IEICE TRANS. INF. & SYST., VOL.E92–D, NO.10 OCTOBER 2009

PAPER

Acceleration of Genetic Programming by Hierarchical Structure Learning: A Case Study on Image Recognition Program Synthesis*

Ukrit WATCHAREERUETAI[†a)], ***Nonmember***, **Tetsuya MATSUMOTO**[†b)], **Noboru OHNISHI**[†c)], **Hiroaki KUDO**[†d)], ***and*** **Yoshinori TAKEUCHI**[†e)], ***Members***

SUMMARY We propose a learning strategy for acceleration in learning speed of genetic programming (GP), named hierarchical structure GP (HSGP). The HSGP exploits multiple learning nodes (LNs) which are connected in a hierarchical structure, e.g., a binary tree. Each LN runs conventional evolutionary process to evolve its own population, and sends the evolved population into the connected higher-level LN. The lower-level LN evolves the population with a smaller subset of training data. The higher-level LN then integrates the evolved population from the connected lower-level LNs together, and evolves the integrated population further by using a larger subset of training data. In HSGP, evolutionary processes are sequentially executed from the bottom-level LNs to the top-level LN which evolves with the entire training data. In the experiments, we adopt conventional GPs and the HSGPs to evolve image recognition programs for given training images. The results show that the use of hierarchical structure learning can significantly improve learning speed of GPs. To achieve the same performance, the HSGPs need only 30–40% of the computation cost needed by conventional GPs.

key words: *hierarchical structure genetic programming, acceleration, learning node, training subsets, population integration*

1. Introduction

Fitness evaluation is the most computationally intense process in genetic programming (GP) [21]. Using GP to evolve very complex programs, e.g., object recognition programs, fitness evaluation sometimes spends nearly 100% of total computation time of evolutionary process [10], [29]. Consequently, there are many researches attempt to accelerate the fitness evaluation process of GPs. Some of them try to solve this problem by using a parallel computer architecture or using a special hardware such as GPU [10], [23]. On the other hand, there are many works attempting to accelerate GPs by algorithmic approaches, as discussed below.

One approach is to avoid executions of individuals that have been discovered in the search [11], [29]. In particular, fitness values and canonical forms of the discovered individuals are stored in memory to be retrieved later. If the new individual has the same canonical form as one of those stored in the memory, it means that the individual represents something discovered before. Therefore the GPs need not to re-evaluate it but just fetch its fitness value from the memory instead. Therefore the number of fitness evaluations is reduced, resulting in acceleration of GPs. In addition, to avoid executions of redundant subtrees in tree-based GP could reduce significant amount of computation time [19].

Dynamic subset selection approach [3], [5], [6] was proposed to accelerate in learning speed of GP. In this approach, only a subset of entire training data (fitness cases) is selected and used for learning GP in each generation. This approach may randomly select fitness cases into the subset [5], or select them based on some strategies [3], [5], [6]. Improved methods of this approaches have been proposed in [20].

Gustafon and Hsu studied on the use of layered learning scheme in GP for a robot soccer problem [22]. In particular, they divided GP into two phases (layers); one is to learn robot movement and passing, and the other is to learn team strategy. Although the main goal of this approach is not to reduce the number of fitness evaluations, it can improve GP learning speed. The other approaches that learn GP in multi-phase were proposed in [7], [18].

In [8], Jackson proposed two GP approaches based on division of training data into subsets. One is the function-based architecture that uses the program learned from a subset as a function in tree-based GP. The other is the selection architecture that uses the root node as a switching node. The switching node selects a program, which is learned from a subset, to be executed depending on a test input (in the training set). However, this approach would not function with the problems with unknown test inputs.

Fillon and Bartroli proposed a divide and conquer approach to improve efficiency and probability of success of GP [4]. They divide evolutionary process into two levels. In the first level, multiple subpopulations are used; each one learns with a subset of functions and terminals. Then the second level combines the best individuals found in the first level subpopulations and learns with the original set of functions and terminals.

The method proposed here is different from those explained above. We propose a learning strategy for GP named hierarchical structure GP (HSGP). It is designed for supervised learning with GP. The HSGP solves a problem by using multiple learning nodes (LNs) connected in a hierarchi-

Manuscript received February 2, 2009.
Manuscript revised April 30, 2009.

[†]The authors are with the Department of Media Science, Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8601 Japan.

*This paper was presented at Learning and Intelligent Optimization (LION3) Conference, Trento, Italy, January 14-18, 2009.

a) E-mail: ukrit@ieee.org
b) E-mail: matumoto@is.nagoya-u.ac.jp
c) E-mail: ohnishi@is.nagoya-u.ac.jp
d) E-mail: kudo@is.nagoya-u.ac.jp
e) E-mail: takeuchi@is.nagoya-u.ac.jp

DOI: 10.1587/transinf.E92.D.2094

cal structure. This is similar to the divide and conquer GP in [4]; however, instead of using different subsets of functions and/or terminals, each LN in the HSGP uses different training subset. The lower-level LNs learn with smaller training subsets, whereas the top-level LN learns with the entire training data. From the experimental results, we found that the HSGPs can significantly accelerate in learning speed compared with the conventional GPs (CGPs).

The rest of this paper is organized as follows. Section 2 explains the ideas and the details of the HSGPs. Section 3 briefly describes our target problem, i.e., GP-based synthesis of image recognition programs. Section 4 shows and discusses experimental results of the HSGPs compared with the CGPs, which learn without hierarchical learning structure. Section 5 concludes the paper and gives the ideas of future research.

2. GPs Acceleration by a Hierarchical Structure

2.1 Motivation and Ideas

This work was motivated by the desire to accelerate learning speed of our GP-based image feature synthesis system [29] (more detail will be described in Sect. 3). In this system, an individual represents a feature extraction program which is constructed from basic image processing operations. A problem we found is that the overall evolution time is very long. This is caused by the three following issues:

- By nature of image processing, fitness evaluation of the system spends relatively long computation time, compared with other problems such as symbolic regression problems. The reason is that it has to execute a lot of image processing operation sequences in the fitness evaluation.
- To overcome overfitting problem, we should provide an enough number of representative training data (sometime called samples or fitness cases) to the system. For simple problems, a few training data may be enough to represent the underlying characteristics of the problem. However, for more complex problems, a great number of training data may be unavoidably required. Increasing in the training set size M means increasing in number of program executions too; it increases overall computation time.
- Usually, to achieve good results, many GP-based systems require large population size N and enough number of generations G. These two parameters directly relate to the number of program executions and overall computation time.

However, we found that many programs which are evolved in the beginning of evolutionary process are frequently non-reasonable and give very bad fitness. Consequently, it would be wasteful to evaluate those programs by using the entire training set. Instead, we may estimate their fitness values from only a portion of training set and decide based upon the estimated fitness whether they should survive in the next generation. This reduces the number of program executions. However, the entire training set should be provided in the end of evolutionary process to make the GPs generate more reliable, non-over-fitting programs. Based upon these considerations, we propose a learning strategy for GPs that learns in such manner, and we investigate its effect on acceleration in learning speed.

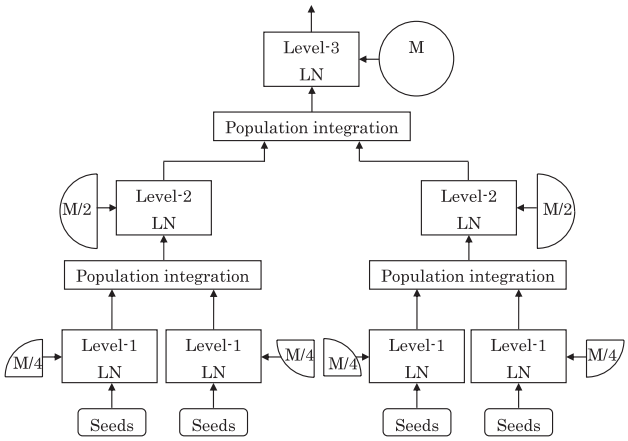


Fig. 1 The HSGP using a binary-tree structure with three levels. The top-level LN uses the entire training set with M fitness cases (illustrated by circle) whereas the middle-level and bottom-level LNs use only portions of the entire training set (the half-circles and 1/4 circles, respectively).

2.2 Hierarchical Structure Genetic Programming

We propose an approach named HSGP that evolves multiple LNs connected in a hierarchical structure. The lower-level LNs learn with smaller subsets of training data and send the evolved population into the next higher-level LN to evolve further with larger subsets. The top-level LN learns with the entire training set. The HSGPs studied in this work use a binary-tree structure as shown in Fig. 1.

In the figure, the binary-tree structure consists of three levels (level 1, 2, and 3). The L^{th} level contains $S_L = 2^{L_{max}-L}$ LNs, and learns with a training subset of size $M_L = M/2^{L_{max}-L}$, where L_{max} is the highest level and M is the size of the original training set. The HSGP starts evolutionary process from the bottom-level LNs. Each LN of the bottom-level generates its own initial population, and executes conventional evolutionary process to evolve the population for a fixed number of generations G_L which can be shorter than the number of generations G of the CGPs. Each LN in a level learns with different and non-overlapped subset with each other. Then each LN of the next higher-level receives the evolved population from the connected lower-level LNs and integrates them into a new population. The details of various integration methods will be described in Sect. 2.3. The integrated population will be used as the initial population of the higher-level LN. Also the training subsets of the connected lower-level LNs are integrated (union) together and used as the training subset of the higher-level LN. Each

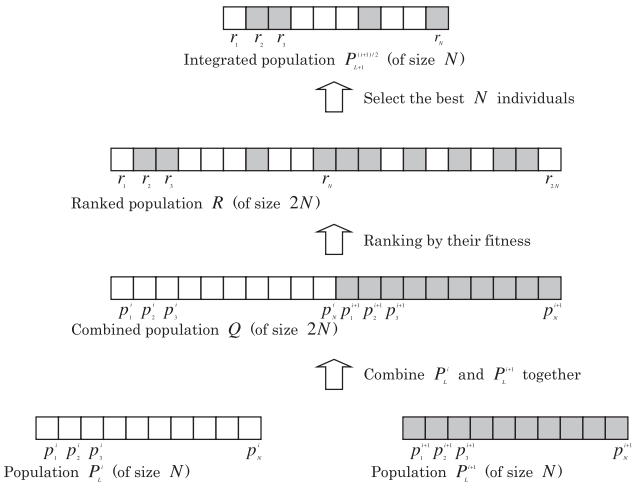


Fig. 2 The ranked-whole method.

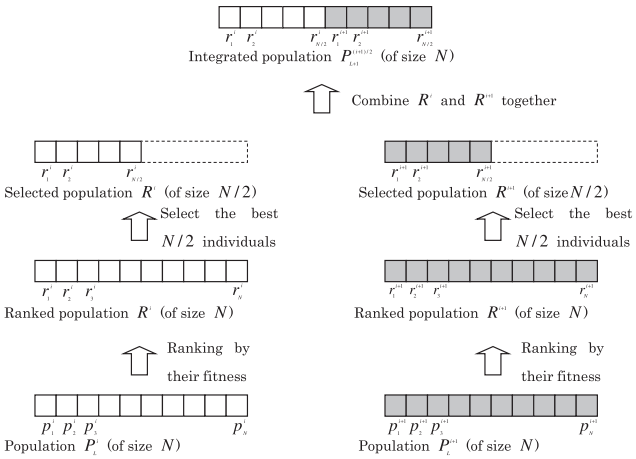


Fig. 3 The ranked-half-half method.

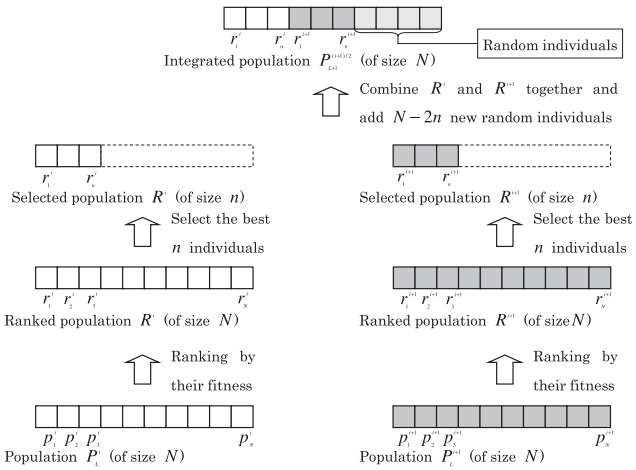


Fig. 4 The n-elitist method.

level of LNs is executed level-by-level and the subsets used become larger. After the learning at the top-level LN (with the entire training set) finished, the best individual obtained by the top-level LN is considered as the best solution for the problem.

One advantage of learning in this hierarchical manner is that we can use more initial individuals ($2^{L_{max}-1} \times N$), compared with CGPs. This provides more genetic material to be evolved to GPs and gives more diversities than those of CGPs. In Sect. 4.2, we will show that the use of different seeds in each bottom-level LN can improve performance of HSGPs.

From this learning strategy, the evolutionary process is done in hierarchical manner in which the problem difficulty increases with the level index. The goal of the bottom-level LNs is just to identify the potential areas of the solutions (not to find the global optimum), whereas that of the middle-level LNs is to confirm whether the potential areas learned by the lower-level are really global ones. Finally, the top-level LN exploits the complete training set and focus on the identified potential area to find the global optima in the search space.

2.3 Population Integration

There are a number of possible ways to integrate two population learned by the lower-level LNs into one population for the next higher-level LN. In this paper, three population integration methods are investigated. The first is named ranked-whole method. This method firstly combines the two populations (each contains N solutions) into a bigger population (of size $2N$), and then chooses only the best N solutions from the whole population and deletes the remaining N solutions out of the population (Fig. 2). The second is named ranked-half-half method. Contrasting with the ranked-whole method, the ranked-half-half method firstly selects the best $N/2$ solutions from each population, and then integrates those selected individuals together (Fig. 3). The third is named n-elitist method. It selects the best n solutions ($n < N/2$) from each population and randomly generates $N - 2n$ new solutions into the integrated population (Fig. 4). The comparison of these population integration methods will be discussed in Sect. 4.3.

2.4 Computation Cost

In the case of the CGP, the number of program execution of an independent GP run is directly related with population size N, training set size M, and the number of generations G. In particular, the number of fitness evaluations of an independent run is equal to $N \times G$. For one time of fitness evaluation, it has to execute genetic programs M times because there are M training data in the training set. Therefore the total number of program executions is equal to

$$N \times G \times M. \tag{1}$$

In the case of the HSGP, a LN of the L^{th} level executes programs $N \times G_L \times M_L$ times, where G_L is the number of generations at level L. Therefore the total number of program executions (NPEs) of the HSGP is equal to

$$\sum_{L=1}^{L_{max}} S_L \times N \times G_L \times M_L = \sum_{L=1}^{L_{max}} N \times G_L \times M. \quad (2)$$

In all experiments in this paper, we use the same number of generations G_L for all levels, i.e., we set $G_L = g$. In this case, the number of program execution in Eq. 2 can be reduce to

$$L_{max} \times N \times g \times M. \quad (3)$$

Comparing Eq. 3 with Eq. 1, the common variables are N and M. Therefore the variables that have to be controlled to gain advantage in program execution reduction are only L_{max} and g. In the experiments, we set $L_{max} = 3$ and adjust g to compare the performance of the HSGPs with CGPs.

For the sake of clarity, the following example compares computation cost of the CGP and HSGP. Let $N = 50$, $M = 20$, and $G = 300$. According to Eq. 1, computation cost of CGP is 300,000 NPEs. Letting $L_{max} = 3$ (the structure in Fig. 1), the HSGP needs $3 \times 50 \times 20\, g = 3000\, g$, according to Eq. 3. If we set $g = 100$, the HSGP will need the same NPEs as the above CGP (300,000). If $g = 50$, only half number of the NPEs (150,000) is needed.

3. GP-Based Image Recognition Program Synthesis

Evolutionary algorithms (EAs) have been widely used in many image recognition problems. However, in this paper, we are interested in the EA-based approaches that synthesize image recognition programs (or a part of them such as feature extraction programs) for a given problem. Up to now, many approaches have been studied [1], [2], [9], [10], [12]–[14], [16]–[19], [24]–[26], [28], [29]. The approach considered here is based on GP with linear representation (linear GP), which encodes a program as a sequence of primitive operations (POs) [15]. Figure 5 shows the overview of evolutionary system for synthesis of image recognition programs. In this case, inputs which are needed from user are just image processing library, training images, and objective function. Image processing library consists of basic image processing and pattern recognition operations, e.g., edge detection, lowpass filtering, image thresholding (see Appendix). These operations are used as POs to construct feature extraction programs. The system randomly generates a population of individuals, which encode feature extraction programs. These individuals are decoded and evaluated by using the given training images. In the evaluation process, the defined objective function is used to compute fitness value. The individuals with the higher fitness values will have higher chance to survive and be evolved. After evolution process finished, the program encoded by the individual that give the best fitness value is considered the output of the system.

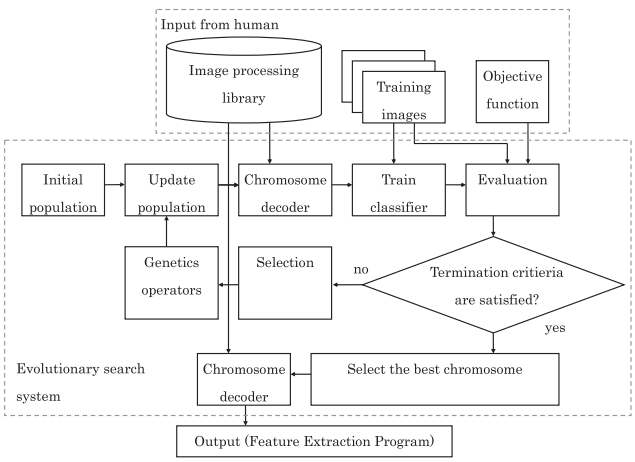


Fig. 5 Overview of GP-based image feature extraction program synthesis.

In this paper, an image recognition program is represented as a sequence of POs encoded in a fixed-length string. In execution process, each operation is sequentially interpreted and executed one by one, until all of the operations in the sequence are finished. Similar to other linear GPs, the executions are based on a set of shared registers. In particular, an operation fetches inputs from registers, processes them, and stores its output into a register. After that, we go to the next operation and repeat this process until all operations are finished. Each operation is coded by four parts, i.e., one opcode (operation code) and three arguments. The opcode tells us what operation will be executed, whereas the arguments define related input registers and output register. Two types of shared registers, i.e., image and numerical registers, are used. Image registers store input images and processed images, whereas numerical registers store real-value constants and numerical outputs. Before program execution start, all image registers are initialized by an input images. After program execution finished, the image stored in a predefined image register is used as output of program.

The test problem is the lawn weed detection problem [27]. The goal of GPs is to evolve a program that can accurately segment the area of weeds from lawn backgrounds (Fig. 6). The dataset we used contains 30 images; 20 images are used as a training set and the remaining 10 images as a validation set. The image size is smaller than the original ones used in [27]; its size is 120×160 pixels.

To calculate fitness value of an individual, leave-one-out cross validation is adopted. In particular, m times of validation are done (m is the number of images). In each time of validation, one image is used for validation and the remaining $m - 1$ images for training, and segmentation accuracy of the validation image is computed. The average of segmentation accuracies is used as the fitness value. It is

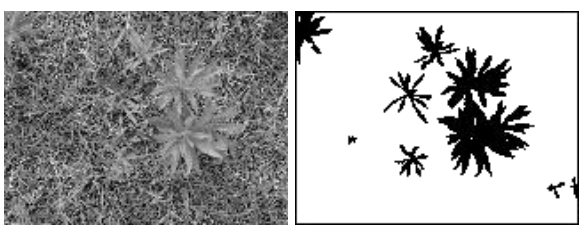


Fig. 6 An example of training image and its corresponding ground truth.

Table 1 Segmentation accuracy (%) of the programs evolved by the CGPs and the HSGP with the same seeds, fixed parameters, and the ranked-whole method ($g = 50$ and $g = 100$). p_c, p_m, and t_s denote crossover rate, mutation rate, and tournament size, respectively. NPE denotes the number of program executions.

Method	p_c	p_m	t_s	NPE	Training		Validation	
					Mean	STD	Mean	STD
CGP	0.75	0.1	2	300,000	97.157	0.334	95.698	0.571
CGP	0.75	0.1	5	300,000	97.126	0.369	95.838	0.490
CGP	0.75	0.2	5	300,000	97.072	0.299	95.729	0.466
CGP	0.50	0.1	5	300,000	97.441	0.236	96.143	0.433
HSGP ($g = 50$)	0.50	0.1	5	150,000	97.322	0.300	95.979	0.342
HSGP ($g = 100$)	0.50	0.1	5	300,000	97.566	0.230	96.300	0.390

defined as

$$\frac{1}{m \cdot H \cdot W} \sum_{i=1}^{m} \sum_{x=0}^{H-1} \sum_{y=0}^{W-1} \left(1 - \frac{|O_i(x,y) - GT_i(x,y)|}{255}\right),$$

where $O_i(x, y)$ is the segmentation result of the i^{th} image, whereas $GT_i(x, y)$ is the corresponding ground truth, H and W are image height and width, respectively.

4. Experiments and Discussions

We have conducted four experiments to evaluate the performance of the HSGPs compared with the CGPs. For all experiments, the population size of the CGPs and each level LNs of the HSGP is 50. In the case of HSGPs, the number of levels of the binary-tree structure is three ($L_{max} = 3$), as shown in Fig. 1. The training set was divided into four non-overlapped subsets for the bottom-level LNs. In the training set, four images are lawn images that contain no weeds. We assigned one image that contains no weeds to each subset. The remaining images were assigned to the subsets depending on their file name (no other special selection). As mentioned before, we set the numbers of generation $G_1 = G_2 = G_3 = g$ for all experiments. This value g was varied to compare the both GP approaches. We used parameterized uniform crossover with genes exchange probability of 0.2, and the mutation operator that randomly exchanges op-code or arguments of mutated operations. The elitist model, i.e., to preserve a copy of the best individual into the next generation, was adopted. The number of POs in library is 51 (see Appendix). String length is 10 (operations). The number of image and numerical registers are eight. For each GP approach, 20 independent runs were experimented.

4.1 Experiment 1: Comparison of GPs with and without Hierarchical Structure Learning

The goal of this experiment is to investigate whether *only* the use of the hierarchical structure learning can improve performance of GPs. To do that, the use of different seeds in the bottom-level LNs, which is an advantage of the HSGPs, was off, and the ranked-whole population integration method was used so that no new random individuals were introduced into the population during runs. All levels LNs used the same set of parameters to investigate only the effect of hierarchical learning structure.

Firstly, we experimented with the CGPs with four different sets of parameters as shown in Table 1. The maximum number of generations G of the CGPs is 300. From the table, we found that among the four parameter sets, the CGP with crossover rate $p_c = 0.5$, mutation rate $p_m = 0.1$, and tournament size $t_s = 5$ gives the best results. We then experimented with the HSGP with the same set of parameters. Each level learns with 50 generations ($g = 50$). According to Eq. (3), the HSGP totally needs program executions of 1.5×10^5 times, whereas the CGP tested in this experiment needs 3×10^5 times of program executions (according to Eq. (1)). That means the HSGP used only the half number of program executions of the CGPs. Its result is shown at the second bottom row of Table 1. We found that by using only half number of program executions, the HSGP outperforms three CGPs and give slightly lower performance than the CGP with the best parameter setting. We then experimented with HSGP with $g = 100$ (i.e., it used the same NPEs as the CGPS). In this case, the HSGP provides better results than all of the CGPs (the last row in Table 1). This shows the advantage of using the binary-tree learning structure over the CGPs.

4.2 Experiment 2: Effect of Population Initialization

Here we investigated the effect of population initialization of the HSGPs. The use of LNs connected in a binary-tree structure allows the HSGPs to have more initialized seeds in the bottom-level LNs, i.e., each one has its own initialized seeds. The HSGPs that use different seeds in the bottom-level LNs maintain more diversity than the HSGPs with the same seeds in all bottom-level LNs and the CGPs. Table 2 shows the comparison between the HSGPs using the same seeds and different seeds. We varied g of the HSGPs from 25, 30, 40, and 50, as shown. We found some improvements in performance of the HSGPs. The HSGPs that use different seeds seem to be better than the HSGPs with the same seeds (except for $g = 30$) and better than all of the CGPs tested in the previous experiment (HSGPs with $g = 40$ and $g = 50$). This is because it allows each of bottom-level LN to focus on its own distinct area, whereas for the HSGPs with the same seeds, the bottom-level LN is more likely to focus on

Table 2 Segmentation accuracy (%) of the programs evolved by the HSGPs with same/different seeds, fixed parameters, and the ranked-whole method. NPE denotes the number of program executions.

Generations G_1-G_2-G_3	NPE	Same seeds				Different seeds			
		Training		Validation		Training		Validation	
		Mean	STD	Mean	STD	Mean	STD	Mean	STD
25-25-25	75,000	97.118	0.377	95.703	0.556	97.191	0.274	95.768	0.506
30-30-30	90,000	97.344	0.283	96.041	0.413	97.261	0.292	96.006	0.380
40-40-40	120,000	97.345	0.249	96.048	0.311	97.540	0.163	96.321	0.204
50-50-50	150,000	97.322	0.300	95.979	0.454	97.470	0.299	96.187	0.445

Table 3 Segmentation accuracy (%) of the programs evolved by the HSGPs with the n-elitist and ranked-half-half methods (using different seeds). NPE denotes the number of program executions.

Generations G_1-G_2-G_3	NPE	n-elitist				Ranked-half-half			
		Training		Validation		Training		Validation	
		Mean	STD	Mean	STD	Mean	STD	Mean	STD
25-25-25	75,000	97.327	0.239	96.046	0.318	97.261	0.400	95.896	0.559
30-30-30	90,000	97.413	0.168	96.136	0.238	97.382	0.231	96.069	0.399
40-40-40	120,000	97.400	0.242	96.128	0.318	97.284	0.254	95.978	0.413
50-50-50	150,000	97.500	0.239	96.263	0.335	97.359	0.244	96.023	0.391

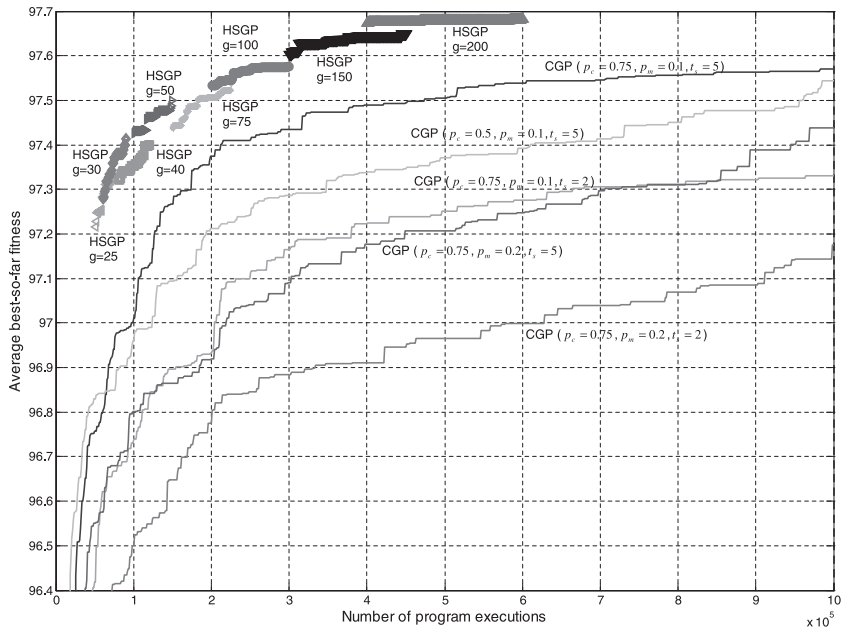


Fig. 7 Comparison of average best-so-far fitness of the HSGPs (of the top-level LN only) and CGPs.

the same area with each other.

4.3 Experiment 3: Comparison of Population Integration Methods

In this experiment, we compare the effects of population integration methods described in Sect. 2.3. The HSGPs using different seeds and fixed parameters in all levels were experimented by changing population integration methods. Table 3 shows the performance comparison of the three integration methods.

From the results in Tables 2 and 3, the n-elitist method ($n = 5$ in this work) seems to give the best performance compared with those of the ranked-half-half and ranked-whole methods (except only the case of $g = 40$ that the ranked-whole method outperforms the others). The reason may be that the HSGP with the n-elitist method introduces random individuals into populations when it combines LNs of the lower-level together. Introduction of new random individuals increases diversity of the population and would prevent the HSGPs from getting stuck at a local optimum.

For the ranked-whole and ranked-half-half methods, they do not introduce more diversity into the integrated population but rather select the best individuals from both LNs. Consequently, if the best individuals from the both LNs are of local optima, it seems that the HSGPs with these integration methods can rarely escape from those local optima, resulting in bad results. However, we found that the ranked-

Table 4 Comparison of the program executions needed by CGPs and the HSGPs to produce comparable performances.

Segmentation accuracy (%)	CGP	HSGP	Ratio (HSGP/CGP)
97.20	1.27×10^5	$< 0.5 \times 10^5$	< 0.3937
97.30	1.73×10^5	0.63×10^5	0.3642
97.40	2.11×10^5	0.89×10^5	0.4218
97.50	4.73×10^5	1.5×10^5	0.3171
97.60	$> 10 \times 10^5$	$< 3 \times 10^5$	< 0.3000

whole method sometimes gives the best individuals with very high fitness (many of them are better than those of the n-elitist method). The reason would be that the HSGP with the ranked-whole method can exploit the best individuals faster than the HSGP with the n-elitist method; if the best individuals found by the lower-level LNs are near the global optimum, the ranked-whole method is more likely to find the global optimum.

4.4 Experiment 4: Reduction in the Number of Program Executions

In the previous experiments, we have investigated some characteristics of the HSGPs. Here we compare the HSGPs with the best setting, i.e., using different seeds and the n-elitist integration method, with the CGPs with the maximum number of generations of 1000. The number of generations g of the HSGPs is also varied from 25 to 200. Figure 7 shows the average best-so-far fitness comparison of the CGPs and the HSGPs (only the best-so-far fitness of the top-level LN is shown). The figure indicates that the HSGPs outperforms the CGPs for all values of g. Also we compare the numbers of program executions needed by both CGPs and HSGPs to give the comparable performance (Table 4). We found that the HSGPs need only 30–40% of the number of program executions needed by the CGP with the best setting.

5. Conclusion and Future Works

We have investigated an idea of using the hierarchical structure learning to accelerate GPs, named HSGP. Multi LNs are connected in a binary-tree whose lower-level LNs learn with smaller training subsets. A higher-level LN integrates the connected lower-level LNs together and learns with integrated training subsets. Experimental results demonstrate the success of the HSGPs in reduction of the number of program executions; around 60–70% of program executions can be reduced without degradation of performances. From the experiments, we found that major source of success of the HSGP are as follows:

- To learn from training subsets in the lower-level LNs. Using only training subsets to evaluate the programs in the beginning of evolutionary process does not affect performance of GPs but can significantly reduce the number of program executions.
- More diversities which are introduced by three issues. The first is the use of larger initial populations (different seeds). The second is the introduction of new random individuals by the n-elitist population integration. And the third is the use of hierarchical structure in which the populations in the same level are evolved separately and independently.

Although we have experimented with a GP-based image recognition program synthesis, the proposed hierarchical structure learning is general; it can be used with the other problems and with the other EAs techniques such as GAs. In the future, we plan to compare the HSGPs with the other approaches and extend the experiments with the other test problems. In addition, the following questions should be studied further.

- What hierarchical structure is the best?
- How do we select the number of levels of the structure?
- How do we select the appropriate parameters (e.g., population size, maximum generation, p_c, p_m) in each level?

These parameters would affect the performance of the HSGPs.

Acknowledgments

Authors thank the Hori Information Science Promotion Foundation for a research grant.

References

[1] A. Teller and M. Veloso, "A controlled experiment: Evolution for learning difficult image classification," Proc. 7th Portuguese Conf. Artificial Intelligence, LNCS 990, pp.165–185, Berlin, Germany, 1995.
[2] A. Teller and M. Veloso, "PADO: A new learning architecture for object recognition," in Symbolic Visual Learning, ed. K. Ikeuchi and M. Veloso, pp.77–112, Oxford Univ. Press, Oxford, UK, 1997.
[3] B.-T. Zhang and D.-Y. Cho, "Genetic programming with active data selection," in Simulated Evolution and Learning, LNCS-1585, ed. B. Mckay, X. Yao, C.S. Newton, J-H. Kim, and T. Furuhashi, pp.146–153, Springer-Verlag, Berlin Heidelberg, 1999.
[4] C. Fillon and A. Bartoli, "A divide & conquer strategy for improving efficiency and probability of success in genetic programming," Proc. EuroGP 2006, LNCS-3905, pp.13–23, Springer-Verlag, Berlin Heidelberg, 2006.
[5] C. Gathercole and P. Ross, "Dynamic training subset selection for supervised learning in genetic programming," in Parallel Problem Solving from Nature III, LNCS-866, ed. Y. Davidor, H.-P. Schwefel, R. Männer, pp.313–321, Springer-Verlag, Berlin Heidelberg, 1994.
[6] C. Lasarczyk, P. Dittrich, and W. Banzhaf, "Dynamic subset selection based on a fitness case topology," Evolutionary Computation, vol.12, no.2, pp.223–242, 2004.
[7] D. Howard and S.C. Roberts, "A staged genetic programming strategy for image analysis," Proc. GECCO-99, vol.2, pp.1047–1052, Florida, USA, 1999.
[8] D. Jackson, "Hierarchical genetic programming based on test input subsets," Proc. GECCO-2007, pp.1612–1619, London, England, 2007.
[9] G. Olague, E. Romero, L. Trujillo, and B. Bhanu, "Multiclass object recognition based on texture linear genetic programming," in Application of Evolutionary Computing, LNCS 4448, ed. M. Giacobini,

A. Brabazon, S. Cagnoni, G.A. Di Caro, R. Drechsler, M. Farooq, A. Fink, E. Lutton, P. Machado, S. Minner, M. O'Neill, J. Romero, F. Rothlauf, G. Squillero, H. Takagi, A.S. Uyar, and S. Yang, pp.291–300, Springer-Verlag, Berlin, 2007.

[10] J. Ando and T. Nagao, "Fast tree-structural image processing using GPU," Proc. IWAIT-2007, pp.423–428, Bangkok, Thailand, 2007.

[11] J. Niehaus, C. Igel, and W. Banzhaf, "Reducing the number of fitness evaluations in graph genetic programming using a canonical graph indexed database," Evolutionary Computation, vol.15, no.2, pp.199–221, 2007.

[12] K. Krawiec, "Generative learning of visual concepts using multiobjective genetic programming," Pattern Recognit. Lett., vol.28, no.16, pp.2385–2400, 2007.

[13] K. Krawiec and B. Bhanu, "Visual learning by coevolutionary feature synthesis," IEEE Trans. Syst. Man Cybern., B, Cybern., vol.35, no.3, pp.409–425, 2005.

[14] K. Krawiec and B. Bhanu, "Visual learning by evolutionary and coevolutionary feature synthesis," IEEE Trans. Evol. Comput., vol.11, no.5, pp.635–650, 2007.

[15] M. Brameier and W. Banzhaf, Linear Genetic Programming, Springer, 2007.

[16] M. Roberts and E. Claridge, "Co-operative coevolution of image feature construction and object detection," Parallel Problem Solving from Nature - PPSN VIII, LNCS 3242, pp.899–908, Springer, Birmingham, UK, 2004.

[17] M. Roberts and E. Claridge, "A multi-stage approach to cooperatively coevolving feature construction and object detection," in Application of Evolutionary Computing, LNCS 3449, ed. F. Rothlauf, J. Branke, S. Cagnoni, D.W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G.D. Smith, and G. Squillero, pp.396–406, Springer-Verlag, Berlin, 2005.

[18] M. Zhang, U. Bhowan, and B. Ny, "Genetic programming for object detection: A two-phase approach with an improved fitness function," Electronic Letters on Computer Vision and Image Analysis, vol.6, no.1, pp.27–43, 2007.

[19] P. Wong, and M. Zhang, "SCHEME: Caching subtrees in genetic programming," Proc. IEEE CEC-2008, pp.2683–2690, Hong Kong, China, 2008.

[20] R. Curry, P. Lichodzijewski, and M.I. Heywood, "Scaling genetic programming to large datasets using hierarchical dynamic subset learning," IEEE Trans. Syst. Man Cybern., B, Cybern., vol.37, no.4, pp.1065–1073, 2007.

[21] R. Poli, W.B. Langdon, and N.F. McPhee, A Filed Guide to Genetic Programming, Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008.

[22] S.M. Gustafon and W.H. Hsu, "Layered learning in genetic programming for a cooperative robot soccer problem," Proc. EuroGP 2001, LNCS-2038, pp.291–301, Springer-Verlag, Berlin Heidelberg, 2001.

[23] S. Harding, and W. Banzhaf, "Fast genetic programming on GPUs," Proc. EuroGP 2007, LNCS 4445, pp.90–101, Springer-Verlag, Berlin Heidelberg, 2007.

[24] S. Shirakawa and T. Nagao, "Genetic image network (GIN): Automatically construction of image processing," Proc. IWAIT-2007, pp.643–648, Bangkok, Thailand, 2007.

[25] S. Aoki and T. Nagao, "Automatic construction of tree-structural image transformations using genetic programming," Proc. ICAIP-99, pp.136–141, Venezia, Italy, 1999.

[26] T. Nagao and S. Masunaga, "Automatic construction of image transformation processes using genetic algorithm," Proc. ICIP-96, vol.3, pp.731–734, Lausanne, Switzerland, 1996.

[27] U. Watchareeruetai, Y. Takeuchi, T. Matsumoto, H. Kudo, and N. Ohnishi, "Computer vision based methods for detecting weeds in lawns," Machine Vision and Applications, vol.17, no.5, pp.287–296, 2006.

[28] U. Watchareeruetai, Y. Takeuchi, T. Matsumoto, H. Kudo, and N. Ohnishi, "Improving search performance of linear genetic programming based image recognition program synthesis by redundancy-removed recombination," Proc. IEEE SMCia/08, pp.393–398, Muroran, Japan, 2008.

[29] U. Watchareeruetai, Y. Takeuchi, T. Matsumoto, H. Kudo, and N. Ohnishi, "Transformation of redundant representations of linear genetic programming into canonical forms for efficient extraction of image features," Proc. IEEE CEC-2008, pp.1996–2003, Hong Kong, China, 2008.

Appendix: Primitive Operations

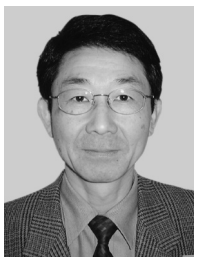
Table A· 1 shows the list of primitive operations we used.

Table A· 1 Primitive operators used in this work.

One-input operations	Two-input operations
image → image	**image, image → image**
highpass filter	image addition
Sobel operation	image subtraction
image negative	**image, real value → image**
mean thresholding	lowpass filter
entropy thresholding	median filter
histogram equalization	morphological dilation
image → real value	morphological erosion
global mean	morphological opening
global variance	morphological closing
global STD	local histogram equalization
global skewness	thresholding
global kurtosis	local variance
global maximum	local skewness
global minimum	local kurtosis
global median	local maximum (max filter)
global mode	local minimum (min filter)
global range	local mode
global entropy	local range
	local entropy

Ukrit Watchareeruetai received the B. Eng in Electrical Engineering from Kasetsart University (Thailand) in 2002. He worked as a research assistant at Kasetsart Signal and Image Processing Laboratory (KSIP LAB), Kasetsart University, from 2002 to 2004. He received the Master degree of Information Science from Graduate School of Information Science, Nagoya University (Japan) in 2007. He is currently a doctoral student at the same school. His research areas include computer vision, pattern recognition and evolutionary computation. He is a student member of IEEE, ACM and IPSJ.

Tetsuya Matsumoto received the B.E., M.E., and Dr. Eng. degrees from Nagoya University, Nagoya, Japan, in 1982, 1984 and 1996, respectively. From 1984 to 1989, he worked in Toshiba Corporation, Fuchu, Japan, where he engaged in research and development of the control system of nuclear power plant. In 1993, he joined Education Center for Information Processing, Nagoya University. In 1998, he moved to Department of Information Engineering, Graduate School of Engineering, Nagoya University. His current interests include neural networks, image processing and machine learning. He is a member of JNNS of Japan and JSAI of Japan.



Noboru Ohnishi received the B. Eng., M. Eng. and D. Eng. degrees from Nagoya University, Nagoya, Japan, in 1973, 1975 and 1984, respectively. From 1975 to 1986 he was with the Rehabilitation Engineering Center under the Ministry of Labor. From 1986 to 1989 he was an Assistant Professor in the Department of Electrical Engineering, Nagoya University. From 1989 to 1994, he was an Associate Professor. Since 1994, he is a professor in Nagoya University. From 1993 to 2001, he concurrently held a Head of Laboratory for Bio-mimetic Sensory System at the Bio-mimetic Control Research Center of RIKEN. He is now in the Graduate School of Information Science. His research interests include computer-vision and -audition, robotics, bio-cybernetics, and rehabilitation engineering. Dr. Ohnishi is a member of IEEE, IEEJ, IPSJ, SICE, JNNS, IIITE and RSJ.

Hiroaki Kudo received the degrees of B. Eng., M. Eng. and Dr. Eng. at Nagoya University, Japan in 1991, 1993 and 1996, respectively. In April 1996, he was a faculty member of the School of Engineering, Nagoya University as a Research Associate. In April 1997, he was a faculty member of the Graduate School of Engineering, Nagoya University. In April 1999, he was an Assistant Professor. In August 2000, he was an Associate Professor at Center for Information Media Studies, Nagoya University. Since 2003, he has been a member of Graduate School of Information Science, Nagoya University. He was a Research Fellow of the Japan Society for the Promotion of Science in 1995. His research interests include visual perception and computer vision. He is a member of IEEE, ITE, and IEEJ.

Yoshinori Takeuchi received the degrees of B. Eng., M. Eng. and Dr. Eng. from Nagoya University in 1994, 1996 and 1999, respectively. In 1999, he was a Research Fellow of the Japan Society for the Promotion of Science. In 2000, he was a member of the Graduate School of Engineering, Nagoya University. Currently, he is an Associate Professor at the Information Security Promotion Agency, Nagoya University. His research interests include computer vision and computer audition. He is a member of IEEE and RSJ.