Evolutionary Algorithm Considering Program Size: Efficient Program Evolution using GRAPE

Shinichi Shirakawa
Graduate School of Environment and Information Sciences
Yokohama National University
79-7, Tokiwadai, Hodogaya-ku, Yokohama
Kanagawa, 240-8501, Japan
shirakawa@nlab.sogo1.ynu.ac.jp

Tomoharu Nagao
Graduate School of Environment and Information Sciences
Yokohama National University
79-7, Tokiwadai, Hodogaya-ku, Yokohama
Kanagawa, 240-8501, Japan
nagao@ynu.ac.jp

ABSTRACT

Today, a lot of Automatic Programming techniques have been proposed and applied various fields. Graph Structured Program Evolution (GRAPE) is one of the recent Automatic Programming techniques. GRAPE succeeds in generating the complex programs automatically. In this paper, a new generation alternation model for GRAPE, called Evolutionary Algorithm Considering Program Size (EACP), is proposed. EACP maintains the diversity of program size in the population by using particular fitness assignment and generation alternation. We apply EACP to three test problems, factorial, exponentiation and sorting a list. And we show the effectiveness of EACP and confirm evolution of maintaining the diversity of program size.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Experimentation, Performance

Keywords

Automatic Programming, Genetic Programming, Generation Alternation Model, Graph-based Genetic Programming, Evolutionary Algorithm, Genetic Algorithm

1. INTRODUCTION

Automatic Programming is the method of generating computer programs automatically. Genetic Programming (GP) [4] is a typical example of Automatic Programming, which was originally introduced by Koza. GP evolves computer programs, which are usually tree structure, and searches a desired program using Genetic Algorithm (GA). GP has a tendency to create programs with unnecessarily large size [4]. This phenomenon is called *bloat*. To prevent *bloat*, a lot of ideas are investigated. One of the ideas to prevent *bloat* is Multi Objective Genetic Programming (MOGP). Bleuler et al. [1] use Strength Pareto Evolutionary Algorithm 2 (SPEA2) [12] to control program size and reduce *bloat* in standard GP. Two objectives are considered: functionality of the program and code size. SPEA2 incorporates a close-grained fitness assignment strategy and an adjustable elitism scheme. SPEA2 shows better performance compared with respect to standard GP, Constant Parsimony Pressure [9] and Adaptive Parsimony Pressure [11] in several even-parity problems.

Various representations for GP have been proposed so far including graph representation[3, 5, 10]. **GRA**ph structured **P**rogram **E**volution (GRAPE) [7, 8] is one of the recent Automatic Programming technique. GRAPE succeeds in generating the complex programs automatically (e.g. factorial, exponentiation, sorting a list and so on). The work described in this paper is based on this GRAPE technique. We believe that various program size should be searched in evolutionary process. Thus, it is necessary to maintain the diversity of program size in the population. In this paper, a new generation alternation model for GRAPE, called Evolutionary Algorithm Considering Program Size (EACP), is proposed. EACP maintains the diversity of program size in the population by using particular fitness assignment and generation alternation. We apply EACP to three problems, factorial, exponentiation and sorting a list. The present work is intended to take into account the effectiveness of EACP and confirm evolution maintaining various program size.

2. RELATED WORKS

2.1 Graph Structured Program Evolution (GRAPE)

GRAPE [7, 8] constructs graph structured programs automatically. The graph structured programs is composed of arbitrary directed graph of nodes and *data set*.

The features of GRAPE are summarized as follows:

- Arbitrary directed graph structures.
- Handle multiple data types using the *data set*.
- Genotype of integer string.

The representation of GRAPE is graph structure. Each program is constructed as an arbitrary directed graph of nodes and *data set*. The *data set* flows the directed graph

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-130-9/08/07.

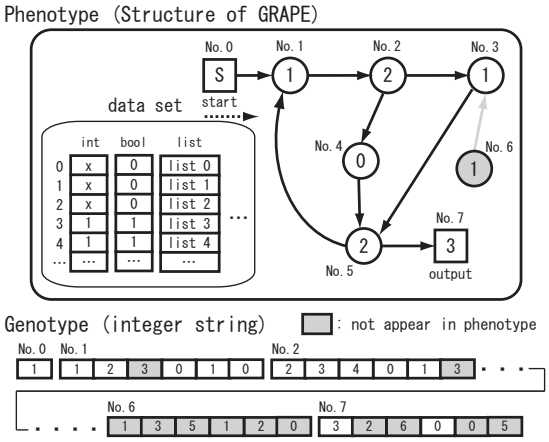

Figure 1: Structure of GRAPE (phenotype) and the genotype which denotes a list of node types, connections and arguments. "No.6" is the *inactive node*. Size of program (the number of active nodes) is 7.

and is processed at each node. Figure 1 illustrates an example of structure of GRAPE. Each node in GRAPE has two parts, a *processing* and *branching*. The *processing* executes several kinds of processing using the *data set*, for instance, arithmetic calculation and boolean calculation. After the *processing* is executed, a next node is selected. The *branching* decides the next node according to the *data set*. "No.7 node" is the *output node*. When this node is reached, the GRAPE program outputs data and then the program halts. The representation of GRAPE is graph structure, therefore it can represent complex programs (e.g. branches and loops) using its graph structure. There are several data types in GRAPE program, integer data type, boolean data type, list data type and so on. The GRAPE program handles multiple data types using the *data set* for each type.

To adopt evolutionary method, genotype-phenotype mapping is used in GRAPE. This genotype-phenotype mapping method is similar to CGP [5]. The GRAPE program is encoded in the form of a linear string of integers. The genotype is an integer string which denotes a list of node types, connections and arguments. Although the genotype in GRAPE is a fixed length representation, the number of nodes in the phenotype can vary but is bounded (not all of the nodes encoded in the genotype have to be connected). This allows the existence of *inactive nodes*. In Figure 1, "No.6 node" is an *inactive node*. The other nodes are *active nodes*. In this paper, program size of GRAPE is the number of *active nodes* in the phenotype.

To obtain the optimum structure of GRAPE, an evolutionary method is adopted. The genotype of GRAPE is a linear string of integers. Therefore, GRAPE is able to use a usual GA. In this paper we use uniform crossover and mutation as the genetic operators.

2.2 Minimal Generation Gap (MGG)

The MGG model [2, 6] is a steady state model proposed by Satoh et al. The MGG model has a desirable convergence property maintaining the diversity of the population, and shows higher performance than the other conventional models in a wide range of applications (especially real-parameter optimization). The MGG model is summarized as follows:

1. Set generation counter $t = 0$. Generate N individuals randomly as the initial population $P(t)$.
2. Select a set of two parents M by random sampling from the population $P(t)$.
3. Generate a set of m children C by applying the crossover and the mutation operation to M.
4. Select two individuals from set $M + C$. One is the elitist individual and the other is the individual by the roulette-wheel selection. Then replace M with the two individuals in population $P(t)$ to get population $P(t + 1)$.
5. Stop if a certain specified condition is satisfied, otherwise set $t = t + 1$ and go to step 2.

The MGG model localizes its selection pressure not to the whole population as Simple GA or Steady State does, but only to the family (children and parents).

GRAPE with MGG model shows higher performance than GRAPE with Simple GA in a variety of evolution of programs [8].

3. EVOLUTIONARY ALGORITHM CONSIDERING PROGRAM SIZE (EACP)

3.1 Overview

The EACP algorithm maintains the diversity of program size in the population by using particular fitness assignment and generation alternation. Each individual is evaluated by considering *functional fitness value* and *program size* of the individual in *EACP fitness* assignment. The EACP algorithm also localizes its selection pressure to the family (children and parents) as in the case of MGG.

3.2 The EACP Algorithm

Figure 2 shows the outline of generation alternation in EACP. The EACP algorithm is designed to maintain the diversity of program size in the population. The EACP algorithm is summarized as follows:

Definition:
$P(t)$: Population at generation t
N: Population size
M: Parents
C: Children
m: Child size

Step 1. Initialization:
Set generation counter $t = 0$. Generate N individuals randomly as the initial population $P(t)$.

Step 2. Selection of Parents:
Select a set of two parents M by random sampling from the population $P(t)$.

Step 3. Recombination:
Generate a set of m children C by applying the crossover and the mutation operation to M.

Step 4. EACP Fitness Assignment:
EACP fitness assignment to children C and parents M using $P(t) + C$ (*cf.* Section 3.3).

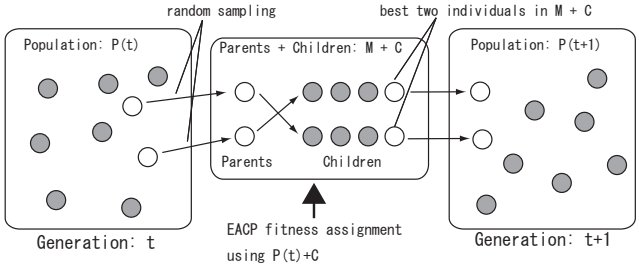

Figure 2: Outline of generation alternation in EACP.

Step 5. Selection:

Select EACP fitness best and 2nd best individuals from set $M+C$. Then replace the two parents M with these two individuals in population $P(t)$ to get population $P(t+1)$.

(When the individuals A and B have same EACP fitness value, the individual which has greater functional fitness value is evaluated better. i.e. $(F_{eacp}(A) = F_{eacp}(B)) \wedge (F_A > F_B) \Rightarrow$ A is evaluated better individual.)

Step 6. Stop Criteria:

Stop if a certain specified condition is satisfied, otherwise set $t = t + 1$ and go to step 2.

3.3 EACP Fitness Assignment

To keep the diversity of program size in the population, EACP uses particular fitness assignment. The EACP fitness assignment is considered the distribution of program size in the population. Specifically, each individual is assigned the EACP fitness for each program size. If there are a lot of same program size individuals in the population, these individuals should be assigned low fitness form concept of the EACP algorithm.

The EACP fitness assignment procedure is as follows:

1. Each individual $i \in (M + C)$ (parents and children) is calculated a value of $n(i)$ using Equation 1. $n(i)$ is the number of individuals which have same program size and equal or greater functional fitness value of the individual i.

$$n(i) = \sum_{j \in P(t)+C, j \neq i} x_i(j),$$

$$\text{where } x_i(j) = \begin{cases} 1 & \text{if } (S_i = S_j) \wedge (F_i \leq F_j) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where S_i is the program size of individual i, F_i is the functional fitness value of individual i. In this case, the higher the numerical value F indicates the better performance. In GRAPE, program size S is the number of active nodes in the phenotype.

2. Each individual $i \in (M + C)$ is assigned the EACP fitness value $F_{eacp}(i)$ in Equation 2.

$$F_{eacp}(i) = \frac{1}{n(i) + 1} \quad (2)$$

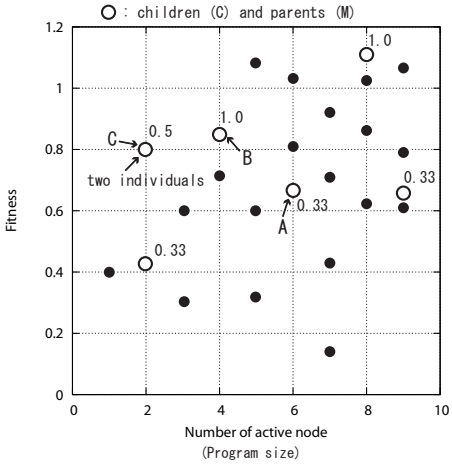

Figure 3: Example of fitness assignment in EACP.

Table 1: The parameters of each algorithm used in the experiments.

Parameter	Value
The number of evaluations	5000000
Population size (N)	500
Child size (m)	50 (for MGG, EACP)
Uniform Crossover rate P_c	0.1
Crossover rate	0.7
Mutation rate P_m	0.02
The maximum number of nodes	30

Figure 3 illustrates an example of fitness assignment of EACP. In Figure 3, the individual A has two individuals which are same program size and greater functional fitness value (i.e. $n(A) = 2$). Thus, the EACP fitness value is 0.33. The individual B has no individual which are same program size and greater functional fitness value (i.e. $n(B) = 0$), and the EACP fitness value is 1.0. The individual C has one individual which are same program size and equal functional fitness value (i.e. $n(C) = 1$), and the EACP fitness value is 0.5.

4. SETTINGS OF EXPERIMENTS

Several different problems are tackled in order to verify the effectiveness of EACP. The EACP algorithm is compared to Simple GA (SGA), MGG and SPEA2 on a number of test problems. The problems include the computations of factorial, exponentiation and sorting a list. We use GRAPE as an Automatic Programming technique. Evolution of these programs is difficult for standard GP. It needs to prepare iteration or recursion mechanisms to solve these problems.

The parameters of each algorithm are given in Table 1. In SGA, tournament selection (a tournament size of 2) along with elitist strategy (an elite size of 1) is used as the selection mechanism. In SPEA2, we use an archive size of 500, and two objectives are considered: functional fitness value and program size (the number of active nodes). Therefore, SPEA2 has a tendency to generate small program size indi-

viduals. The maximum number of nodes of GRAPE (maximum program size) is 30. In order to avoid the problem caused by non-terminating structures we limited the execution step to 500 (factorial and exponentiation) and 3000 (sorting a list). When a program reaches the execution limit, the individual is assigned the fitness 0.0. We prepare sufficient *data set* size to compute the problems. Initially, we set input values and constant values on the *data set*. Therefore, GRAPE handles or creates constants within its programs.

In the problem of factorial, we seek to evolve an implementation of the factorial function. We use integers from 0 to 5 as the training set. The functional fitness F used in the experiments is:

$$F = 1 - \frac{\sum_{i=1}^{n} \frac{|Correct_i - Out_i|}{|Correct_i| + |Correct_i - Out_i|}}{n} \quad (3)$$

where $Correct_i$ is correct value for the training data i, Out_i is the value returned by the generated program for the training data i, and n is the size of the training set. The range of this fitness function is [0.0, 1.0]. The higher the numerical value indicates the better performance. If this functional fitness is equal to 1.0, the program gives the perfect solution for the training set. If the functional fitness in Equation 3 is reached 1.0, the functional fitness is calculated as follows:

$$F = 1.0 + \frac{1}{S_{exe}} \quad (4)$$

where S_{exe} is the total number of execution steps of the generated program. This fitness function means the less execution step is the better solution. In the experiment of factorial, integer data type is used, and the size of integer data in GRAPE is 10. Initially, we set input value on the $data$[0] to $data$[4] and constant value 1 on the $data$[5] to $data$[9]. The node functions used in this experiment are $\{+, -, *, =, >, <, OutputInt\}$. For instance, "+" is add $data$[x] to $data$[y] and substitute for $data$[z]. On the "=", if $data$[x] is equal $data$[y] connection 1 is chosen else connection 2 is chosen.

In the problem of exponentiation, we seek to evolve an implementation of the integer exponential a^b. There are two inputs in this problem. We also use the functional fitness in Equation 3 and 4 on the training set. In the experiment of exponentiation, integer data type is used, and the size of integer data in GRAPE is 15. Initially, we set input value a on the $data$[0] to $data$[4], input value b on the $data$[5] to $data$[9] and constant value 1 on the $data$[10] to $data$[14]. The node functions used in this experiment are $\{+, -, *, =, >, <, OutputInt\}$.

In the problem of sorting a list, we seek to evolve an implementation of the sorting algorithm. We provide a list of integers as the inputs. A correct program returns a sorting input list, of any length (e.g. input: (2 1 7 5 1), output: (1 1 2 5 7)). The training data set is 30 random lists whose lengths are between 10 and 20. Elements of the list are randomly chosen from the range of [0, 255]. The functional fitness function F used in this experiment is given in Equation 5. The range of this fitness function is [0.0, 1.0]. The higher the numerical value indicates the better performance. If this functional fitness is equal to 1.0, the program gives the perfect solution for the training set.

$$F = 1.0 - \frac{\sum_{i=1}^{n} \frac{\sum_{j=0}^{l} (1 - \frac{1}{2^{d_{ij}}})}{l_i}}{n} \quad (5)$$

where d_{ij} is the distance between the correct position and the return value position for the training data i for the element j. l_i is the length of the list for the training data i and n is the size of the training data set. If the functional fitness in Equation 5 is reached 1.0, the functional fitness is calculated using Equation 4. We prepare simple node functions, *arithmetic functions*, *swap the elements of list* and *compare the elements of list*. For instance, "*SwapList*" swaps $list$[$data$[x]] for $list$[$data$[y]], "*EqualList*" is that if $list$[$data$[x]] is equal $list$[$data$[y]] connection 1 is chosen else connection 2 is chosen. We do not prepare special node functions such as *iteration functions*. In this experiment, a list of integers and integer data type are used, and the size of integer data in GRAPE is 15. Initially, we set the size of input list (the list length) on the $data$[0] to $data$[4], constant value 0 on the $data$[5] to $data$[9] and constant value 1 on the $data$[10] to $data$[14].

5. RESULTS AND DISCUSSION

Results are given for 100 different runs with the same parameter set. GRAPE with EACP is successful in finding correct solutions to all of the problems.

The transitions of average functional fitness value over 100 runs for each algorithm are display in Figure 4. This graph also shows the effectiveness of EACP. From the point of view of transition of functional fitness in Figure 4, EACP is superior to other algorithms for all the experiments.

A comparison of variance of the number of active nodes (program size) in the population for each algorithm can be seen in Figure 5. Note that the variance of the number of active nodes with EACP is higher and constant value as against other algorithms. In Figure 5, EACP is realized maintenance of the diversity of program size in the population at all generations. Thus, EACP searches individuals (programs) of various program sizes. In contrast, the individuals converge on a certain program size with other algorithms (SGA, MGG and SPEA2).

Table 2 provides a summary and comparison of the number of successful runs for each algorithm on each of the problem domains tackled. When an individual whose functional fitness is more than 1.0 is found, the run is counted as successful run. MGG and EACP algorithm achieve higher performance than SGA and SPEA2. According to the results, we can consider that localizing the selection pressure to the family is improved the performance of GRAPE. EACP approach which is maintenance of the diversity of program size in the population is significant for improvement of the number of successful runs and functional fitness. In SPEA2, it is difficult to generate the individuals which have large program size.

Figure 6 and 7 show the distribution of EACP and MGG population at the generation 2000, 20000 and 80000 in the experiment of factorial, respectively. The horizontal axis represents the number of active nodes (program size) and the vertical axis is the functional fitness. Each dot in the

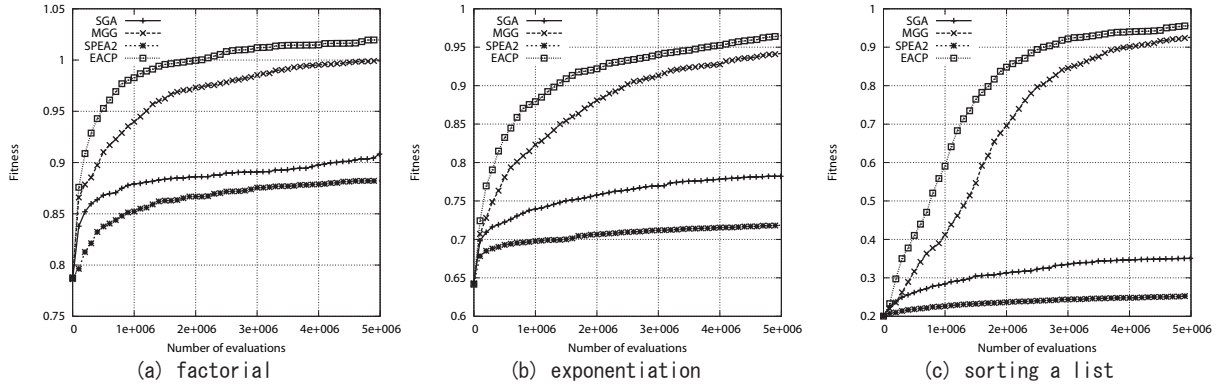

Figure 4: These graphs show the comparison of the transition of average functional fitness for each algorithm over 100 runs.

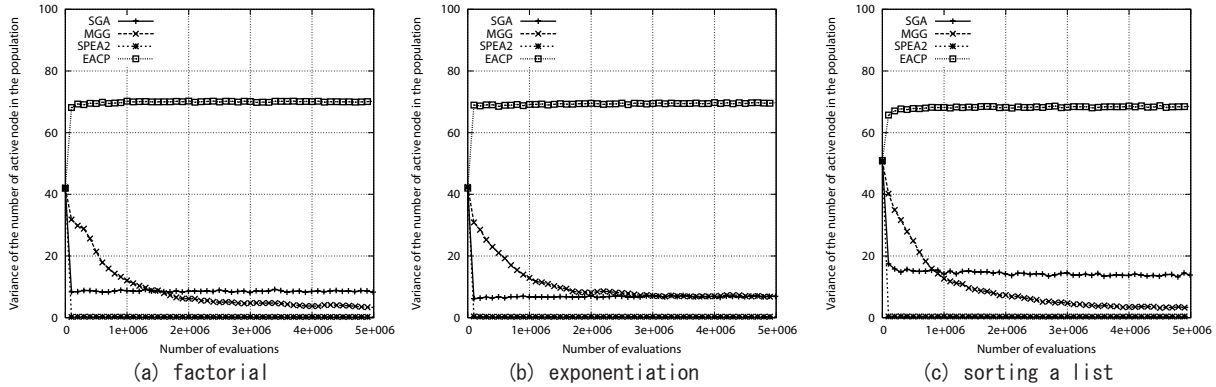

Figure 5: These graphs show the comparison of the variance of the number of active nodes (program size) in the population for each algorithm over 100 runs.

Table 2: The number of successful runs after 5000000 fitness evaluations for each algorithm over 100 runs.

Problem Domain	Number of successful runs			
	SGA	MGG	SPEA2	EACP
Factorial	19	80	7	**95**
Exponentiation	2	**39**	0	37
Sorting a list	1	72	0	**80**

graphs represents one individual. While EACP keeps a variety of the number of active nodes (program size) at all generation, the individuals in MGG has a tendency to converge on a certain program size. Furthermore, large program size individuals (near 30 nodes) cannot be generated by using MGG algorithm.

6. CONCLUSIONS AND FUTURE WORKS

In this paper, we have proposed a new generation alternation model for GRAPE, Evolutionary Algorithm Considering Program Size (EACP). We have applied EACP to three different test problems and examined the performance of EACP. We have demonstrated that EACP improves the performance of GRAPE and maintains the diversity of program size in the population.

As for further research topics, we need to apply EACP to other types of problems to show more effectiveness of EACP. Moreover, we will plan to investigate the performance of EACP using other Automatic Programming techniques, for instance, standard Genetic Programming, Cartesian Genetic Programming and so on.

7. REFERENCES

[1] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler. Multiobjective genetic programming: Reducing bloat using SPEA2. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 536–543, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[2] K. Deb, A. Anand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation*, 10(4):371–395, 2002.

[3] H. Katagiri, K. Hirasawa, J. Hu, and J. Murata. Network structure oriented evolutionary model-genetic network programming-and its comparison with genetic

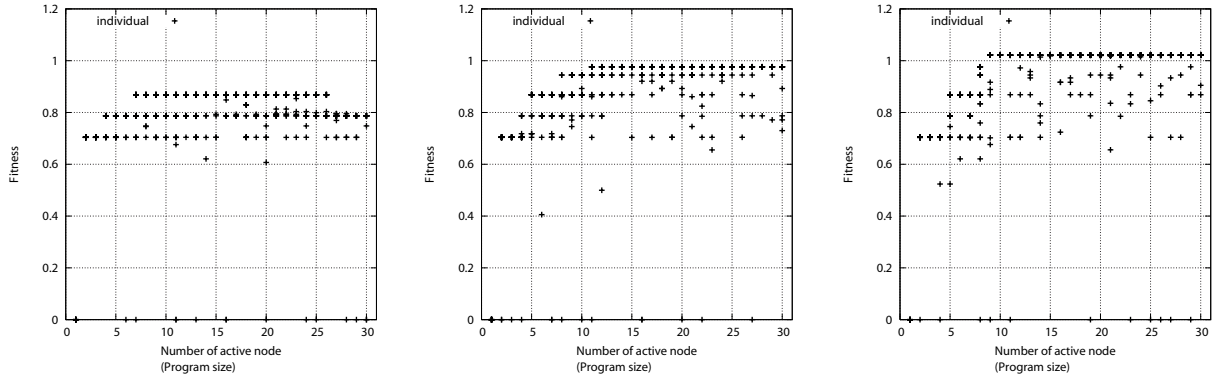

Figure 6: The distributions of EACP population at generation 2000, 20000 and 80000 in the experiment of factorial. Each dot in the graphs represents one individual.

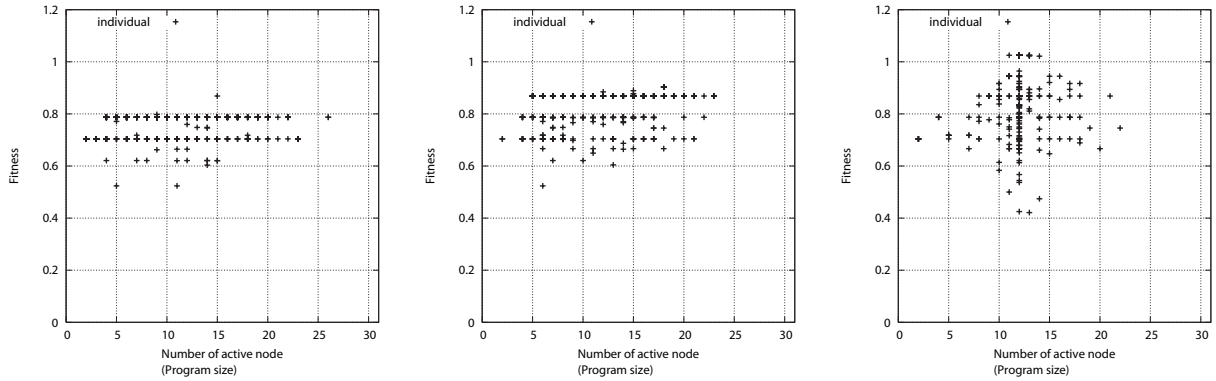

Figure 7: The distributions of MGG population at generation 2000, 20000 and 80000 in the experiment of factorial. Each dot in the graphs represents one individual.

programming. In *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 219–226, San Francisco, California, USA, 9-11 July 2001.

[4] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[5] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 15-16 Apr. 2000. Springer-Verlag.

[6] H. Satoh, M. Yamamura, and S. Kobayashi. Minimal generation gap model for considering both exploration and exploitations. In *Proceedings of the IIZUKA'96*, pages 494–497, 1996.

[7] S. Shirakawa and T. Nagao. Evolution of sorting algorithm using graph structured program evolution. In *Proceedings of the 2007 IEEE International Conference on Systems, Man and Cybernetics (SMC 2007)*, pages 1256–1261, Montreal, Canada, 7-10 Oct. 2007.

[8] S. Shirakawa, S. Ogino, and T. Nagao. Graph Structured Program Evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, volume 2, pages 1686–1693, London, 7-11 July 2007. ACM Press.

[9] T. Soule and J. A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6(4):293–309, 1998.

[10] A. Teller and M. Veloso. PADO: A new learning architecture for object recognition. In K. Ikeuchi and M. Veloso, editors, *Symbolic Visual Learning*, pages 81–116. Oxford University Press, 1996.

[11] B.-T. Zhang and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

[12] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland, 2001.