Genetic Programming with Active Data Selection

Byoung-Tak Zhang and Dong-Yeon Cho

Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering
Seoul National University
Seoul 151-742, Korea
{btzhang, dycho}@scai.snu.ac.kr
http://scai.snu.ac.kr/

Abstract. Genetic programming evolves Lisp-like programs rather than fixed size linear strings. This representational power combined with generality makes genetic programming an interesting tool for automatic programming and machine learning. One weakness is the enormous time required for evolving complex programs. In this paper we present a method for accelerating evolution speed of genetic programming by active selection of fitness cases during the run. In contrast to conventional genetic programming in which all the given training data are used repeatedly, the presented method evolves programs using only a subset of given data chosen incrementally at each generation. This method is applied to the evolution of collective behaviors for multiple robotic agents. Experimental evidence supports that evolving programs on an incrementally selected subset of fitness cases can significantly reduce the fitness evaluation time without sacrificing generalization accuracy of the evolved programs.

1 Introduction

Genetic programming (GP) is a method for finding the most fit computer programs by means of artificial evolution. A population of computer programs are generated at random. They are evolved to better programs using genetic operators. The ability of the program to solve the problem is measured as its fitness value.

The genetic programs are usually represented as *trees*. A genetic tree consists of elements from a function set and a terminal set. Function symbols appear as nonterminal nodes. Terminal symbols are used to denote actions taken by the program. Since Lisp S-expressions can be represented as trees, genetic programming can, in principle, evolve any Lisp programs. Due to this powerful expressiveness, GP provides an effective method for automatic programming and machine learning.

One difficulty in genetic programming is, however, that it requires enormous computational time. The time for evolution is proportional to the product of population size, generation number, and the data size needed for fitness evaluation. Typical population size for GP ranges from a few hundreds to several thousands [4]. A typical run requires fifty to hundreds of generations. The data size depends on the application. Fitness evaluation takes the most of evolution time in GP since it requires programs to be executed against fitness cases.

In this paper we present two methods for reducing computational costs for genetic programming by evolving programs on a selected subset of given fitness cases. The idea of active data selection in supervised learning was originally introduced in 1991 by one of the authors for efficient training of neural networks [11,7,8]. Motivated by this work Gathercole *et al.* used training subsets for genetic programming [1,2]. Our approach is different from that of Gathercole *et al.* in that we increase the training set incrementally as generation goes on, rather than using the same number of fitness cases. The effectiveness of the presented methods was tested on a multiagent learning problem in which a group of mobile agents are to transport together a large table to the goal position.

The paper is organized as follows. Section 2 describes the multiagent task. Section 3 presents the genetic programming approach with active data selection. Section 4 shows experimental results. Section 5 discusses the result.

2 Evolving Multiagent Strategies Using Genetic Programming

The table transport problem that will be used in our experiments is an example of multi-robot applications [9]. In an $n \times n$ grid world, a single table and four robotic agents are placed at random positions, as shown in Figure 1. A specific location is designated as the destination. The goal of the robots is to transport the table to the destination in group motion. The robots need to move in herd since the table is too heavy and large to be transported by single robots.

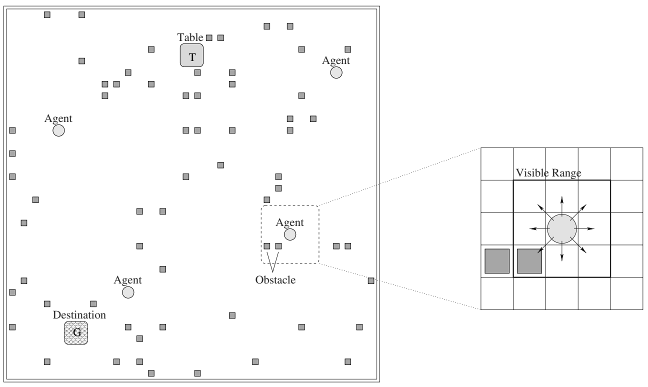

Fig. 1. The environment for multiagent learning.

Table 1. Terminals and functions of GP-trees for the table transport problem.

	Symbol	Description
Terminals	`FORWARD`	Move one step forward in the current direction
	`AVOID`	Check clockwise and make one step in the first direction that avoids collision
	`RANDOM-MOVE`	Move one step in the random direction
	`TURN-TABLE`	Make a clockwise turn to the nearest direction of the table
	`TURN-GOAL`	Make a clockwise turn to the nearest direction of the goal
	`STOP`	Stay at the same position
Functions	`IF-OBSTACLE`	Check collision with obstacles
	`IF-ROBOT`	Check collision with other robots
	`IF-TABLE`	Check if the table is nearby
	`IF-GOAL`	Check if the table is nearby
	`PROG2`, `PROG3`	Evaluate two (or three) subtrees in sequence

Each robot i ($i = 1, .., N_{robots}$) is equipped with a control program A_i. If $A_i \neq A_j$ for $i \neq j$, then control programs are said to be *private*. In case of *public* control programs, all instances of A_i are constrained to be the same A.

The robots activate A_i's in parallel to run a team trial. At the beginning of the trial, the robot locations are chosen at random in the arena. They have different positions and orientations. During a trial, each robot is are granted a total of S_{max} elementary movements. The robot is allowed to stop in less than S_{max} steps if it reaches the goal. At the end of the trial, each robot i gets a fitness value which was measured by summing the contributions from various factors.

The objective of a GP run is to find a multi-robot algorithm that, when executed by the robots in parallel, causes efficient table transport behavior in group. The terminal and function symbols used for GP to solve this problem are listed in Table 1. The terminal set consists of six primitive actions: `FORWARD`, `AVOID`, `RANDOM-MOVE`, `TURN-TABLE`, `TURN-GOAL` and `STOP`. The function set consists of six primitives: `IF-OBSTACLE`, `IF-ROBOT`, `IF-TABLE`, `IF-GOAL`, `PROG2` and `PROG3`. Each fitness case represents a world of 32 by 32 grid on which there are four robots, 64 obstacles, and the table to be transported. A set of training cases are used for evolving the programs.

All the robots use the same control program. To evaluate the fitness of robots, we made a complete run of the program for one robot before the fitness of another is measured. The fitness value, $f_{ij}(g)$, of individual i at generation g against case j is computed by considering various penalty factors. These include the distance between the target and the robot, the number of steps moved by the robot, the number of collisions made by the robot, the distance between starting and final position of the robot, and the penalty for moving away from other robots. More details can be found elsewhere [9].

The fitness, $F_i(g)$, of program i at generation g is measured as the average of its fitness values $f_{ij}(g)$ for the cases j in the training set:

$$F_i(g) = \frac{1}{S} \overset{S}{\underset{j=1}{}} f_{ij}(g) \tag{1}$$

where S is the number of fitness cases.

In the following section we present the active data selection method for genetic programming.

3 Genetic Programming with Incremental Data Selection

With each program is associated a small set of initial training cases of size n_0, chosen from the base training set $D^{(N)}$ of size N. Individuals are evolved by the usual genetic programming. In addition, the algorithm has an additional step, i.e. incremental data inheritance (IDI), in which data sets are evolved.

For the initial data population, a small subset of fitness cases, $D(0)$, is chosen from the base training set $D^{(N)}$ of size N:

$$D(0) \subset D^{(N)}, \quad |D(0)| = n_0. \tag{2}$$

After individuals are evolved by the usual evolutionary process (fitness evaluation, selection, and mating to generate offsprings), a portion of training set, $\Delta(g)$, is chosen from the previous candidate set $C(g-1)$

$$\Delta(g) \subset C(g-1), \quad |\Delta(g)| = \lambda, \tag{3}$$

where $C(g-1) = D^{(N)} - D(g-1)$. And it is mixed with the previous training set to make a new training set $D(g)$ for the next generation

$$D(g) = D(g-1) \cup \Delta(g), \quad D(g-1) \cap \Delta(g) = \{\}. \tag{4}$$

That is, the sequence of training sets for GP active is

$$D(0) \subset D(1) \subset D(2) \subset ... \subset D(G) = D^{(N)}, \tag{5}$$

where G is the number of maximum generation.

We use a variant of uniform crossover to produce offspring data from their parent data. Two parent data sets, $D_i(g)$ and $D_j(g)$, are crossed to inherit their subsets to two offspring data sets, $D_i(g+1)$ and $D_j(g+1)$. In uniform data crossover, the data of parents' are mixed into a union set

$$D_{i+j}(g) = D_i(g) \cup D_j(g), \tag{6}$$

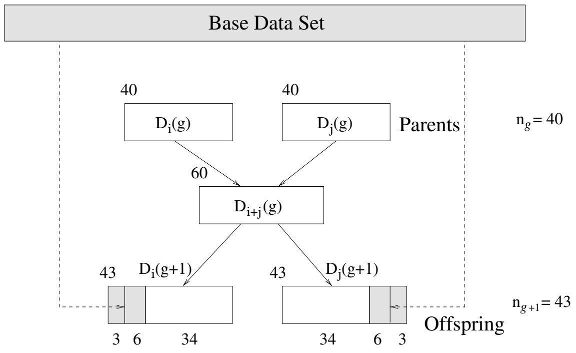

Fig. 2. Uniform data crossover for data inheritance.

which are then redistributed to two offspring:

$$\begin{aligned} D_i(g+1) &\subset D_{i+j}(g) \\ D_j(g+1) &\subset D_{i+j}(g) \end{aligned} \tag{7}$$

where the size of offspring data sets are equal to $n_{g+1} = n_g + \lambda$, where $\lambda \geq 1$ is the data increment size.

To ensure performance improvement, it is important to maintain the diversity of the training data as generation goes on. The diversity of data set $D_i(g)$ is measured by the ratio of distinctive examples:

$$d_i = \frac{|D_{i+j}(g)|}{|D_i(g)|} - 1, \qquad 0 \leq d_i \leq 1 \tag{8}$$

where $d_i = 0$ if the parents have the same data and $d_i = 1$ if parents have no common training examples. To maintain the diversity, a portion ρ of the diversity factor d_i is used to import examples from the base data set.

$$r_i = \rho \cdot (1 - d_i), \qquad 0 \leq \rho \leq 1. \tag{9}$$

For example, assume that the current parents have data sets, $D_i(g)$ and $D_j(g)$, of size $n_g = 40$ each and $|D_{i+j}(g)| = 60$. Let the parameters be $\rho = 0.3$, $\lambda = 3$. Then, we need to generate two training sets of size $n_{g+1} = n_g + \lambda = 43$ for the offspring (Figure 2). The diversity is $d_i = \frac{|D_{i+j}(g)|}{|D_i(g)|} - 1 = 1.5 - 1 = 0.5$ and the import rate is $r_i = \rho \cdot (1 - d_i) = 0.3 \cdot (1 - 0.5) = 0.15$. The data for each offspring is generated by randomly choosing 34 examples from $D_{i+j}(g)$, 6 examples from $D^{(N)}$ and again $\lambda = 3$ examples from $D^{(N)}$. Figure 2 illustrates this process.

4 Experimental Results

Experiments have been performed using the parameter values listed in Table 2. The terminal set and function set consist of six primitives, respectively, as summarized in Table 1. A total of 100 training cases were used for evolving the programs for standard GP runs. GP runs with active data selection used $10+3g$ examples out of the given data set, i.e. $n_0 = 10, \lambda = 3$, for fitness evaluation. For all methods, a total of 100 independent worlds were used for evaluating the generalization performance of evolved programs.

We compared the performance of the GP with active data selection to the GP with random data selection. Results are shown in Figure 3. GPs with IDI and incremental random selection (IRS) achieved better than GP without active selection (GP standard). Figure 4 shows the fitness of three methods with repect to the total number of evaluations. Since the GP with active data selection uses variable data size, we calculated the number of evaluations at generation g by a product of the population size and the data size at generation g. The active GP methods achieved a speed-up factor of approximately two compared with that of the standard GP. The results are summarized in Table 3. Though the GP with active data selection methods used a smaller set of fitness cases, its training and test performance were slightly better than those of the standard GP. Though further experiments are necessary for more definite conclusion, it seems that the active GP has a potential to evolve smaller programs than the standard GP since small data usually tends to require smaller programs. This seems interesting from the Occam's razor principle point of view [10,6].

Table 2. Parameters used in the experiments.

Parameter	Value
Population size	100
Max generation	30
Crossover rate	0.9
Mutation rate	0.1

Table 3. Comparison of time and average fitness values (lower is better) for the standard GP and the GP with active data selection. The values are averaged over ten runs. Also shown are the standard deviation.

Method	Time	Average Fitness	
		Training	Test
GP standard	300000	211.21 ± 9.19	225.64 ± 12.05
GP with IRS	170500	209.60 ± 7.67	219.91 ± 13.11
GP with IDI	170500	195.97 ± 8.41	203.39 ± 10.78

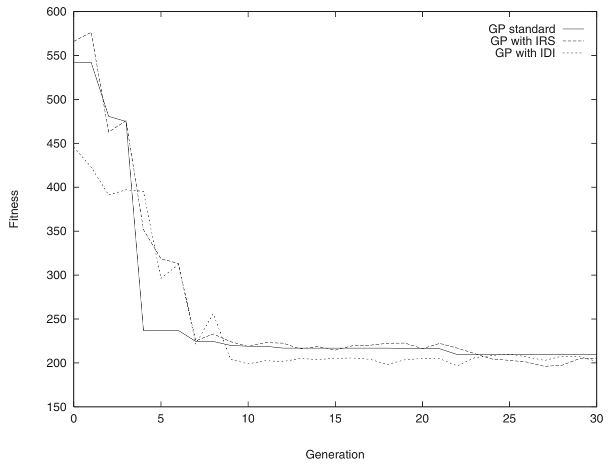

Fig. 3. Comparison of fitness values as a function of generation number.

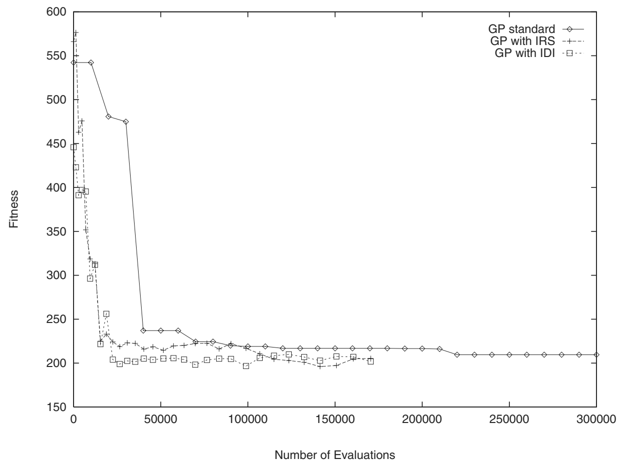

Fig. 4. Comparison of fitness values as a function of the number of function evaluations.

5 Conclusions

We have presented a method for accelerating evolution speed of genetic programming by selecting a subset of given fitness cases. Since the fitness evaluation step is a bottleneck in GP computing time, this method can make an essential contribution to improving the GP performance.

Experimental results have shown that by reducing the fitness cases the evolution speed of GP can be enhanced without loss of generality of the evolved programs. This is especially true for problem settings in which a large amount of fitness cases are available. In this case, the active data selection can exploit the redundancy in the data, while the standard GP blindly re-evaluates all the fitness cases.

Acknowledgements

This research was supported in part by the Korea Science and Engineering Foundation (KOSEF) under grants 96-0102-13-01-3 and 981-0920-350-2.

References

1. Gathercole, C. and Ross, P. 1994. Dynamic training subset selection for supervised learning in genetic programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, (eds.). *Parallel Problem Solving from Nature III*, Berlin: Springer-Verlag, Pages 312-321.
2. Gathercole, C. and Ross, P. 1997. Small populations over many generations can beat large populations over few generations in genetic programming. In J.R. Koza (eds.). *Genetic Programming 1997.* Cambridge, MA: The MIT Press. Pages 111-118.
3. Haynes, T., Sen, S., Schoenefeld, D., and Wainwright, R. 1995. Evolving a team, In *Proc. AAAI-95 Fall Symposium on Genetic Programming* AAAI Press. Pages 23-30.
4. Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: The MIT Press.
5. Luke, S. and Spector, L. 1996. Evolving teamwork and coordination with genetic programming. In J.R. Koza (eds.). *Proc. First Genetic Programming Conf.* Cambridge, MA: The MIT Press. Pages 150-156.
6. Soule, T., Foster, J. A., and Dickinson, J. 1996. Code growth in genetic programming. In J.R. Koza (eds.). *Genetic Programming 1996.* Cambridge, MA: The MIT Press. Pages 215-223.
7. Zhang, B. T. 1992. *Learning by Genetic Neural Evolution*, DISKI Vol. 16, 268 pages, ISBN 3-929037-16-6, Infix-Verlag, St. Augustin/Bonn.
8. Zhang, B. T. 1994. Accelerated learning by active example selection, *International Journal of Neural Systems*, 5(1): 67-75.
9. Zhang, B. T. and Cho, D. Y. 1998. Fitness switching: Evolving complex group behaviors using genetic programming. In *Genetic Programming 1998*, Madison, Wisconsin, pp. 431-438, 1998.
10. Zhang, B. T. Mühlenbein, H. 1995. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation.* 3(1) 17-38.
11. Zhang, B. T. and Veenker, G. 1991. Focused incremental learning for improved generalization with reduced training sets, *Proc. Int. Conf. Artificial Neural Networks*, Kohonen, T. *et al.* (eds.) North-Holland, pp. 227-232.