6

Automatic Construction of Programs Using Dynamic Ant Programming

Shinichi Shirakawa, Shintaro Ogino, and Tomoharu Nagao
Yokohama National University
Japan

1. Introduction

Automatic programming is the research field of generating computer programs automatically. Genetic programming (GP) (Koza, 1992; 1994) is a typical example of automatic programming, which was proposed by Koza. GP evolves computer programs with tree structure based on genetic algorithm (GA). GP has, however, the problem of bloating (Langdon & Poli, 1998; Poli, 2003), the growth of the average size of an individual in the population. This chapter introduces a new method for automatic programming using ant colony optimization (ACO) (Dorigo et al., 1999; Dorigo & Stutzle, 2004). ACO is a technique for solving combinatorial optimization problems. ACO algorithm is inspired by the behavior of the ants.

Several automatic programming techniques using ACO were investigated (Engelbrecht, 2006). Typical examples are ant programming (AP) (Roux & Fonlupt, 2000), ant colony programming (ACP) (Boryczka & Czech, 2002), AntTAG (Abbass et al., 2002) and generalized ant programming (GAP) (Keber & Schuster, 2002). AP is similar to Probabilistic Incremental Program Evolution (PIPE) (Salustowicz & Schmidhuber, 1997), and it was used to solve the symbolic regression problems. PIPE generates successive populations from a probabilistic prototype tree. ACP was used to solve the symbolic regression problems with two different versions that are the expression approach and the program approach. AntTAG and GAP are grammar-based work.

The method proposed in this chapter is named dynamic ant programming (DAP). DAP is based on ACO and generates desired programs using the dynamically changing pheromone table. The nodes (terminal and nonterminal) are selected using the value of the pheromone table. The higher the rate of pheromone, the higher is the probability that it can be chosen. The unnecessary node in DAP is deleted according to pheromone value. Therefore, the average size of programs tends to be small, and it is possible to search desired programs effectively. In order to verify the effectiveness, we applied the proposed method to the symbolic regression problem that is widely used as a test problem for GP systems. We compare the performance of DAP to GP and show the effectiveness of DAP. In order to investigate the influence of several parameters, we compare experimental results obtained using different settings.

This chapter consists of five sections. Section 2 is an overview of some related works. Section 3 describes DAP. Several experiments are shown in Section 4. Section 5 is devoted to the conclusions and the discussion of future works.

2. Related works

2.1 Genetic programming

GP is a method of automatic programming, which was introduced by Koza (Koza, 1992; 1994). It evolves computer programs with tree structure, and searches a desired program using GA. A population (a set of programs) will evolve by repeatedly selecting the fitter programs, combining them, and producing new programs. In GP, individuals in the population are computer programs. These programs are usually represented as trees. This tree-form representation is a useful way of representing LISP expression (also known as S-expression). The internal nodes represent the functions (non-terminal) and the leaves are the terminal nodes. GP has the problem of bloating (Langdon & Poli, 1998; Poli, 2003) potentially, the growth of the average size of an individual in the population. The general GP method can be roughly described as follows:

1. Random generation of programs (trees);
2. Evaluation of programs (fitness measure);
3. Application of genetic operations to individuals: crossover, mutation;
4. Selection of individuals;
5. Go to step 2 until some termination criterion is satisfied.

2.2 Ant colony optimization

ACO (Dorigo et al., 1999; Dorigo & Stutzle, 2004) is a technique for solving combinatorial optimization problems, and it was introduced in the early 1990's. ACO algorithm is inspired by the behavior of real ants. It has been applied to many combinatorial optimization problems, e.g. Traveling Salesman Problems (TSP) (Dorigo et al., 1996; Stutzle & Hoos, 2000), network routing problems (Schoonderwoerd et al., 1996; Caro & Dorigo, 1998), graph coloring problems (Costa & Hertz, 1997), and so on. The first ACO algorithm, called ant system (AS) Dorigo et al. (1996), was proposed by Dorigo et al. and applied to the TSP. Figure 1 shows the AS algorithm.

An ant located at city i selects an edge between city i and city j by using the probabilities:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k(t)} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}} & \text{if } j \in N_i^k(t) \\ 0 & \text{if } j \notin N_i^k(t) \end{cases} \quad (1)$$

definition m: the number of ants
Initialize
repeat
  for $k = 1$ to m **do**
    repeat
      From current node i, select next node j with probability as defined in Equation (1);
    until *full path has been constructed;*
  end
  Update pheromone using Equation (3);
until *stopping condition is true;*

Fig. 1. Ant system (AS) algorithm

where $\tau_{ij}(t)$ is the amount of pheromone trail on edge (i, j) at time t, a heuristic value η_{ij} equals to $1/d_{ij}$, where d_{ij} is the distance between city i and j, α and β are parameters which control the relative weight of the pheromone trail and the heuristic value respectively, where N_i^k is the set of cities which remains to be visited. After every ant completes a tour, pheromone is deposited:

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{Q}{L_k} & \text{if } (i,j) \in T_k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where Q is a constant; L^k is the length of the tour generated by ant k; T_k is the tour generated by ant k. The amount of pheromone is updated according to the rule:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \tag{3}$$

where m is the number of ants and $\rho(0 < \rho < 1)$ is a parameter called evaporation rate. These processes are repeated a predefined $t_{\max}$ number of times.

In addition to finding very good solutions to 'static' problems, the ACO algorithm is also effective in 'dynamic' problems (Guntsch & Middendorf, 2001; Guntsch et al., 2001; Caro & Dorigo, 1998; Schoonderwoerd et al., 1996). The ACO algorithm maintains portions of solution as the pheromone trails, which can be especially useful when the problem is dynamically changing (Bonabeau et al., 2000).

Several extended and improved versions of the original AS algorithm were introduced. For instance, ant colony system (ACS) (Dorigo & Gambardella, 1997), elitist ant system (EAS) (Dorigo et al., 1996), rank-based ant system (RAS) (Bullnheimer et al., 1999), and MAX-MIN ant system (MMAS) (Stutzle & Hoos, 1997; 2000). One of the most successful ACO variants is MMAS. MMAS limits the possible range of pheromone trail values to the interval $[\tau_{\min}, \tau_{\max}]$. Additional difference is that only the best ant may reinforce pheromones, and initial pheromones are set to the maximum allowed value. The pheromone update rule in MMAS is:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{best}(t) \tag{4}$$

where $\Delta\tau_{ij}^{best}(t)$ is calculated on the basis of either the global-best path or the iteration-best path.

2.3 Ant colony optimization for automatic programming

Several automatic programming techniques based on ACO were investigated (Engelbrecht, 2006). AP (Roux & Fonlupt, 2000) appears to be the earliest attempt at using ACO to automatic programming. It is similar to PIPE (Salustowicz & Schmidhuber, 1997), and it was used to solve symbolic regression problems. PIPE generates successive populations from a probabilistic prototype tree. AntTAG (Abbass et al., 2002) and GAP (Keber & Schuster, 2002) are grammar-based works. AntTAG is an ACO algorithm which constructs programs using tree-adjunct grammar (TAG). GAP system uses context-free grammar.

The most related work to our proposed method is ACP (Boryczka & Czech, 2002). ACP was used for solving the symbolic regression problems with two different versions that are the expression approach and the program approach. The expression approach constructs an expression in prefix notation, while the program approach constructs an

```
definition m: the number of ants
Initialize
repeat
  for k = 1 to m do
    repeat
      From current node i, select next node j with probability as defined in Equation (5);
    until full path has been constructed (Ant completely has reached terminal);
  end
  Update pheromone using Equation (7);
  Deletion of nodes;
  Insertion of nodes;
until stopping condition is true;
```

Fig. 2. Dynamic ant programming (DAP) algorithm

expression from a sequence of assignment instructions. Both approaches base on the ACS (Dorigo & Gambardella, 1997). In expression approach, the components of graph $G = (N, E)$ have the following meaning: N is the set of nodes, where each node represents either a terminal symbol (i.e. a constant or variable) or a non-terminal symbol (i.e. an arithmetic operator or function). Each link in E represents a branch of the expression tree. The tabu list is not used, since the multiple occurrences of a node in the expression are not prohibited.

3. Dynamic ant programming

DAP is a novel automatic programming method using ACO. GP has the problem of bloating, potentially the growth of the average size of an individual in the population. In the conventional method, ACP, the number of nodes is fixed beforehand, and the tabu list is not used. Therefore, it tends to select the same node repeatedly and has the multiple occurrence of the same expression in one program.

The unnecessary node in DAP is deleted according to pheromone value. The size of the pheromone table in DAP changes dynamically in each iteration, and the number of nodes is variable. Therefore, the average size of programs tends to be small, and it is possible to search desired programs effectively. The tabu list is used in DAP; thus, it is possible to generate various programs. Figure 2 shows the DAP algorithm.

3.1 Construction of tree structure

First, the ant starts at *start node* (*start node* is an initial node for ants). The ant chooses the next node using pheromone values. The ant never selects a nonterminal node which was already selected (i.e. the tabu list is used), whereas terminal node can be visited by the ants repeatedly. If the selected node is a terminal node, the search is finished. The higher the rate of pheromone, the higher the probability that it can be chosen. The ant located in node i selects only the number of argument of node i using the pheromone value in the subsequent nodes. Therefore, the pheromone table size in DAP is:

- column size
  total number of nodes (terminal and nonterminal).
- row size
  total number of arguments + 1 (*start node).*

	Arg No.	x	1.0	$+$	$*$	sin
Start	–			1		
$+$	1	2				
	2					3
$*$	1	5				
	2		6			
sin	1				4	

Table 1. Example of pheromone table in DAP

Table 1 shows an example of the pheromone table, and Figure 3 illustrates how to construct tree structure using the pheromone table. The ant visits the sequence of nodes $\{\text{Start} \rightarrow + \rightarrow x \rightarrow \sin \rightarrow * \rightarrow x \rightarrow 1.0\}$ and constructs a numerical formula '$x + \sin(x * 1.0)$'.
The probability of ant k located in node i moving to node j through argument u at time t is:

$$p_{i_u j}^k(t) = \begin{cases} \frac{\tau_{i_u j}(t)}{\sum_{l \in N_i^k(t)} \tau_{i_u l}(t)} & \text{if } j \in N_i^k(t) \\ 0 & \text{if } j \notin N_i^k(t) \end{cases} \tag{5}$$

where $\tau_{i_u j}(t)$ is the amount of pheromone trail on edge (i_u, j) at time t; $N_i^k(t)$ is the set of nodes that remains to be visited.

3.2 Pheromone update

The pheromone update rules in DAP is similar to MMAS (Stutzle & Hoos, 2000). To avoid stagnation of search, a possible range of pheromone value is limited to an interval $[\tau_{\min}, \tau_{\max}]$. The parameters $\tau_{\min}$ and $\tau_{\max}$ are determined in advance. After each program (individual) is evaluated, the pheromone table is updated according to:

$$\Delta\tau_{ij}^{\text{best}}(t) = \begin{cases} f^{\text{ib}} & \text{if}(i,j) \in T_{\text{ib}} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where f^{ib} is a fitness of iteration-best ant and T_{ib} is the tour (program) generated by iteration-best ant. The amount of pheromone is updated according to the rule:

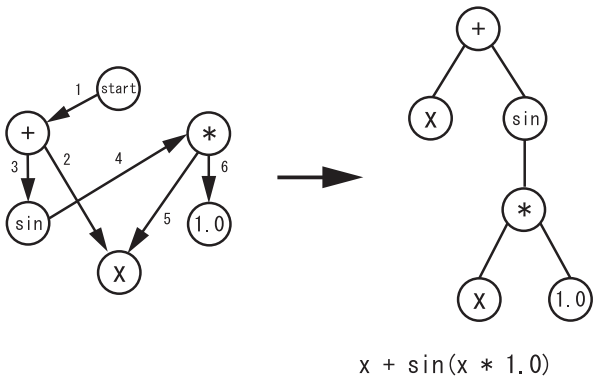


Fig. 3. An example of constructing tree structure in DAP. The ant visits the sequence of nodes $\{\text{Start} \rightarrow + \rightarrow x \rightarrow \sin \rightarrow * \rightarrow x \rightarrow 1.0\}$ and constructs a numerical formula '$x + \sin(x * 1.0)$'

Terminal or function	Arg No.	x	1.0	$+$	$*$	sin
Start	-			$\tau_{\min}$		
$+$	1			$\tau_{\min}$		
	2			$\tau_{\min}$		
$*$	1			$\tau_{\min}$		
	2			$\tau_{\min}$		
sin	1			$\tau_{\min}$		

Table 2. An example of deletion of node. The node '$+$' is deleted in this case

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{\text{best}}(t) \tag{7}$$

where $\rho(0 < \rho < 1)$ is a parameter called evaporation rate. Initial pheromones are set to the maximum allowed value ($\tau_{\max}$).

3.3 Deletion and insertion of nodes

Deletion and insertion of nodes are performed after pheromone update in each iteration. So, the pheromone table changes dynamically in DAP. Table 2 illustrates an example of the deletion of node, and Table 3 illustrates an example of the insertion of node.

- Deletion of node
  If all pheromone values to the node j are minimum values $\tau_{\min}$ (if all $\tau_{lj} = \tau_{min}, l = \{1,...,n\}$), the node j is deleted, where n is the number of row size of pheromone table).
- Insertion of node
  The probability of inserting node in iteration i equals to:

$$p(i) = \frac{\frac{1}{m}\sum_{k=1}^{m} L_i^k}{N_i} \tag{8}$$

  where L^k is the number of nodes generated by ant k; N_i is total number of nodes (column size of pheromone table); and m is the number of ants. The type of node inserted is decided randomly. The pheromone value of an inserted node is set to τ_{ins}.

Although the search space (i.e. the pheromone table of DAP) is dynamically changing, the ants find good solution using portions of solutions, which are pheromone trail values.

Terminal or function	Arg No.	x	1.0	$+$	$*$	sin	sin
Start	-						τ_{ins}
$+$	1						τ_{ins}
	2						τ_{ins}
$*$	1						τ_{ins}
	2						τ_{ins}
sin	1						τ_{ins}
sin	1	τ_{ins}	τ_{ins}	τ_{ins}	τ_{ins}	τ_{ins}	τ_{ins}

Table 3. An example of insertion of node. The node 'sin' is inserted in this case

4. Experiments and results

In this section several experiments with DAP and GP are performed. We use the well-known test problem, namely, the symbolic regression.

4.1 Symbolic regression

Symbolic regression is widely used as a test problem for GP systems. The problem is to search for a function that fits sampled data points. The functions used in these experiments are:

$$f_1(x) = x^4 + x^3 + x^2 + x \tag{9}$$

$$f_2(x) = \sin(x) + x^2 + 1 \tag{10}$$

$$f_3(x) = \sin(x^3 + x) \tag{11}$$

The set of sampled data points for these problems was generated using (for the variable x) 20 uniformly chosen values in the interval [-1, 1]. We use the mean error on the sampled data points as a fitness function, which is defined as:

$$\text{fitness} = \frac{1}{1 + \sum_{i=1}^{n} |C_{\text{correct}_i} - C_{\text{estimate}_i}|} \tag{12}$$

where C_{correct_i} is the correct value for the sampled data point i; C_{estimate_i} is the value returned by the generated program for the sampled data point i; and n is the size of the sampled data points. The range of this fitness function is [0.0, 1.0]. A higher numerical value indicates better performance. The common parameters between the two methods (DAP and GP) are identical. The individual parameters of DAP and GP are given in Table 4 and Table 5 respectively.

4.2 Experimental results

Results are given for 100 different runs with the same parameter set. Table 6 shows the success rate of 100 trials at the final generation. The success rate is computed as:

$$\text{Success rate} = \frac{\text{Number of successful runs}}{\text{Total number of runs}} \tag{13}$$

According to the result, DAP obtains a better solution than GP for all test problems.
Figure 4 (a) shows the comparison of the success rate between DAP and GP for $f_1(x)$, with the learning has been pursued on. We can see that DAP reaches 98% success rate in $f_1(x)$, while the success rate of GP is 69% after 1000 generations. Figure 4 (b) shows the average number of nodes of DAP and GP for $f_1(x)$. The average number of nodes in GP expands in the evolutionary process. After 1000 generations, the average number of nodes in GP is more than 100 nodes. So, the tree structural programs of GP bloat. However, DAP keeps the average number of nodes between 10 and 20 in this experiment. It shows that DAP is more efficient than GP.
The results for $f_2(x)$ and $f_3(x)$ are shown in Figure 5 (a), (b) and 6 (a), (b). For both test problems, DAP has better results than GP, with a smaller average number of nodes.
Figure 7 is the relationship between the average number of column size of pheromone table in DAP (total number of nodes) and the number of generations. The size of pheromone table in DAP changes dynamically. The size is 20-28 for $f_1(x)$, 18-25 for $f_2(x)$, and 8-14 for $f_3(x)$. It also shows that the size of the pheromone table does not bloat.

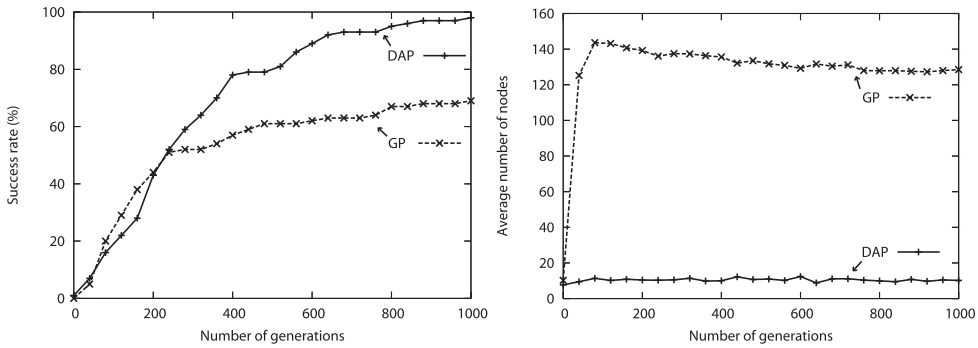
Parameter	Value
Terminal set	The variable x and the constant 1.0
Function set	$F = \{+, -, *, /, \sin\}$
The number of generations	1000
Population size	50
ρ	0.50
$\tau_{\max}$	1.0
$\tau_{\min}$	0.01
τ_{ins}	$\tau_{\max}$ $(= 1.0)$

Table 4. The parameters of DAP algorithm for symbolic regression problems

Parameter	Value
Terminal set	The variable x and the constant 1.0
Function set	$F = \{+, -, *, /, \sin\}$
The number of generations	1000
Population size	50
Crossover probability	1.0
Mutation probability	0.9 (for individual)
Selection	Tournament selection
Tournament size	2

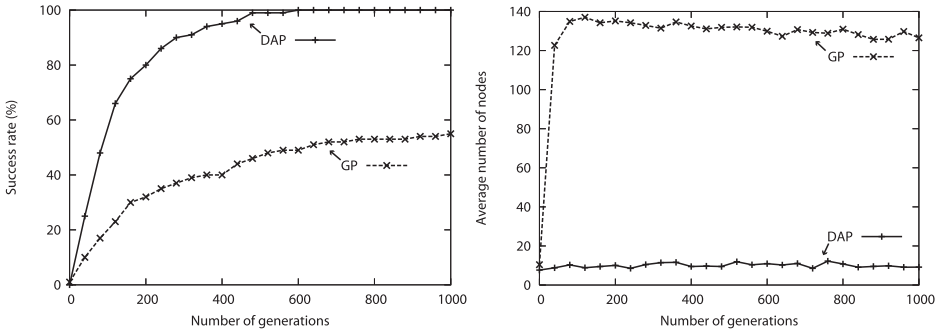
Table 5. The parameters of GP algorithm for symbolic regression problems

Incidentally, GP has a tendency to create redundant programs. In this experiment, GP does not have a factor for restraining the redundancy. On the other hand, DAP has a function for deletion of nodes. Therefore, the comparison between DAP and GP would not be fair. Figure 8 shows the comparison of the fitness between DAP and GP for $f_1(x)$. In terms of fitness value, the performance of DAP and GP is comparable. However, GP cannot find an accurate solution, while DAP has a higher success rate.



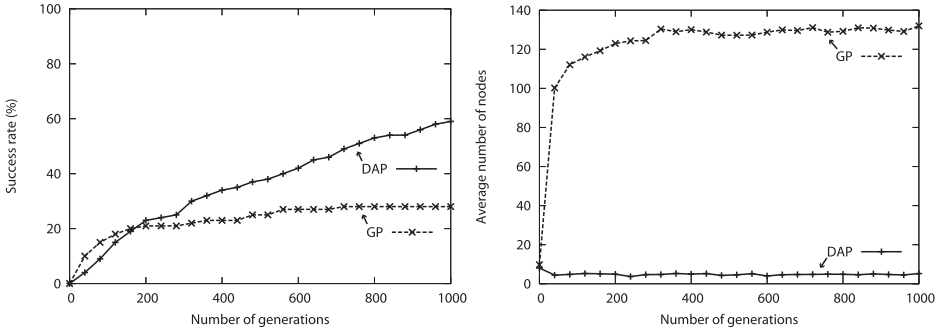
(a) Success rate (b) Average number of nodes

Fig. 4. The comparison of the success rate and the average number of nodes between DAP and GP for $f_1(x)$, with the learning has been pursued on



(a) Success rate (b) Average number of nodes

Fig. 5. The comparison of the success rate and the average number of nodes between DAP and GP for $f_2(x)$, with the learning has been pursued on



(a) Success rate (b) Average number of nodes

Fig. 6. The comparison of the success rate and the average number of nodes between DAP and GP for $f_3(x)$, with the learning has been pursued on

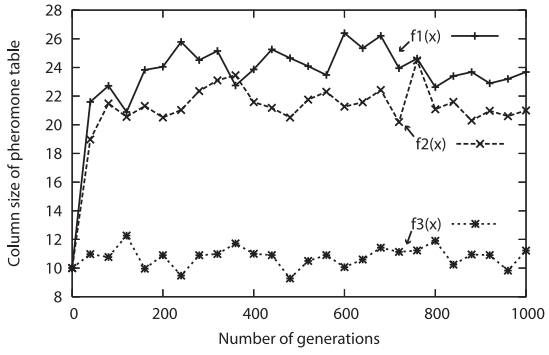


Fig. 7. The curves show the relationship between the number of column size of the pheromone table in DAP (total number of nodes) and the number of generations. Each curve is an average over 100 runs

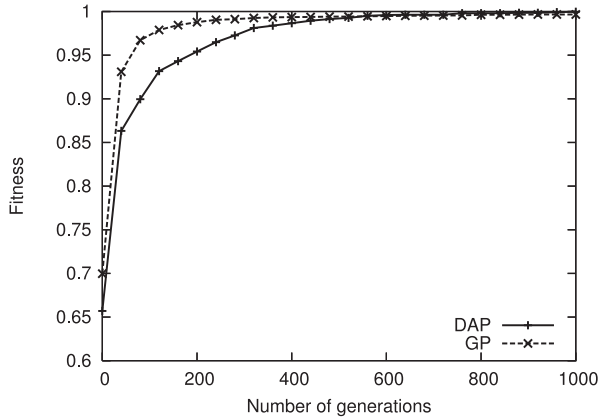


Fig. 8. The comparison of the fitness between DAP and GP for $f_1(x)$. Each curve is an average over 100 runs

	DAP (%)	GP (%)
$f_1(x)$	98	69
$f_2(x)$	100	55
$f_3(x)$	59	28

Table 6. Success rate of DAP and GP at final generation

4.3 Pheromone evaporation rate ρ

To examine the influence of the values of the pheromone evaporation rate ρ, we compare the experimental results obtained using different settings of ρ. The pheromone evaporation rate ρ is varied between 0.10 and 0.90. Results are given for 100 different runs with the same parameter set and using $f_1(x)$ as a test problem.

Table 7 shows the success rate of 100 trials at the final generation. Figure 9 showa the transitions of the success rate for $f_1(x)$ when different values have been set for the parameter ρ. In Table 7, it can be observed that better solutions are found when using higher values of ρ=0.50-0.70. This is due to the fact that the pheromone trail values decrease faster when the values of ρ are higher; hence, the size of the pheromone table tends to be small. Figure 10 shows the relationship between the average number of column size of pheromone table in DAP (total number of nodes) and the number of generations. The size of the pheromone table is small (about 10-25) when using the higher values of ρ, while the size is large (about 30-50) using the lower values of ρ. If ρ is low, it is difficult for the pheromone trail values to reach minimum value ($\tau_{\min}$).

4.4 Parameter of τ_{ins}

In order to investigate the influence of the values of τ_{ins}, we compare the experimental results obtained using different settings of τ_{ins}. τ_{ins} is the pheromone value of an inserted node, and it varies between 0.20 and 1.00. Results are given for 100 different runs with the same parameter

ρ	0.10	0.30	0.40	0.50	0.60	0.70	0.90
Success rate (%)	45	92	96	98	99	98	93

Table 7. Success rate of DAP at final generation

set using $f_1(x)$ as a test problem. The parameter of the pheromone evaporation rate ρ used is 0.50 in this experiment.

Table 8 shows the success rate of 100 trials at the final generation. The transitions of the success rate for $f_1(x)$ are shown in Figure 11. In Table 8, the success rate reaches 100% using τ_{ins}=0.80, 0.60, 0.40. In Figure 11, earlier convergence has been obtained when using a lower value of τ_{ins}. When the value of τ_{ins} is low, the inserted node has little chance to be visited, and the node will be deleted soon. Therefore, we can consider the value of τ_{ins} to be related to diversity. Figure 12 shows the transitions of the total number of nodes. The size of the pheromone table is between 20 and 30 for all parameter of τ_{ins}. There is little relationship between the size of pheromone table and the parameter of τ_{ins} although the size of pheromone table is dynamically changing.

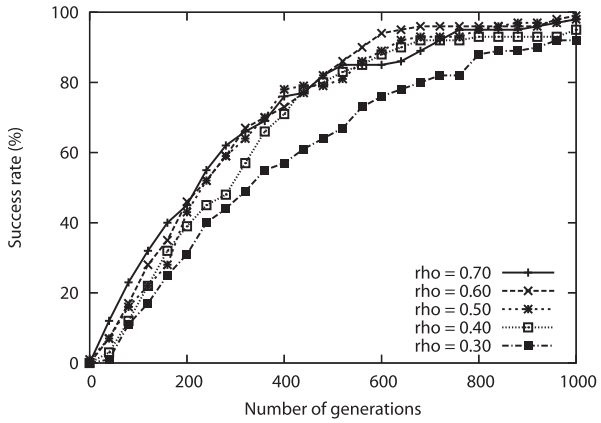


Fig. 9. The transitions of the success rate for $f_1(x)$ ($\rho = 0.30, 0.40, 0.50, 0.60, 0.70$)

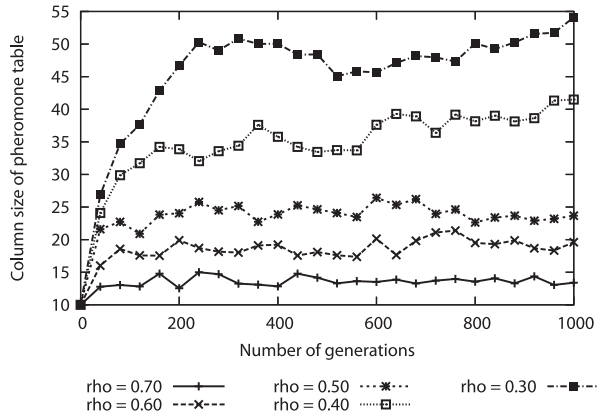


Fig. 10. The curves show the relationship between the number of column size of the pheromone table (total number of nodes) and the number of generations. Each curve is an average over 100 runs

τ_{ins}	1.00	0.80	0.60	0.40	0.20
Success rate (%)	98	100	100	100	99

Table 8. Success rate of DAP at final generation

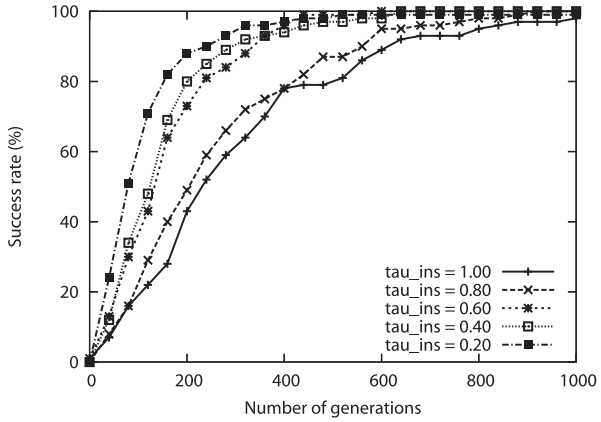


Fig. 11. The transitions of the success rate for $f_1(x)$ ($\tau_{\text{ins}} = 1.00, 0.80, 0.60, 0.40, 0.20$)

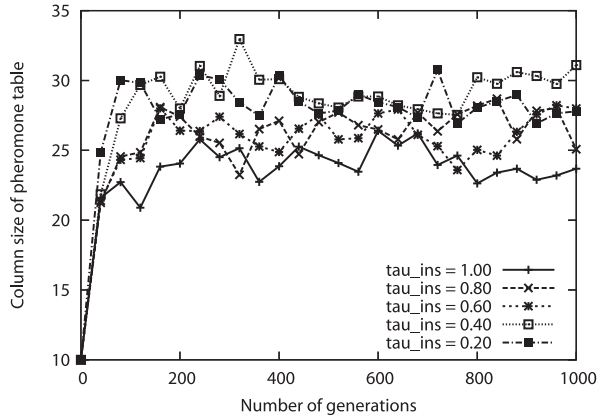


Fig. 12. The curves show the relationship between the number of column size of the pheromone table (total number of nodes) and the number of generations. Each curve is an average over 100 runs ($\tau_{\text{ins}} = 1.00, 0.80, 0.60, 0.40, 0.20$)

5. Conclusions

A novel automatic programming method based on ACO, which is DAP has been proposed in this chapter. We tested the performance of DAP against GP using the symbolic regression problems. In all test cases, DAP performed better than GP, and the search space of DAP was more compact. The size of the pheromone table of DAP is maintained at 10-30, while the tree size of GP bloats. From these tests, we can say that DAP has an advantage against GP. We also investigated the influence of the parameters of DAP. The parameter of ρ relates to the total number of nodes, and the parameter τ_{ins} relates to diversity.

In order to clarify the effectiveness of DAP, more experiments have to be run on a variety of test problems. In future works we plan to extend tests to other problems. We applied DAP to the construction of tree structural programs automatically in this work. In recent years other structures have been investigated in GP (e.g. linear structure (Brameier & Banzhaf, 2001), graph structure (Miller & Smith, 2006; Mabu et al., 2007; Shirakawa & Nagao, 2009)). We plan to apply DAP to construct other structures automatically as well.

6. References

Abbass, H., Hoai, N. & McKay, R. (2002). AntTAG: A new method to compose computer programs using colonies of ants, *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC '02)*, Honolulu, HI, USA, pp. 1654–1659.

Bonabeau, E., Dorigo, M. & Theraulaz, G. (2000). Inspiration for optimization from social insect behavior, *Nature* 406: 39–42.

Boryczka, M. & Czech, Z. J. (2002). Solving approximation problems by ant colony programming, *Late Breaking Papers at the Genetic and Evolutionary Computation Conference 2002 (GECCO '02)*, New York, pp. 39–46.

Brameier, M. & Banzhaf, W. (2001). A comparison of linear genetic programming and neural networks in medical data mining, *IEEE Transactions on Evolutionary Computation* 5(1): 17–26.

Bullnheimer, B., Hartl, R. F. & Strauss, C. (1999). A new rank based version of the ant system: A computational study, *Central European Journal for Operations Research and Economics* 7(1): 25–38.

Caro, G. D. & Dorigo, M. (1998). Antnet: Distributed stigmergetic control for communications networks, *Journal of Artificial Intelligence Research* 9: 317–365.

Costa, D. & Hertz, A. (1997). Ants can colour graphs, *Journal of the Operational Research Society* 48: 295–305.

Dorigo, M., Caro, G. D. & Gambardella, L. M. (1999). Ant algorithm for discrete optimization, *Artificial Life* 5(2): 137–172.

Dorigo, M. & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation* 1(1): 53–66.

Dorigo, M., Maniezzo, V. & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics -Part B* 26(1): 29–41.

Dorigo, M. & Stutzle, T. (2004). *Ant Colony Optimization*, MIT Press, Cambridge, MA.

Engelbrecht, A. P. (2006). *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons, New York.

Guntsch, M. & Middendorf, M. (2001). Pheromone modification strategies for ant algorithms applied to dynamic tsp, *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, Vol. 2037 of *LNCS*, Springer, London, pp. 213–222.

Guntsch, M., Middendorf, M. & Schmeck, H. (2001). An ant colony optimization approach to dynamic TSP, *Proceedings of the Genetic and Evolutionary Computation Conference 2001 (GECCO '01)*, San Francisco, pp. 860–867.

Keber, C. & Schuster, M. G. (2002). Option valuation with generalized ant programming, *Proceedings of the Genetic and Evolutionary Computation Conference 2002 (GECCO '02)*, New York, pp. 74–81.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural*

Selection, MIT Press, Cambridge, MA.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, MA.

Langdon, W. B. & Poli, R. (1998). Genetic programming bloat with dynamic fitness, *Proceedings of the First European Workshop on Genetic Programming (EuroGP '98)*, Vol. 1391 of *LNCS*, Springer, Paris, France, pp. 96–112.

Mabu, S., Hirasawa, K. & Hu, J. (2007). A graph-based evolutionary algorithm: Genetic network programming (gnp) and its extension using reinforcement learning, *Evolutionary Computation* 15(3): 369–398.

Miller, J. F. & Smith, S. L. (2006). Redundancy and computational efficiency in cartesian genetic programming, *IEEE Transactions on Evolutionary Computation* 10(2): 167–174.

Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming, *Proceedings of the 6th European Conference on Genetic Programming (EuroGP '03)*, Vol. 2610 of *LNCS*, Springer, Essex, UK, pp. 204–217.

Roux, O. & Fonlupt, C. (2000). Ant programming: Or how to use ants for automatic programming, *in* M. D. et al. (ed.), *Proceedings of ANTS '00*, Brussels, Belgium, pp. 121–129.

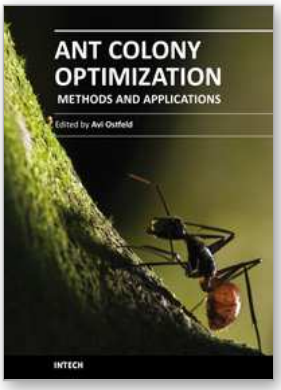
Salustowicz, R. P. & Schmidhuber, J. (1997). Probabilistic incremental program evolution, *Evolutionary Computation* 5(2): 123–141.

Schoonderwoerd, R., Holland, O. E., Bruten, J. L. & Rothkrantz, L. J. M. (1996). Ant-based load balancing in telecommunications networks, *Adaptive Behavior* 5(2): 169–207.

Shirakawa, S. & Nagao, T. (2009). Graph structured program evolution: Evolution of loop structures, *in* R. L. Riolo, U.-M. O'Reilly & T. McConaghy (eds), *Genetic Programming Theory and Practice VII*, Springer, pp. 177–194.

Stutzle, T. & Hoos, H. (1997). The MAX-MIN ant system and local search for the traveling salesman problem, *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC '97)*, IEEE Press, Indianapolis, IN, USA, pp. 308–313.

Stutzle, T. & Hoos, H. (2000). MAX-MIN ant system, *Future Generation Computer Systems* 16(8): 889–914.



Ant Colony Optimization - Methods and Applications

Edited by Avi Ostfeld

ISBN 978-953-307-157-2
Hard cover, 342 pages
Publisher InTech
Published online 04, February, 2011
Published in print edition February, 2011

Ants communicate information by leaving pheromone tracks. A moving ant leaves, in varying quantities, some pheromone on the ground to mark its way. While an isolated ant moves essentially at random, an ant encountering a previously laid trail is able to detect it and decide with high probability to follow it, thus reinforcing the track with its own pheromone. The collective behavior that emerges is thus a positive feedback: where the more the ants following a track, the more attractive that track becomes for being followed; thus the probability with which an ant chooses a path increases with the number of ants that previously chose the same path. This elementary ant's behavior inspired the development of ant colony optimization by Marco Dorigo in 1992, constructing a meta-heuristic stochastic combinatorial computational methodology belonging to a family of related meta-heuristic methods such as simulated annealing, Tabu search and genetic algorithms. This book covers in twenty chapters state of the art methods and applications of utilizing ant colony optimization algorithms. New methods and theory such as multi colony ant algorithm based upon a new pheromone arithmetic crossover and a repulsive operator, new findings on ant colony convergence, and a diversity of engineering and science applications from transportation, water resources, electrical and computer science disciplines are presented.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Shinichi Shirakawa, Shintaro Ogino, and Tomoharu Nagao (2011). Automatic Construction of Programs Using Dynamic Ant Programming, Ant Colony Optimization - Methods and Applications, Avi Ostfeld (Ed.), ISBN: 978-953-307-157-2, InTech, Available from: http://www.intechopen.com/books/ant-colony-optimization-methods-and-applications/automatic-construction-of-programs-using-dynamic-ant-programming

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821