Evolution of Search Algorithms Using Graph Structured Program Evolution

Shinichi Shirakawa and Tomoharu Nagao

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, nagao@ynu.ac.jp

Abstract. Numerous evolutionary computation (EC) techniques and related improvements showing effectiveness in various problem domains have been proposed in recent studies. However, it is difficult to design effective search algorithms for given target problems. It is therefore essential to construct effective search algorithms automatically. In this paper, we propose a method for evolving search algorithms using Graph Structured Program Evolution (GRAPE), which has a graph structure and is one of the automatic programming techniques developed recently. We apply the proposed method to construct search algorithms for benchmark function optimization and template matching problems. Numerical experiments show that the constructed search algorithms are effective for utilized search spaces and also for several other search spaces.

1 Introduction

Automatic programming, which generates computer programs automatically, has been actively investigated in recent studies. Genetic programming (GP) [1] is a typical example of automatic programming, which was originally introduced by Koza. GP evolves computer programs, which usually have a tree structure, and searches for a desired program using a genetic algorithm (GA). Various representations for GP have been proposed thus far, including graph representation. The advantages of a graph structure are that it can represent complex structures compared with tree structures, and that it can contain time-series information in its graph structure. Graph Structured Program Evolution (GRAPE) [2, 3] is one of a number of automatic programming techniques developed recently. The representation of GRAPE is a graph structure, therefore, it can represent complex programs (e.g., branches and loops) using this structure. The genotype of GRAPE is in the form of a linear string of integers. GRAPE succeeds in generating complex programs automatically (e.g., factorial, exponentiation, list sorting, etc.). The study described in this paper is based on this GRAPE technique.

Numerous evolutionary computation (EC) techniques and related improvements, showing their effectiveness in various problem domains such as function optimization and combination problems, have been proposed. In addition, no free lunch (NFL) theorems [4] show that all black-box search algorithms have the same average performance over the entire set of optimization problems. Therefore, it is essential to construct a search algorithm that is efficient in its approach to certain problems. In general, it is difficult to instantly construct efficient search algorithms for the given problems. Moreover, the automatic construction of search algorithms requires the use of key techniques. In this context, Meta GA, which optimizes the parameters of GA using GA, was proposed. Parameters such as population size, crossover rate, and mutation rate can be optimized in Meta GA. In addition, several techniques for the evolution of evolutionary algorithms have been investigated [5–8]. A method for evolving evolutionary algorithms, which evolves generation alternation procedures, was proposed by Oltean [6]. Dioşan and Oltean proposed a method for evolving crossover operators for evolutionary function optimization using GP [7] and a method for evolving the structure of a particle swarm optimization (PSO) algorithm [8].

Recently, the researches on hyper-heuristics, which are the methods to combine heuristics to generate new ones, have attracted increasing attentions. Several literatures on hyper-heuristics using GP are existed. Burke et al. generated a heuristic for the bin-packing problem using GP [9]. Kumar et al. showed that GP can evolve a set of heuristics for biobjective 0/1 knapsack problem [10]. Pillay and Banzhaf generated a heuristic for the timetabling problem [11]. The motivation of this work is the same as hyper-heuristics.

Our aim is to construct more efficient search strategies automatically so that they resemble human-designed search algorithms. In this study, we generate search algorithms by evolving the programs of moving agents in the search spaces using GRAPE. In particular, we focus on function optimization problems. A search algorithm is represented by a graph structure. The parameters are updated at each node. We apply the proposed method to benchmark function optimization problems and template matching problems. After evolving the search algorithms, we apply them to test problems that were not used during their evolution in order to verify their efficiency and versatility.

An overview of GRAPE is presented in the next section. In Section 3, we describe our proposed method, the evolution of search algorithms using GRAPE. Next, in Section 4, we apply the proposed method to the problem of the automatic construction of search algorithms and show several experimental results. Finally, in Section 5, we present our conclusions and possible future work.

2 Graph Structured Program Evolution (GRAPE)

GRAPE [2, 3] constructs graph-structured programs automatically. The features of GRAPE are summarized as follows:

– Arbitrary directed graph structures.
– Ability to handle multiple data types using a data set.
– Genotype of an integer string.

A graph-structured program is composed of an arbitrary directed graph of nodes and a data set. The data set through flows the directed graph and is processed at

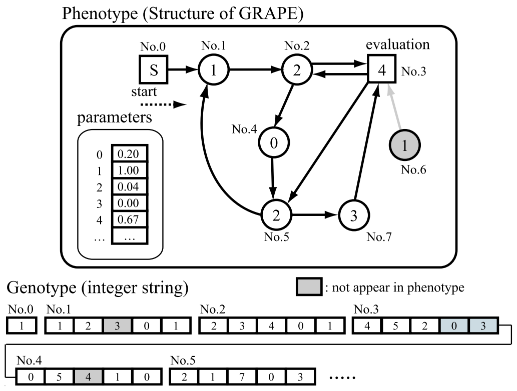


Fig. 1. Structure of GRAPE (phenotype) and the genotype, which denotes a list of node types, connections, and arguments.

each node. In GRAPE, each node has two parts, one for processing and the other for branching. The processing component executes several types of processing using the data set, e.g., arithmetic calculation and Boolean calculation. After processing is complete, the next node is selected. The branching component decides the next node according to the data set. When the output node is reached, the GRAPE program outputs data and the program terminates. The representation of GRAPE is a graph structure; therefore, it can represent complex programs (e.g., branches and loops) using this structure. There are several data types in the 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 an evolutionary method, genotype-phenotype mapping is used in GRAPE. This genotype-phenotype mapping method is similar to Cartesian GP (CGP) [12]. 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. GRAPE has been shown to generate a variety of programs successfully, such as factorial, Fibonacci sequence, exponentiation, list reversing, and list sorting.

3 Evolution of Search Algorithms using GRAPE

In this study, an action program of searching agents is evolved by GRAPE. The agents search for optimal parameters in the search space. The search algorithm is achieved using the agents control program represented by GRAPE. The agents decide the next search point by moving through the search space in this model.

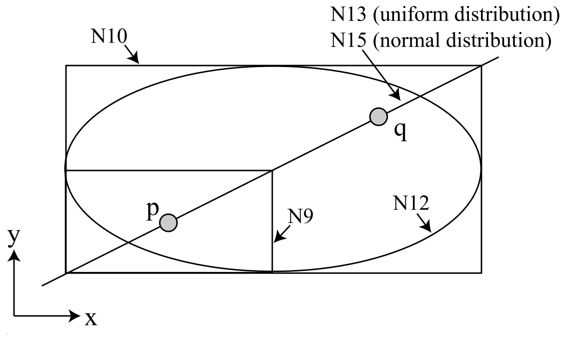


Fig. 2. Distributions of generated parameters using the prepared node functions (two dimensions).

An example of the GRAPE program is shown in Figure 1. The data set (parameters) of GRAPE contains the parameters of the target problem. The data set flows the directed graph and is updated at each node. Following this, the parameters are evaluated when the evaluation node is reached, and then it moves to the next node. After the evaluations of the predefined numbers, the search finishes.

Table 1 shows the node functions used in the experiments. Examples of node functions used in the experiments are the initialization of parameters with uniformly distributed random numbers, the substitution of other parameters, and the update of the parameters using two parameter sets with either uniformly distributed random numbers or normally distributed random numbers. When the parameters are updated, the agent (p) selects the best global parameters ($q0$), the best local parameters ($q1$), or the parameters of the random selected agent ($q2$), shown as parameters "q" in Table 1. The best global parameters are the best parameters of all agents; The best local parameters are the best parameters of the agent. These nodes are based on operators that are usually used in real-coded genetic algorithms [13, 14]. For instance, N16 is the $BLX-\alpha$ [13]. Examples of generated parameters using these node functions are shown in Figure 2. This figure shows examples in the case of two-dimensional parameters. Moreover, the evaluation node (EVAL) has three branching components, which correspond to an updating global best, an updating local best, and another case scenario.

A better search algorithm is evolved based on the results using the search algorithm represented by GRAPE. While the operator at each node is relatively simple, it can construct complex search algorithms using a combination of these operators.

The fitness assignment of each individual is based on the results of the search for target search spaces, e.g., the summed best fitness for several search simulations. In this manner, a search algorithm, which is efficient for specific problems, is obtained.

Table 1. Node functions used in the experiments.

Id	# Connection	Argument	Definition
For a random selected parameter p_r			
N1	1	–	Initialize the parameter p_r with uniform random number.
N2	1	q	$p_r = q_r$
N3	1	q	$p_r = p_r + u(p_r - \alpha\|p_r - q_r\|, p_r + \alpha\|p_r - q_r\|)$
N4	1	q	$p_r = u(min(p_r, q_r) - \alpha\|p_r - q_r\|, max(p_r, q_r) + \alpha\|p_r - q_r\|)$
N5	1	q	$p_r = N(p_r, (\alpha\|p_r - q_r\|)^2)$
N6	1	q	$p_r = N(\frac{(p_r+q_r)}{2}, (\alpha\|p_r - q_r\|)^2)$
For the all parameters p_i $(i = 1, 2, 3, ...)$			
N7	1	–	Initialize the all parameters p_i with random number.
N8	1	q	$p_i = q_i$
N9	1	q	$p_i = p_i + u(p_i - \alpha\|p_i - q_i\|, p_i + \alpha\|p_i - q_i\|)$
N10	1	q	$p_i = u(min(p_i, q_i) - \alpha\|p_i - q_i\|, max(p_i, q_i) + \alpha\|p_i - q_i\|)$
N11	1	q	$p_i = N(p_i, (\alpha\|p_i - q_i\|)^2)$
N12	1	q	$p_i = N(\frac{(p_i+q_i)}{2}, (\alpha\|p_i - q_i\|)^2)$
Other types			
N13	1	q	$p_i = (p_i - \alpha de_i) + R \cdot ((q_i + \alpha de_i) - (p_i - \alpha de_i))$ d: the distance between p_i and q_i, $e_i = \frac{(q_i - p_i)}{\|q_i - p_i\|}$, $R = u(0.0, 1.0)$
N14	1	q	$p_i = p_i + ze_i$ $z = N(0, (\alpha d)^2)$, d: the distance between p_i and q_i, $e_i = \frac{(q_i - p_i)}{\|q_i - p_i\|}$
N15	1	q	$p_i = \frac{(p_i+q_i)}{2} + ze_i$ $z = N(0, (\alpha d)^2)$, d: the distance between p_i and q_i, $e_i = \frac{(q_i - p_i)}{\|q_i - p_i\|}$
N16	2	–	Decide the next node with rondom number. If $u(0.0, 1.0) < 0.5$, then connection 1 is chosen. Else connection 2 is chosen.
Evaluation node			
EVAL	3	–	Evaluate the current parameters p. And if global best is updated, then connection 1 is chosen. Else if local best is updated, then connection 2 is chosen. Else connection 3 is chosen.

p_i: ith parameter of agent p.
q_i: ith parameter of partner agent q.
(q is best global parameters ($q0$) or best local parameters ($q1$)
or parameters of other randomly selected agents ($q2$).)
$u(x, y)$: uniformly distributed random number among $[x, y]$.
$N(\mu, \sigma^2)$: normally distributed random number.
Parameter $\alpha = 0.5$.

4 Experiments and Results

In this section, we apply the proposed method to the evolution of search algorithms. The target problems are the benchmark function optimization and the template matching problems. Template matching is an important real-world problem. After evolving the search algorithm, we apply it to test problems that are not used during evolution in order to verify its efficiency and versatility.

4.1 Experimental Settings

The parameters used in the experiments are listed in Table 2. In order to avoid the problem caused by the non-reaching structures of the evaluation node, we limited the number of execution steps to 50. When the program reaches the execution limit, the individual is assigned the worst fitness value. We used uniform

Table 2. Parameters used in the experiments.

Parameters for GRAPE	
Number of generations	10000
Population size	100
Children size (for MGG)	20
Crossover rate	0.6
Uniform crossover rate (P_c)	0.1
Mutation rate (P_m)	0.02
Maximum number of nodes	50
Execution step limits	50
Parameters for search simulation	
Number of agents	1, 10, 50
Number of function evaluations	10000 (benchmark functions)
	5000 (template matching)
Number of trials (N_t)	5

crossover and mutation as the genetic operators. The Minimal Generation Gap (MGG) [15, 16] is used as the generation alternation model. The MGG model 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 other conventional models in a wide range of applications. GRAPE used with the MGG model shows higher performance than GRAPE used with Simple GA in a variety of program evolutions [2].

4.2 Benchmark Function

The Rastrigin-d function is used in order to evaluate the search algorithm represented by GRAPE. The Rastrigin-d function is defined in equation (1) and is a multimodal function.

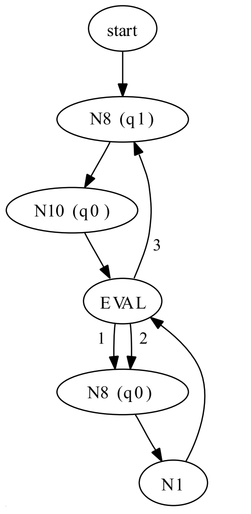
$$f(x_1, ..., x_n) = 10n + \sum_{i=1}^{n} \{(x_i - d)^2 - 10cos(2\pi(x_i - d))\} \qquad (-5.12 \leq x_i \leq 5.12) \tag{1}$$

where n is the number of dimensions. This function has the global minimum value 0 at $(d, ..., d)$. In this experiment, we use $n = 20$ and $d = 1.0$.

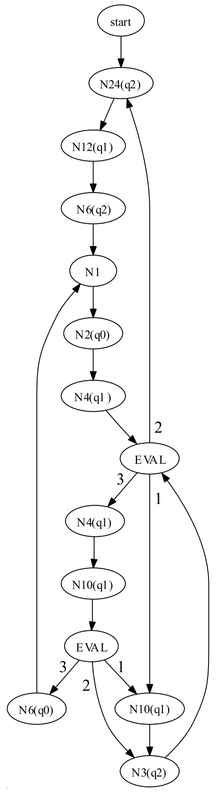
The parameters are randomized with the domain of definition. Then, search is performed using the algorithm represented by GRAPE for a predefined number of evaluations. We use 1, 10, and 50 as the number of agents in order to verify the influence of this parameter. If the number of agents is greater than 1, all agents refer to the same search algorithm represented by GRAPE (this model is homogeneous). Equation (2) is used as the fitness function of each individual in GRAPE.

$$fitness = -\sum_{i=1}^{N_t} f_i \tag{2}$$

where f_i is the best evaluation value of the ith search simulation, and N_t is the number of trials. The higher the numerical value, the better is the performance.



(a) Function optimization



(b) Template matching

Fig. 3. Obtained search algorithm structures for (a) a benchmark function optimization problem and (b) a template matching problem using GRAPE (number of agents is 10).

We tested the evolution of the search algorithms using these experimental settings and obtained a search algorithm represented by GRAPE, which is efficient with respect to the Rastrigin-d function.

Figure 3 (a) shows an example of the obtained structure (graph-structured search algorithm) constructed by the proposed method when the number of agents is 10. Unnecessary nodes are deleted in this structure. The processing branches to two through the difference in the evaluation of the structure. This structure is relatively simple. However, we can see a more complex structure that is difficult to understand at sight in other experiments.

Next, we apply the obtained search algorithms to other test functions. We use the famous seven test functions in Table 3. The experimental condition is as follows:

Table 3. Test functions used in the experiments. Sphere, Ridge, and Rosenbrock are unimodal functions. Rastrigin, Ackley, and Schwefel are multimodal functions. n is the number of dimensions.

Name	Functions	Range of x_i	f_{min}
Sphere	$f = \sum_{i=1}^{n} x_i^2$	$[-100, 100]$	0
Ridge	$f = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$	$[-100, 100]$	0
Rosenbrock	$f = \sum_{i=1}^{n-1}\{100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\}$	$[-30, 30]$	0
Rastrigin	$f = 10n + \sum_{i=1}^{n}\{x_i^2 - 10cos(2\pi x_i)\}$	$[-5.12, 5.12]$	0
Ackley	$f = 20 - 20exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) + e - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i))$	$[-32, 32]$	0
Schwefel	$f = \sum_{i=1}^{n} -x_i sin(\sqrt{\lvert x_i \rvert})$	$[-500, 500]$	$-418.98 * n$

- Number of dimensions n: $10, 20, 30$
- Number of trials: 50
- Number of function evaluations: 5.0×10^5

We use a sufficient value as the number of function evaluations.

For comparison with conventional search algorithms, we compare the performance of our method with that of particle swarm optimization (PSO) [17, 18]. PSO is one of a number of search algorithms introduced by Kennedy and Eberhart. It is categorized as an evolutionary algorithm and is based on an analogy of the movement of the flight of a flock of birds. Each particle represents a searching point in search space. The particles start at a random initial position and search for the minimum or maximum of a given objective function by moving through the search space. Each particle includes information relating to "a velocity vector," "a current position vector," and "own best position vector." Moreover, all particles have the same information, i.e., "a position vector of the best particle of all (global best)." At each iteration, a new velocity vector and a new position vector for each particle update provides a better solution according to these four factors. The updating equations are

$$v_i = \omega v_i + c_1 r_1(\hat{x} - x_i) + c_2 r_2(\hat{x}_g - x_i) \tag{3}$$

$$x_i = x_i + v_i \tag{4}$$

where x_i is the ith particle's position vector, V_i is the ith particle's velocity vector, $\hat{x}$ is the ith particle's previous best position vector, and $\hat{x}_g$ is the global best vector. The parameters ω, c_1, and c_2 are the coefficients of weight. We set $\omega = 0.72, c_1, c_2 = 1.49$. These values are recommended in [18]. The variables

Table 4. The average performances for benchmark functions determined by applying the evolved search algorithm and PSO (average over 50 trials with the same parameter set). n is the number of dimensions. The values shown in the parentheses are calculated using standard deviation.

Function	n	# agents = 1	# agents = 10	# agents = 50	PSO
	10	2.36×10^{-32} (6.58×10^{-32})	1.72×10^{-116} (8.25×10^{-116})	4.10×10^{-119} (1.38×10^{-118})	0.0 (0.0)
Sphere	20	7.05×10^{-28} (1.51×10^{-27})	2.54×10^{-81} (7.96×10^{-81})	2.41×10^{-84} (7.24×10^{-84})	4.94×10^{-324} (0.0)
	30	3.91×10^{-25} (7.38×10^{-25})	2.92×10^{-65} (1.29×10^{-64})	1.77×10^{-67} (9.35×10^{-67})	6.32×10^{-179} (0.0)
	10	2.10×10^{-30} (1.06×10^{-29})	3.25×10^{-114} (2.17×10^{-113})	8.62×10^{-117} (3.84×10^{-116})	1.22×10^{-276} (0.0)
Ridge	20	4.00×10^{-25} (2.04×10^{-24})	3.27×10^{-79} (1.11×10^{-78})	1.28×10^{-82} (3.27×10^{-82})	5.33×10^{2} (1.63×10^{3})
	30	1.03×10^{-22} (1.66×10^{-22})	5.64×10^{-62} (3.75×10^{-61})	2.53×10^{-65} (9.50×10^{-65})	4.07×10^{3} (5.05×10^{3})
	10	8.22 (7.48)	4.36 (5.95)	4.74 (6.30)	3.74×10^{3} (1.78×10^{4})
Rosenbrock	20	9.75 (8.26)	4.90 (6.04)	7.14 (7.88)	9.07×10^{3} (2.73×10^{4})
	30	1.20×10^{1} (1.49×10^{1})	8.65 (7.92)	7.49 (1.15×10^{1})	1.11×10^{4} (2.94×10^{4})
	10	7.39×10^{-15} (1.07×10^{-14})	4.52×10^{-14} (6.14×10^{-14})	1.11×10^{-14} (1.34×10^{-14})	6.27 (3.35)
Rastrigin	20	4.04×10^{-14} (4.14×10^{-14})	3.16×10^{-13} (2.76×10^{-13})	7.22×10^{-14} (5.25×10^{-14})	3.79×10^{1} (1.18×10^{1})
	30	1.51×10^{-13} (9.69×10^{-14})	2.80×10^{-12} (9.27×10^{-12})	2.58×10^{-13} (1.37×10^{-13})	9.38×10^{1} (2.91×10^{1})
	10	1.68×10^{-14} (5.36×10^{-15})	3.42×10^{-14} (2.76×10^{-14})	1.44×10^{-14} (6.27×10^{-15})	4.00×10^{-15} (3.61×10^{-2})
Ackley	20	4.80×10^{-14} (1.58×10^{-14})	1.57×10^{-13} (2.88×10^{-13})	3.42×10^{-14} (1.11×10^{-14})	3.29×10^{-1} (2.23×10^{-2})
	30	2.29×10^{-13} (1.11×10^{-13})	4.23×10^{-13} (1.10×10^{-12})	5.69×10^{-14} (1.20×10^{-14})	1.79 (8.86)
	10	-4.19×10^{3} (9.09×10^{-13})	-4.19×10^{3} (1.66×10^{1})	-4.19×10^{3} (9.09×10^{-13})	-3.52×10^{3} (2.40×10^{2})
Schwefel	20	-8.38×10^{3} (1.82×10^{-12})	-8.38×10^{3} (1.82×10^{-12})	-8.38×10^{3} (1.82×10^{-12})	-6.29×10^{3} (4.76×10^{2})
	30	-1.26×10^{4} (7.28×10^{-12})	-1.26×10^{4} (7.28×10^{-12})	-1.26×10^{4} (7.28×10^{-12})	-8.90×10^{3} (5.73×10^{2})

r_1 and r_2 are random numbers in the range $[0.0, 1.0]$. In the experiments, the number of particles is 50.

Table 4 shows the search results for 50 different runs with the same parameter set. According to these results, the evolved search algorithms perform significantly better than the conventional PSO algorithm. The evolved search algorithms are effective not only in relation to the Rastrigin function for the training search space, but also in terms of other test functions. It also shows better performance in a variety of problem dimensions. Although the search algorithms are generated using only the Rastrigin-d function (20 dimensions), they can be useful for other types of test functions.

4.3 Template Matching

Template matching is an important technique for investigating similarities between two patterns, predefined patterns (template) and target patterns. Template matching is also useful for the detection of an object by determining its position, angle, and magnification ratio in an image. For the detection of a two-dimensional object, four parameters (x-coordinate, y-coordinate, angle, and magnification ratio) must be optimized in template matching. We use a simple objective function: the normalized sum of absolute difference between the template image and the target image. The equation for the objective function in the experiments is as follows:

$$\begin{aligned} f(x, y, rate, angle) &= \tfrac{1}{W \cdot H \cdot V_{max}} \sum_{i=1}^{W} \sum_{j=1}^{H} |f_{i'j'} - t_{ij}| \\ i' &= rate * ((i - \tfrac{W}{2}) * cos(angle) + (j - \tfrac{H}{2}) * sin(angle)) + x \\ j' &= rate * ((i - \tfrac{W}{2}) * sin(angle) + (j - \tfrac{H}{2}) * cos(angle)) + y \end{aligned} \qquad (5)$$

where t_{ij} is the brightness of the template image, f_{ij} is the brightness of the target image, W and H denote the image size of the template image, and V_{max} is the maximum brightness. This objective function should be minimized.

The template images and the target image used in the experiment are shown in Figure 4. These are grayscale images, with the size of the template image being 64×64 pixels, and the size of the target images 640×480 pixels. The ranges of each parameter are as follows:

– Positions of template image: $0 \leq x \leq X, 0 \leq y \leq Y$ (X and Y denote the image size of the target image)
– Magnification ratio: $1.0 \leq rate \leq 2.0$
– Angle: $-180° \leq angle \leq 180°$

The parameters used in the experiment are shown in Table 2. Target image 1 and the template image in Figure 4 are used to evolve the search algorithm represented by GRAPE. Equation (2) is also used as the fitness function of each individual in GRAPE. By these settings, we obtain a search algorithm, which is effective to template matching. Figure 3 (b) shows an example of the obtained structure (graph-structured search algorithm) constructed by the proposed method when the number of agents is 10.

The performances of the evolved search algorithms and PSO are shown in Table 5. The number of evaluations is 10000. We use the two target images shown in Figure 4. The evolved search algorithms perform better than the conventional PSO algorithm for both target images. Therefore, we can conclude that general search algorithms for template matching can be constructed automatically.

5 Conclusions and Future Work

In this paper, we proposed a new method for the automatic construction of search algorithms based on GRAPE, which evolves graph-structured programs.

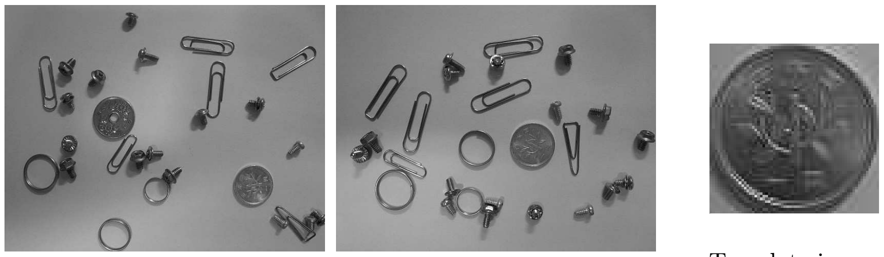

Target image 1 Target image 2 Template image

Fig. 4. Target images and the template image used in the experiments. Target image 1 is used for constructing the search algorithms. Target image 2 is applied to the evolved search algorithm.

Table 5. The average performances for template matching problems found by applying the evolved search algorithm and PSO (average over 50 trials with the same parameter set). Best/Worst refers to the fitness of the best individual in the best/worst run, respectively. The number of evaluations is 10000.

	# agents=1	# agents=10	# agents=50	PSO
		Target image 1		
Average	0.0890	0.0889	0.0890	0.105
Stddev	0.00501	0.00578	0.00424	0.0137
Best	0.0846	0.0845	0.0845	0.0858
Worst	0.107	0.107	0.103	0.134
		Target image 2		
Average	0.0735	0.0726	0.0714	0.0793
Stddev	0.00684	0.00328	0.00281	0.0113
Best	0.0698	0.0698	0.0698	0.070
Worst	0.116	0.0776	0.0790	0.132

We applied the proposed method to evolve search algorithms for function optimization and template matching problems and confirmed that efficient search algorithms for each problem were obtained by the proposed method. From the experimental results, the performances of evolved search algorithms were found to outperform a conventional PSO algorithm. In future work, we will apply the proposed method to other types of problems. Further, we will propose a method for evolving search algorithms with a heterogeneous model.

References

1. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
2. Shirakawa, S., Ogino, S., Nagao, T.: Graph Structured Program Evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007). Volume 2., London, UK, ACM Press (2007) 1686–1693

3. Shirakawa, S., Nagao, T.: Evolution of sorting algorithm using graph structured program evolution. In: Proceedings of the 2007 IEEE International Conference on Systems, Man and Cybernetics (SMC 2007), Montreal, Canada (2007) 1256–1261
4. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1) (1997) 67–82
5. Tavares, J., Machado, P., Cardoso, A., Pereira, F.B., Costa, E.: On the evolution of evolutionary algorithms. In: Proceedings of the Genetic Programming 7th European Conference (EuroGP 2004). Volume 3003 of LNCS., Coimbra, Portugal, Springer-Verlag (2004) 389–398
6. Oltean, M.: Evolving evolutionay algorithm using linear genetic programming. Evolutionary Computation **13**(3) (2005) 387–410
7. Dioşan, L., Oltean, M.: Evolving crossover operators for function optimization. In: Proceedings of the 9th European Conference on Genetic Programming (EuroGP 2006). Volume 3905 of LNCS., Budapest Hungary, Springer-Verlag (2006) 97–108
8. Dioşan, L., Oltean, M.: Evolving the structure of the particle swarm optimization algorithms. In: Proceedings of the 6th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006). Volume 3906 of LNCS., Budapest Hungary, Springer-Verlag (2006) 25–36
9. Burke, E.K., Hyde, M.R., Kendall, G.: Evolving bin packing heuristics with genetic programming. In: Parallel Problem Solving from Nature - PPSN IX. Volume 4193 of LNCS., Reykjavik, Iceland, Springer-Verlag (2006) 860–869
10. Kumar, R., Joshi, A.H., Banka, K.K., Rockett, P.I.: Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008), Atlanta, GA, USA, ACM Press (2008) 1227–1234
11. Pillay, N., Banzhaf, W.: A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. In: Proceedings of the EPIA Workshops. Volume 4874 of LNCS., Guimarães, Portugal, Springer-Verlag (2007) 223–234
12. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Proceedings of the 3rd European Conference on Genetic Programming (EuroGP 2000). Volume 1802 of LNCS., Edinburgh, Springer-Verlag (2000) 121–132
13. Eshelman, L.J., Schaffer, J.D.: Real coded genetic algorithms and interval-schemata. In: Foundations of Genetic Algorithms 2. (1993) 187–202
14. Ono, I., Kobayashi, S.: A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover. In: Proceedings of the 7th International Conference on Genetic Algorithms, East Lansing, MI, USA, Morgan Kaufmann (1997) 246–253
15. Satoh, H., Yamamura, M., Kobayashi, S.: Minimal generation gap model for considering both exploration and exploitations. In: Proceedings of the IIZUKA'96. (1996) 494–497
16. Deb, K., Anand, A., Joshi, D.: A computationally efficient evolutionary algorithm for real-parameter optimization. Evolutionary Computation **10**(4) (2002) 371–395
17. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks. Volume IV., Perth, Australia (1995) 1942–1948
18. Eberhart, R.C., Shi, Y.: Comparing inertia weights and constriction factors in particle swarm optimization. In: Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000). Volume 1. (2000) 84–88