Wilson editors *From Animals to Animats2: Proceedings of Simulation of Adaptive Behaviour (SAB-92)*, MIT Press, 1992

[Reynolds, 1994a] Reynolds, C. W., Evolution of Obstacle Avoidance Behavior: Using Noise to Promote Robust Solutions, In *Advances in Genetic Programming*, MIT Press, 1994

[Reynolds, 1994b] Reynolds, C. W., An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion, In Langton editor *An Evolved, Vision-Based Behavioral Model of Obstacle Avoidance Behaviour*, MIT Press, 1994

[Reynolds, 1994c] Reynolds, C. W., Evolution of Corridor Following Behavior in a Noisy World, In *From Animals to Animats3: Proceedings of Simulation of Adaptive Behaviour (SAB-94)*, MIT Press, 1994

[Rosca, 1997] Rosca, J. P. Hierarchical Learning with Procedural Abstraction Mechanisms, University of Rochester, 1997

[Slavov and Nikolaev, 1997] Slavov, V and Nikolaev, N Fitness Landscapes and Inductive Genetic Programming, In In Smith, G. editor, *Third International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97)*, Springer-Verlag, Vienna, 1997

[Soule and Foster, 1997] Soule, T. and Foster, J. Code Size and Depth Flows in Genetic Programming, In Koza, J., Deb, K., Dorigo, M., Fogel, D. Garzon, M., Iba, H. and Riolo, R. editors, *Proceedings of the Second Annual Conference Genetic Programming 1997 (GP97)*, pages 313–320, MIT Press, 1997

[Soule *et al.*, 1996] Soule, T., Foster, J. and Dickinson, J. Code Growth in Genetic Programming, In Koza, J., Goldberg, D., Fogel, D. and Riolo, R. editors, *Proceedings of the First Annual Conference Genetic Programming 1996 (GP96)*, pages 215–223, MIT Press, 1996

[Tackett, 1994] Tackett, W. A. Recombination, Selection, and the Genetic Construction of Computer Programs, University of Southern California, Department of Electrical Engineering Systems, 1994

[Teller, 1996] Teller, A. Evolving Programmers: The Co-evolution of Intelligent Recombination Operators In Angeline, P. and Kinnear, Jr. K. editors, *Advances in Genetic Programming 2*, pages 45–68, MIT Press, 1996

Publications

[1] Ito, T., Iba, H. and Kimura, M. Robustness of Robot Programs Generated by Genetic Programming, Japan Advanced Institute of Science and Technology, IS-RR-96-0001I, 1996

[2] Ito, T., Iba, H. and Kimura, M. Robustness of Robot Programs Generated by Genetic Programming, In Koza, J., Goldberg, D., Fogel, D. and Riolo, R. editors, *Proceedings of the First Annual Conference Genetic Programming 1996 (GP96)*, pages 321–326, MIT Press, 1996

[3] Ito, T., Iba, H. and Sato, S. Depth-Dependent Crossover for Genetic Programming, WAL (Workshop on Artificial intelligence toward Learning), Abashiri, 1997, in Japanese

[4] 伊庭斉志監訳：遺伝的アルゴリズムの方法 (第 2 章担当)，pages 43–102, 東京電機大学出版局, 1997

[5] Ito, T., Iba, H. and Sato, S. Non-Destructive Depth-Dependent Crossover for Genetic Programming, In *Proceedings of the First European Workshop on Genetic Programming (EuroGP'98)*, pages 71–82, Springer-Verlag, 1998

[6] Ito, T., Iba, H. and Sato, S. Depth-Dependent Crossover for Genetic Programming, In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*, pages 775–780, IEEE Press, 1998

[7] Ho, T. B., Nguyen, T. D., Nguyen, N. B. and Ito, T. Development of Some Methods and Tools for Discovering Conceptual Knowledge, In *Proceedings of the Discovery Science 1998 the First International Conference on Discovery Science (DS'98)*, Springer-Verlag, 1998

[8] Ito, T., Iba, H. and Sato, S. A Self-Tuning Mechanism for Depth-Dependent Crossover, In *Advances in Genetic Programming 3*, MIT Press, 1999