Ant Colony Programming for Approximation Problems*

Mariusz Boryczka[1], Zbigniew J. Czech[2], and Wojciech Wieczorek[1]

[1] University of Silesia, Sosnowiec, Poland, {boryczka,wieczor}@us.edu.pl
[2] University of Silesia, Sosnowiec and Silesia University of Technology, Gliwice, Poland, zjc@us.edu.pl

Abstract. A method of automatic programming, called genetic programming, assumes that the desired program is found by using a genetic algorithm. We propose an idea of ant colony programming in which instead of a genetic algorithm an ant colony algorithm is applied to search for the program. The test results demonstrate that the proposed idea can be used with success to solve the approximation problems.

1 Introduction

Approximation problems which consist in a choice of an optimum function from some class of functions are considered. While solving an approximation problem by ant colony programming the desired approximating function is built as a computer program, i.e. a sequence of assignment instructions which evaluates the function.

2 Ant Colony Programming for Approximation Problems

The ant colony programming system consists of: (a) the nodes of set N of graph $G = (N, E)$ which represent the assignment instructions out of which the desired program is built; the instructions comprise the terminal symbols, i.e. constants, input and output variables, temporary variables and functions; (b) the tabu list which holds the information about the path pursued in the graph; (c) the probability of moving ant k located in node r to node s in time t which is equal to:

$$p_{rs}^{k}(t) = \frac{\tau_{rs}(t) \cdot [\psi_s]^{\beta}}{\sum\limits_{i \in J_r^k} [\tau_{ri}(t)] \cdot [\psi_i]^{\beta}}$$

Here $\psi_s = 1/e$, where e is an approximation error given by the program while expanded by the instruction represented by node $s \in N$.

* This work was carried out under the State Committee for Scientific Research (KBN) grant no 7 T11C 021 21.

3 Test Results

The genetic (GP) and ant colony programming (ACP) methods to solve approximation problems were implemented and compared on the real-valued function of three variables:

$$t = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2 \tag{1}$$

where x, y, $z \in [1.0, 6.0]$. The experiments were conducted in accordance to the learning model. Both methods were first run on a training set, T, of 216 data items, and then on a testing set, S, of 125 data items. The results of the experiments are summarized in Table 1. It can be seen (Table 1a) that the average percentage errors (e_T and e_S) for the ACP method are larger than those for the GP method. The range of this error for the training process and 100 experiments was 0.0007...9.9448 for the ACP method, and 0.0739...6.6089 for the GP method. The error 0.0007 corresponds to a perfect fit solution with respect to function (1). Such a solution was found 8 times in the series of 100 experiments by the ACP method, and was not found at all by the GP method. Table 1b compares our GP and ACP experimental results (for function (1)) with the results cited in the literature.

Table 1. (a) The Average Percentage Error, e_T, e_S, and the Standard Deviation, σ_T, σ_S, for the Training, T, and Testing, S, Data; (b) Comparison of Results

a)

Method	e_T	σ_T	e_S	σ_S
100 experiments, 15 min each				
GP	1.86	1.00	2.15	1.35
ACP	6.81	2.60	6.89	2.61
10 experiments, 1 hour each				
GP	1.07	0.58	1.18	0.60
ACP	2.60	2.17	2.70	2.28

b)

Model/method	e_T	e_S
GMDS model	4.70	5.70
ACP (this work)	2.60	2.70
Fuzzy model 1	1.50	2.10
GP (this work)	1.07	1.18
Fuzzy model 2	0.59	3.40
FNN type 1	0.84	1.22
FNN type 2	0.73	1.28
FNN type 3	0.63	1.25
M-Delta	0.72	0.74
Fuzzy INET	0.18	0.24
Fuzzy VINET	0.08	0.18

4 Conclusions

The idea of ant colony programming for solving approximation problems was proposed. The test results demonstrated that the method is effective. There are still some issues which remain to be investigated. The most important is the issue of establishing the set of instructions, N, which defines the solution space explored by the ACP method. On the one hand this set should be as small as possible so that the searching process is fast. On the other hand it should be large enough so that the large number of local minima, and hopefully the global minimum, are encountered.