Takuya ITO, Hitoshi IBA and Satoshi SATO

There are three genetic operators: crossover, mutation and reproduction in Genetic Programming (GP). Among these genetic operators, the crossover operator mainly contributes to searching for a solution program. Therefore, we aim at improving the program generation by extending the crossover operator. The normal crossover selects crossover points randomly and destroys building blocks. We think that building blocks can be protected by swapping larger substructures. In our former work, we proposed a depth-dependent crossover. The depth-dependent crossover protected building blocks and constructed larger building blocks easily by swapping shallower nodes. However, there was 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, we propose a self-tuning mechanism for the depth selection probability. We call this type of crossover a "self-tuning depth-dependent crossover". We compare 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.

16.1 Introduction

Recently Genetic Programming (GP) has been applied to various applications, e.g., a robot program [Koza, 1992a], a multi-agent task [Iba, 1997] and an image recognition [Iba et al., 1995]. In case of standard GP, it requires huge computational time to search for a solution program of a large scale problem. This is a critical problem when GP is applied to the large scale problem, such as the automatic generation of a practical program.

In many previous studies, it is shown that the crossover and the selection mechanism mainly contribute to generating a solution program in GP [Koza, 1992b, p. 599]. In this paper, we focus on the crossover operator and aim at generating computer programs efficiently by improving the crossover operator. "Efficient" means that reduction of the number of generations required to generate target programs.

A program structure in GP is generally a tree structure [1]. The normal crossover operator selects randomly a crossover point (node) regardless of its position within the tree structure. Thus, if the normal crossover operator destroys building blocks [2], it will result in the degradation of the search for a solution program.

Swapping a larger structure is one way to solve this difficulty. Because building blocks are larger protected by swapping structures. In our previous paper[Ito

[1] There are also a linear and a graph structure besides the tree structure[Banzhaf et al., 1998, pp.239–276]

[2] A "building block" is a pattern of genes in a contiguous section of a chromosome which, if present, confers a high fitness to the individual [Langdon, 1998, p. 238]

et al., 1998a], we 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 the crossover operator. 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 computational time for the evolution was decreased 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 we could not show any advantage for the fitness performance for the ANT problem [Ito et al., 1998a].

We propose a self-tuning mechanism for the depth selection probability to avoid the difficulty mentioned above. We call this type of crossover a "self-tuning depth-dependent crossover". 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 probability was known for one GP problem in advance, setting the optimal depth selection probability 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.

This paper is organized as follows. Section 16.2 explains the self-tuning mechanism for the depth-dependent crossover. Section 16.3 describes experiments in several tasks and compares GP performances of the self-tuning depth-dependent crossover with performances of the original depth-dependent crossover. Discussion is given in Section 16.4, followed by a conclusion and future work in Section 16.5.

16.2 A Self-Tuning Mechanism for Depth-Dependent Crossover

In case of the depth-dependent crossover[3], the depth selection probability is fixed (half of its parent node's probability). It is a user-defined parameter. If the depth selection probability is not set correctly, the crossover operator may not work well.

[3] Detailed description of the depth-dependent crossover is given in [Ito et al., 1998a]

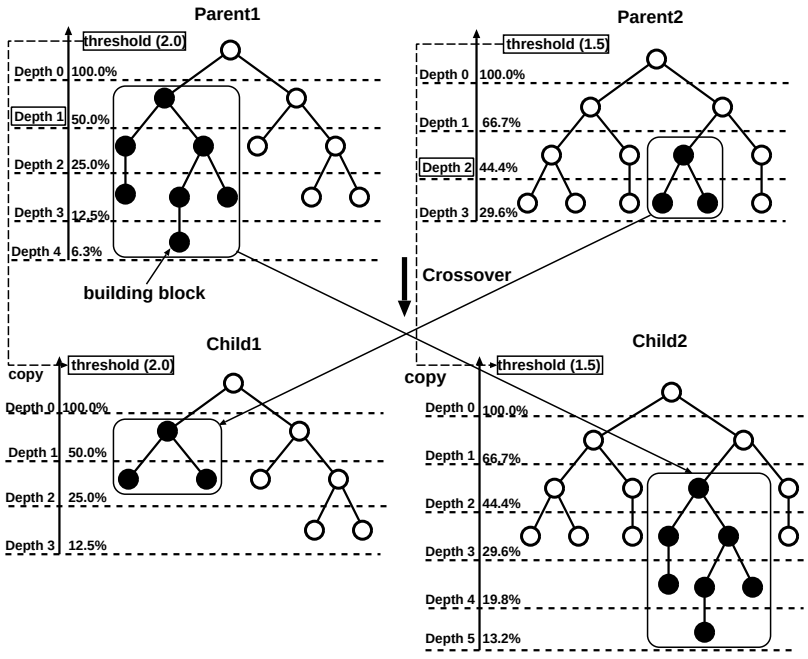

Figure 16.1
Self Tuning without Parameter Crossover. This method copies the depth selection probability of each parent to each child.

To solve this difficulty, we propose a self-tuning mechanism, 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 likely that the crossover operator 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, the crossover will select a deeper node. So that, the selected subtrees will be smaller.

It is difficult to evaluate directly whether each depth selection probability is suitable for each tree structure or not. We do not have any evaluation criteria about the depth selection probability for each GP task. A high depth selection probability may be a good setting on one tree structure, whereas, a low depth selection probability may work well for another tree. In this self-tuning mechanism, we hypothesize that if the depth selection probability is effectively assigned to a tree

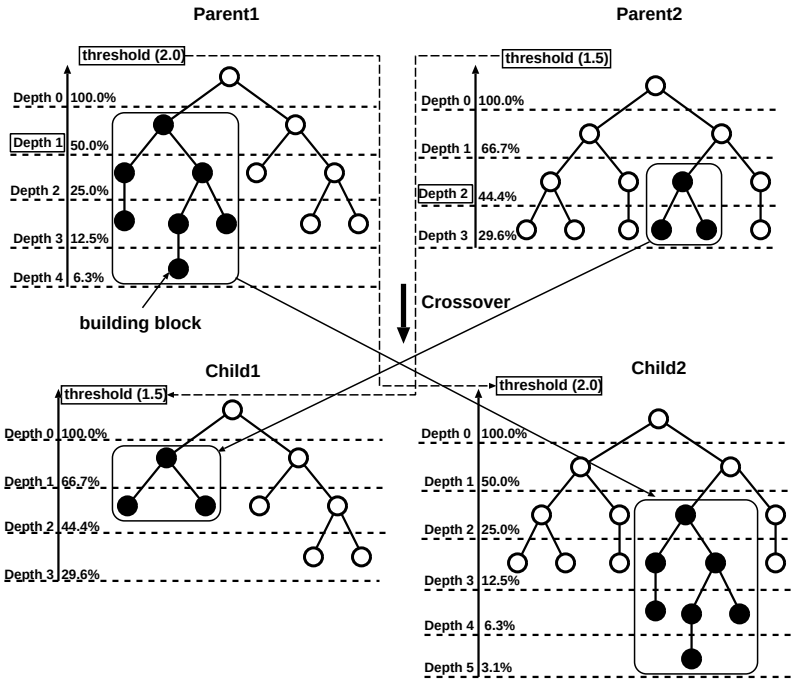

Figure 16.2
Self Tuning with Parameter Crossover. This method also swaps the depth selection probability of each parent when the selected subtrees are swapped.

structure suitably, 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. That is, the depth selection probability of the tree structure, which is selected by fitness selection, is also selected and inherited to the next generation. The advantage is that both the depth selection probability and the program fitness (i.e., program performance) are evaluated by means of fitness selection alone.

We introduce two methods for the above purpose. One is to copy the depth selection probability of each parent to each child (Fig. 16.1). Another method is to swap the depth selection probability (Fig. 16.2). In Fig. 16.1, black nodes means building blocks. The crossover point of each tree is determined by each depth selection probability. As a result of the crossover operator, two children with the same depth selection probability as their parents are generated. In case of 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 are higher (i.e., the probabilities of the depth 3, 4 and 5 are 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%).

In case of Fig. 16.2, 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 Fig. 16.2, 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 are 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 a building block is high (i.e., the probability of depth 2 is 44.4%). Large building blocks of child 2 are protected, whereas small building blocks of child 1 are broken. Therefore, the method shown in Fig. 16.2 is a greedy method. We clarify differences of these methods (i.e, Fig. 16.1 and 16.2) in next section.

The above self-tuning mechanism for the depth-dependent crossover is realized as the following algorithm.

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

Crossover When applying the crossover operator, 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., Fig. 16.2)

Table 16.1
Experimental Set Up

Setting	Crossover	Mutation
NORMAL	Random	Random
DD	Depth-Dependent	Random
SDD	Self-Tuning Depth-Dependent without Parameter Crossover (Fig. 16.1)	Random
SDD-XO	Self-Tuning Depth-Dependent with Parameter Crossover (Fig. 16.2)	Random

Mutation When applying the mutation operator, 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*.

We assign *min-probability* to 1.5 and *max-probability* to 2.0 in this paper (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*. The mutation of the depth selection probability is to escape from a local minimum for the depth selection probability.

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

16.3 Experimental Results

We have investigated the effectiveness of the self-tuning depth-dependent crossover for four GP problems. These problems are Boolean concept formation problems (i.e., the 11MX and the 4EVEN) the ANT and a robot problem. This section briefly explains the results of the 11MX, the ANT and the robot problem. Complete and detailed experimental results are described in [Ito, 1999]) Table 16.1 shows the experimental set up. 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.

16.3.1 The 11MX problem

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, 1992b, p. 170]. The fitness of the 11MX is an error rate for total inputs. The used parameters are the same as [Ito et al., 1998a].

Fig. 16.3 shows the best and the average fitness. Note that the smaller, the better the fitness is. According to the best fitness performance curve, the **DD** shows an ability of which it searches a 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. We statistically examined the best and the average fitness performances at the final generation with the paired t-test [Freund and Wilson, 1992]. Table 16.2 shows the result of t-test. According to this test, we have 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, we have not verified that the **SDD** was superior to the **NORMAL** in terms of the best and the average fitness values. Table 16.3 shows average numbers of hits and its standard deviation at the final generations over twenty runs. Note, 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. According to this table, both of the **DD** and the **SDD-XO** gave best performance among all crossover settings.

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). Fig. 16.4, 16.5 and 16.6 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 tree depth of the **DD** shows the deepest trees among all crossover settings. On the contrary, the pattern of the **SDD-XO** shows the shallower trees than that of the **DD** (Fig. 16.4). The number of function nodes (and terminal nodes) was much larger with **DD** than that with **SDD-XO** (Fig. 16.5 and 16.6). By using the **DD**, a shallower node of the tree structure was selected more frequently. It frequently occurred that a small subtree in the shallower node of the tree structure was replaced

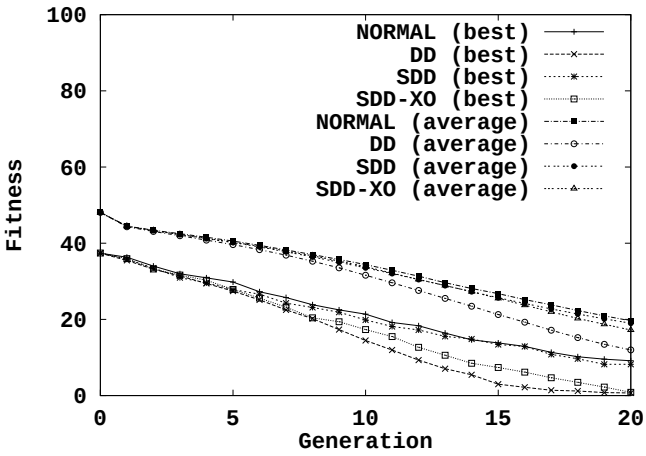

Figure 16.3
Experimental Results, means of twenty runs (11MX). The fitness of the 11MX is an error rate for total inputs. The "Best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

with a large subtree. In that case, the fitness of an individual which received a large subtree (individual 1) was improved because there were large building blocks in the received subtree. On the contrary, the fitness of an individual which received a small subtree (individual 2) was not improved because the size of building blocks in the received subtree was small. As a result, the size of the tree structure became large because individual 1 prospered in the population and individual 2 became extinct. Thus, the selection pressure which enlarged the tree size occurred in case of the **DD**. This phenomenon is called *bloat* [Langdon and Poli, 1997]. Angeline clarified that the crossover operator promoted unnecessary bloat by several experimental results [Angeline, 1998]. The **DD** induced the bloating phenomenon easily by means of the above reason.

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 difference of the depth selection probability of each individual. Fig. 16.7 and 16.8 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.,

Table 16.2
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 16.3
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 four crossover operators.

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

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 was increased during the evolution in case of the **SDD** (Fig. 16.7). 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** (Fig. 16.8). This means that crossover points are often selected at shallower nodes. These phenomena were occurred on the other GP problems, i.e., the 4EVEN, the ANT and the robot problem [Ito, 1999].

The original depth-dependent and the self-tuning depth-dependent crossover have an effect that deep subtrees are swapped more often. This will result in the protection of building blocks. We can consider that the fitness performance may be improved because of this effect on the 11MX problem. Fig. 16.9 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 in the early generations. However, the **DD** swaps deeper subtrees as the evolution proceeds. The **SDD-XO** swapped shallower subtrees than the **DD**. On the contrary, there was no difference between the **SDD** and the **NORMAL**. These

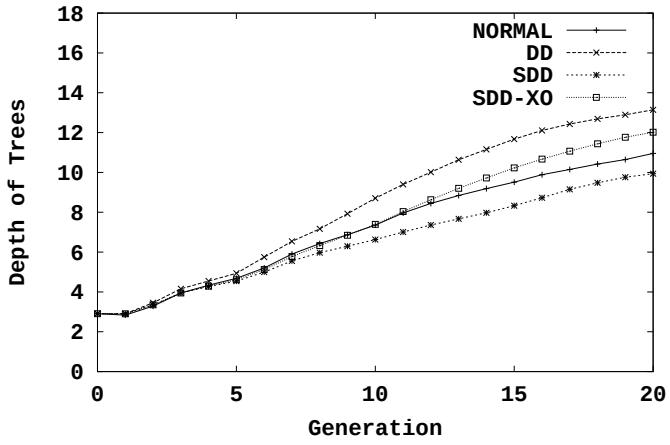

Figure 16.4
Depth of Trees (11MX)

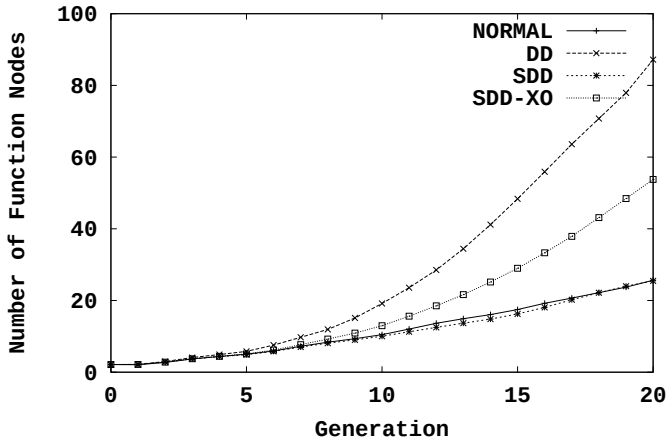

Figure 16.5
Number of Function Nodes (11MX)

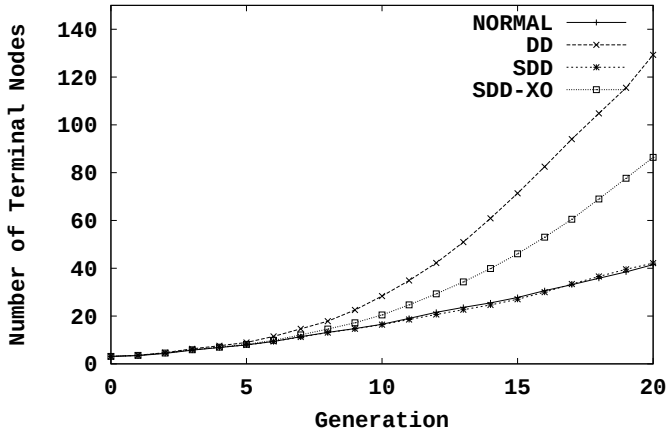

Figure 16.6
Number of Terminal Nodes (11MX)

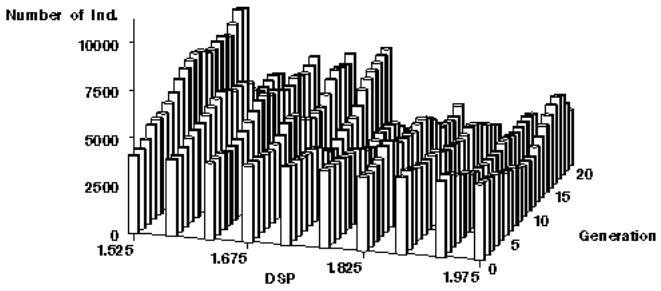

Figure 16.7
Transitions of the Depth Selection Probability (11MX, **SDD**) during the Evolution. DSP means depth selection probability. The numbers of individuals (z axis) are accumulated values over twenty runs.

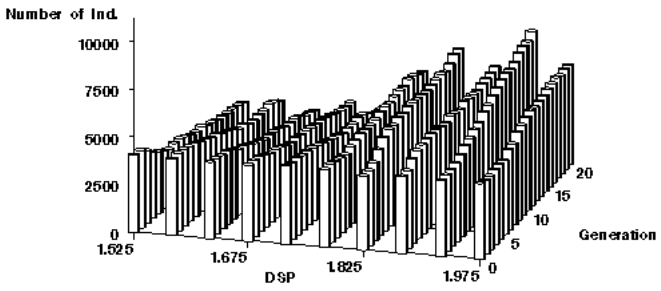

Figure 16.8
Transitions of the Depth Selection Probability (11MX, **SDD-XO**) during the Evolution. DSP means depth selection probability. The numbers of individuals (z axis) are accumulated values over twenty runs.

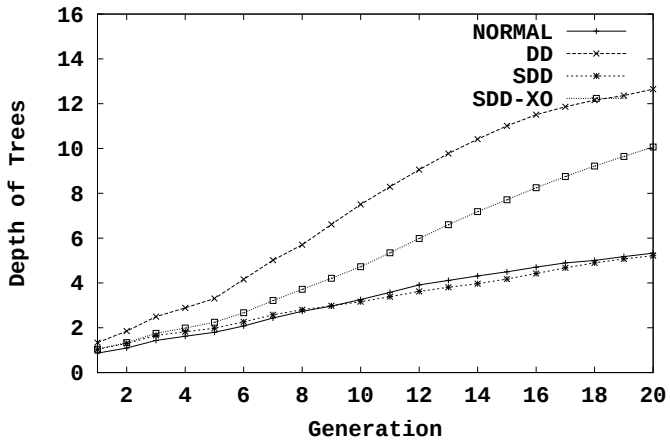

Figure 16.9
Average Absolute Depth of Swapped Tree Structure (11MX)

phenomena are also observed in the numbers of nodes of the swapped subtrees (see [Ito, 1999]). In case of the **DD**, this phenomenon was related to the tree growth and the fixed depth selection probability. In case of the **DD**, the depth selection probability of the crossover operator depended on an absolute tree depth, not a relative tree depth. Besides, the depth selection probability did not changed during the evolution. Due to these two factors, the depth of swapped subtrees became deeper gradually with the growth of the tree structure on the **DD**. A node of which was closer to the root node was selected frequently.

When we compare two particular crossover operators, 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 operator is better if tree structures become large. A large structure requires huge computer memory and computational time [Ito et al., 1998b]. 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, we can claim that the **SDD-XO** is superior than the **DD** on the 11MX problem.

16.3.2 The ANT problem

Next, we show experimental results on the ANT problem. The ANT problem is to generate a program to each as much food as possible given a limited amount of energy (400 units) [Koza, 1992b, p. 54]. The ANT can move forward, turn to the right and left. It consumes one unit of energy when it moves. The fitness of the ANT is the fraction of all available food that the ant could not eat. In other words, 1.0 means that the ant could not eat any of the 89 units of foods and 0.0 means that the ant could eat all the food within the limited energy. The used parameters were the same as [Ito et al., 1998a].

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, 1998] and the depth-dependent crossover was not effective on this problem [Ito et al., 1998a].

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

Fig. 16.10 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

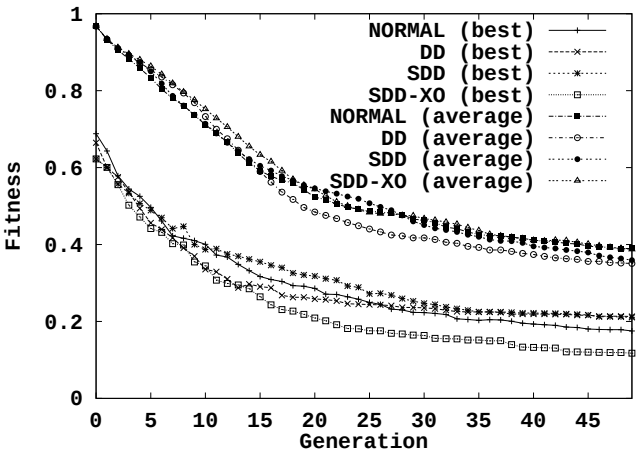

Figure 16.10
Experimental Results, means of twenty runs (ANT, Experiment 1, i.e., the maximum depth for a new tree is 10). The fitness of the ANT is a probability for which the ant could not eat 89 foods. The "Best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

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. We examined these results using paired t-test and we have confirmed that the **SDD-XO** is superior to the **DD** and the **SDD** on the best value (Table. 16.4). On the hits measure, the **SDD-XO** give the best performance among all crossover settings (Table 16.5).

Fig. 16.11 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. We examined these results using paired t-test, and confirmed that the **SDD-XO** was better than the **DD** (Table. 16.4). On the hits measure, the **SDD-XO** gave the best performance among all crossover settings (Table 16.5).

As mentioned above, in case of the **NORMAL**, the number of the hits in experiment 2 (i.e., the maximum depth for a new tree is 5) was improved over experiment

Table 16.4
Statistic t for the Best and the Average Fitness Values at the Final Generation (ANT).

Setting	Experiment 1		Experiment 2	
	Best	Average	Best	Average
DD (against **NORMAL**)	-0.96	1.14	0.29	1.56
SDD (against **NORMAL**)	-0.82	-0.63	1.35	2.61
SDD-XO (against **NORMAL**)	1.34	0.06	1.69	1.26
SDD (against **DD**)	-0.03	-0.16	1.15	1.55
SDD-XO (against **DD**)	2.53	-0.89	1.78	0.09
SDD-XO (against **SDD**)	2.47	-0.90	0.59	-2.46

Table 16.5
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 four crossover operators.

Setting	Experiment 1			Experiment 2		
	Hits	Standard Deviation	Rank	Hits	Standard Deviation	Rank
NORMAL	0.15	0.36	2	0.20	0.40	4
DD	0.10	0.30	3	0.25	0.43	3
SDD	0.10	0.30	3	0.45	0.50	2
SDD-XO	0.35	0.48	1	0.60	0.49	1

1 (i.e., the maximum 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 1 (Table 16.5).

These phenomena are related to the growth of the depth of the tree. Fig. 16.12 shows the depth of the tree of experiment 1 and that of experiment 2, respectively. According to this 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 the tree grows quickly as the evolution proceeds. On the contrary, in case of experiment 2, the depth of the tree became deep slowly during the evolution. At the last generation, the depth of the tree was almost same value (about 15) for both experiments. In case of experiment 2, the crossover operator generated building blocks effectively because the depth of the tree was shallow at early generations. On the contrary, the crossover operator 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 than for experiment 1.

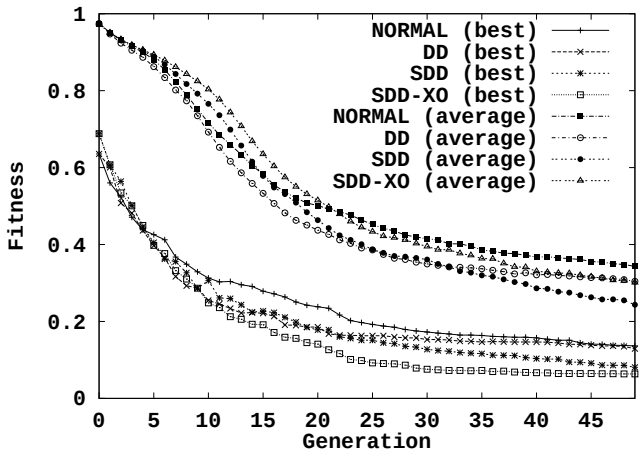

Figure 16.11
Experimental Results, means of twenty runs (ANT, Experiment 2, i.e., the maximum depth for a new tree is 5). The fitness of the ANT is a probability for which the ant could not eat 89 foods. The "Best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

16.3.3 The robot problem

We verified the effectiveness of our proposed crossover operators on a robot problem. This is to show the feasibility of the depth-dependent crossover operators for a real-world task. An autonomous robot simulator was constructed (Fig. 16.13). The model 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, a wall and obstacles. (Fig. 16.14). The simulated robot has three types of sensors. They are eight bump sensors, an eye sensor and a beacon sensor (Table 16.6).

The robot task is to collect a target object (see [Ito, 1999] for the details). The robot can discriminate between the target object and obstacles by their colors.

As commonly used in most GP applications[Ito et al., 1996], the robot commands were considered as terminals, i.e., 0-argument function (Table 16.7). 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 a target object to the station correctly. The maximum number of time-steps is set to be 300.

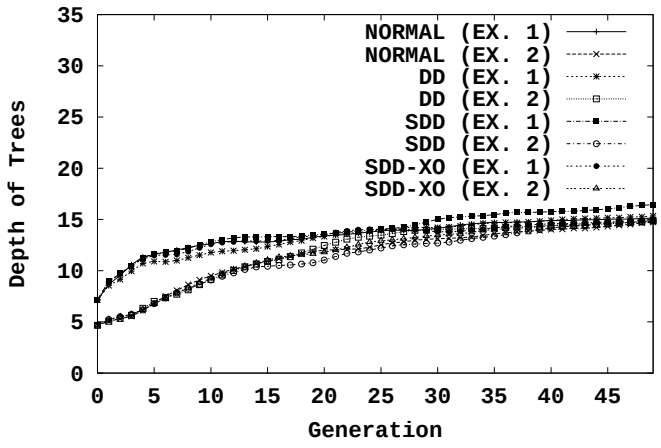

Figure 16.12
Depth of Trees (ANT, Experiments 1 and 2)

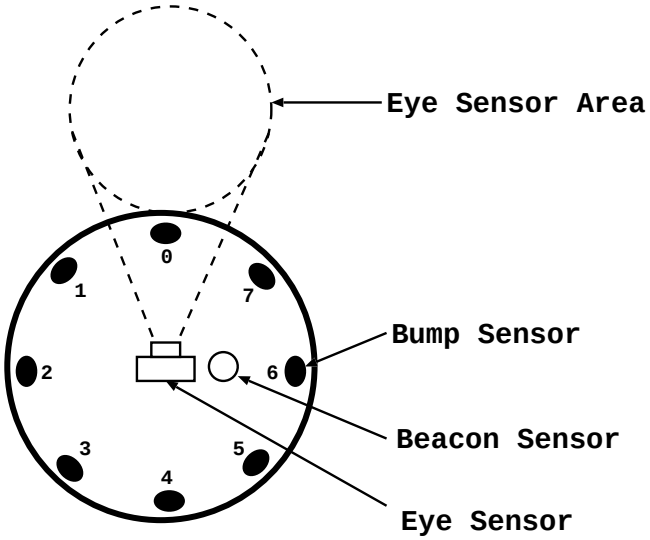

Figure 16.13
The Simulated Robot. The model is a behavior-based robot.

Table 16.6
Sensors of the Simulated Robot

Type	Number	Function
Bumped 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.

Table 16.7
Terminal Set (Robot)

ID	Type	Function
0 ∼ 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.

Table 16.8
Function Set (Robot)

ID	Type	Function
0∼7	**(BS***ID* **p0 p1)**	Evaluates p0 if the bumped *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 an 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 an 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).

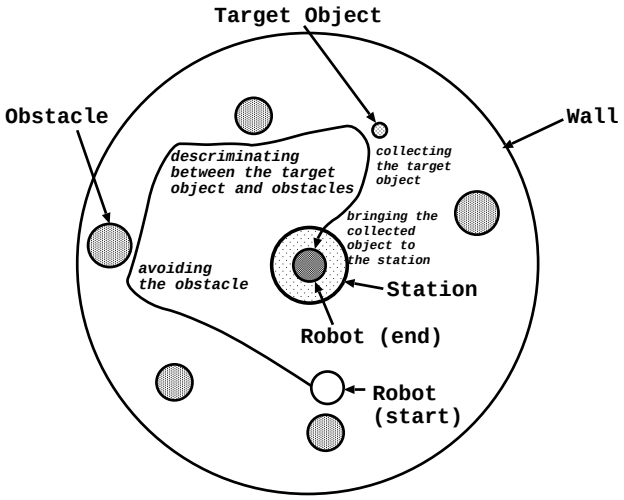

Figure 16.14
The Workspace of the Robot. There are only five types of objects, i.e., the robot, a target object, a station, a wall and obstacles on this workspace.

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 deadlock). To prevent this deadlock, the terminal set includes five **GF** commands.

In order to make the robot respond to its sensor inputs, the 14 functional nodes were introduced (Table 16.8).

The following fitness function is used for this robot task:

$$Fitness = \begin{cases} dist(S, R_t) & \text{if the robot has the} \\ & \text{target object in hand} \\ dist(S, RC_t) + dist(R_t, RC_t) & \text{otherwise} \end{cases} \quad (16.1)$$

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.

Fig. 16.15 plots the best and the average fitness values for the robot problem (the used parameters were the same as [Ito, 1999]). According to this figure, the

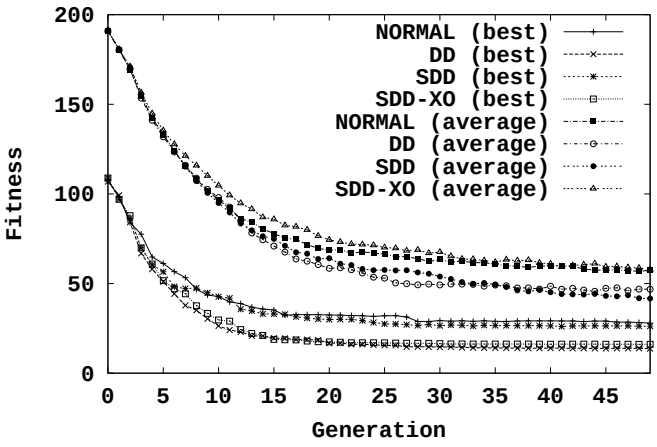

Figure 16.15
Experimental Results, means of twenty runs (Robot). The fitness of the Robot is derived from equation 16.1. The "Best" means the fitness of the best individual and the "average" indicates the average fitness of the population.

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 16.9 shows the result of t-test. We have confirmed that **DD** and **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, we have not verified that the **SDD-XO** was superior to the **NORMAL**.

Table 16.10 shows the averaged numbers of hits 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.

16.4 Discussion

We have hypothesized that, via a self tuning mechanism, if the depth selection selection probability which is suitable for a tree structure is assigned to the tree structure, a fitness of the tree structure is improved. We designed our self-tuning mechanism in accordance with this hypothesis.

Our experimental results have shown that the **SDD-XO** gave good performance

Table 16.9
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 16.10
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 operators.

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

for all the four GP problems. For these problems, each individual has various kinds of depth selection probabilities in early generations because the probability was assigned randomly. However, the number of individuals with high depth selection probability increased gradually as the evolution proceeded (Fig. 16.8). As a result, the fitness performance was improved. According to Fig. 16.8, the self-tuning mechanism worked so as to search for building blocks during early generations and to protect completed building blocks in later generations. We believe that the **SDD-XO** will work for various GP problems due to this effect, i.e., adaptability.

16.5 Conclusion

The goal of this work was to consider an effective search method for applying GP to large scale problems. For this purpose, we proposed the self-tuning depth-dependent crossover. We verified experimentally the effectiveness of the crossover operation. As a result of experiments, the following points have been made clear:

1. The self-tuning depth-dependent crossover worked effectively for various GP problems, i.e., the 11MX, the 4EVEN, the ANT and the robot problem.

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

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]. These structures are quite different from the tree structure. The effectiveness of the depth-dependent crossover operations is not clear for these structures. The future research will evaluate depth-dependent crossover on these structures.

Acknowledgments

We would like to thank W. B. Langdon, Una-May O'Reilly and Peter J. Angeline for their helpful comments.

Bibliography

Angeline, P. J. (1998), "Subtree crossover causes bloat," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo (Eds.), pp 745–752, University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998), *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*, Morgan Kaufmann, dpunkt.verlag.

Freund, R. and Wilson, W. (1992), *Statistical Methods*, Academic Press, Inc.

Iba, H. (1997), "Multiple-agent learning for a robot navigation task by genetic programming," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), pp 195–200, Stanford University, CA, USA: Morgan Kaufmann.

Iba, H., de Garis, H., and Sato, T. (1995), "Recombination guidance for numerical genetic programming," in *1995 IEEE Conference on Evolutionary Computation*, volume 1, p 97, Perth, Australia: IEEE Press.

Ito, T. (1999), *Efficient Program Generation by Genetic Programming*, PhD thesis, Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Tatsunokuchi, Nomi, Ishikawa, 923-1292 Japan.

Ito, T., Iba, H., and Kimura, M. (1996), "Robustness of robot programs generated by genetic programming," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), Stanford University, CA, USA: MIT Press, 321–326.

Ito, T., Iba, H., and Sato, S. (1998a), "Depth-dependent crossover for genetic programming," in *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pp 775–780, Anchorage, Alaska, USA: IEEE Press.

Ito, T., Iba, H., and Sato, S. (1998b), "Non-destructive depth-dependent crossover for genetic programming," in *Proceedings of the First European Workshop on Genetic Programming*, W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty (Eds.), volume 1391 of *LNCS*, pp 71–82, Paris: Springer-Verlag.

Koza, J. R. (1992a), "Evolution of subsumption using genetic programming," in *Proceedings of the First European Conference on Artificial Life. Towards a Practice of Autonomous Systems*, F. J. Varela and P. Bourgine (Eds.), pp 110–119, Paris, France: MIT Press.

Koza, J. R. (1992b), *Genetic Programming: On the Programming of Computers by Natural Selection*, Cambridge, MA, USA: MIT Press.

Langdon, W. B. (1998), *Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming!*, Boston: Kluwer.

Langdon, W. B. and Poli, R. (1997), "Fitness causes bloat," in *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, P. K. Chawdhry, R. Roy, and R. K. Pan (Eds.), Springer-Verlag London.

Langdon, W. B. and Poli, R. (1998), "Why ants are hard," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo (Eds.), pp 193–201, University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann.

Maes, P. (1993), "Behavior-Based Artificial Intelligence," in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB-92)*, MIT Press.