Dynamical Teams of Genetic Predictors

Defoin Platel Michael [1] and **Clergue Manuel** [2] and **Collard Philippe** [2]

Abstract. Genetic Programming (GP) has been shown to be a good method of predicting functions. In this context, a solution given by GP generally consists of a sole predictor. In contrast, Stack-based GP systems manipulate structures containing several predictors, which can be considered as *teams of predictors*. Work in Machine Learning reports that combining predictors gives good results in terms of both quality and robustness. In this paper, we use Stack-based GP to study different linear cooperations between predictors. Preliminary tests and parameter tuning are performed on a GP benchmark. A comparative study with standard methods has shown limits and advantages of teams prediction, leading to encourage the use of combinations taking into account the response quality of each team member.

Introduction

The emergence of GP in the scientific community arose with the use of a tree-based representation. However, there are GP systems manipulating linear structures, which have shown experimental performances equivalent to Tree GP (TGP) [1]. In contrast to TGP, Linear GP (LGP) programs are sequences of instructions of an imperative language (C, machine code, ...). There are at least two kind of LGP implementation. In the first one [1], a finite number of registers are used to store the partial computations and a particular register is chosen to store the final result. In the other one, the stack-based implementation [3], the intermediate computations are pushed into an operand stack and the top of stack gives the final result. It is important to note that in TGP, an individual corresponds to a sole program and its evaluation produces a unique output. In LGP, an individual may be composed of many independent sub-programs and its evaluation may produce several outputs and some of them may be ignored in the final result. In this paper, we propose to combine those sub-programs into a team of predictors.

The goal of Machine Learning (ML) is to find a predictor trained on set of examples that can approximate the function that generated the examples. Over-fitting occurs when a predictor reflects randomness in the data rather than underlying function properties, and so it often leads to poor generalization abilities of predictors. Several methods have been proposed to avoid over-fitting, such as model selection, to stop training or combining predictors, see [5] for a complete discussion. In this study, we mainly focus on combining predictor methods, also called ensemble, or committee methods. In a committee machine, a team of predictors is generated by means of a learning process and the overall predictions of the committee machine is the combination of the predictions of the individual team members. In the GP field, there has been some work on the combination of predictors, see for examples [6][2].

[1] ACRI-ST, Sophia Antipolis, France
[2] Laboratoire I3S, UNSA-CNRS, Sophia Antipolis, France

In this paper, we take advantage of the stack-based GP implementation by evolving teams of predictors. The originality of this study is that teams have a dynamic number of members that can be managed by the system.

1 Teams of Genetic Predictors

In stack-based GP, numerical calculations are performed in *Reverse Polish Notation*. According to the implementation proposed by Perkis[3], an additional type of closure constraint is imposed on functions: they are defined to do nothing when arity is unsatisfied by the current state of the operand stack. In Figure 1, a basic example of program execution is presented. We can see the processing of the

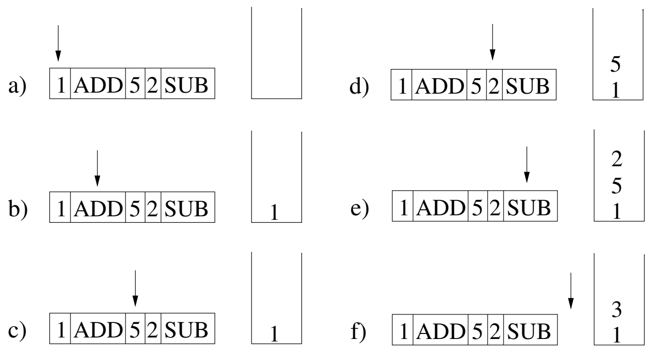

Figure 1. Evaluation of a program in Stack-based GP.

program "1,ADD,5,2,SUB". Initialization phase corresponds to step *a* where the stack is cleared. During steps *b*, *d* and *e*, numerical constants 1, 5 and 2, respectively, are pushed onto the operand stack. During step *f*, since the operand stack stores enough data (at least two), the "SUB" instruction is computed and the result ($3 = 5 - 2$) is pushed onto the stack. During step *c* the stack contains only one value, so computation of the "ADD" instruction is impossible: the instruction is simply skipped with no effect on the operand stack.

Let us note that after step *f*, the operand stack contains the values 1 and 3. This means that there are two independent sub-programs, "1" and "5,2,SUB", in the sequence "1,ADD,5,2,SUB". We propose to make the whole set of sub-programs involved in the fitness evaluation. The idea is to combine the results of each sub-programs, i.e. all the elements of the final stack, according to the nature of the problem addressed. Using this kind of evaluation, the evolutionary process should be able to tune the effect of the genetic operators by changing the number and the nature of sub-programs and so modifying the contribution of each of them to the final fitness. This approach may be viewed as the evolution of teams of predictors corresponding to the combination of sub-programs.

Let us consider a program with m instructions giving a team T having $k \in [1, m]$ predictors p_i. The team output $O(T)$ consists of a combination of the k predictors outputs $o(p_i)$. Several combinations have been tested :

- Sum of each predictor output (Sum),

$$O(T) = \sum_{i=1}^{k} o(p_i)$$

- Arithmetic mean of predictors outputs (AMean),

$$O(T) = \frac{1}{k} \sum_{i=1}^{k} o(p_i)$$

- Product of each predictor output (Pi),

$$O(T) = \prod_{i=1}^{k} o(p_i)$$

- Geometric mean of each predictor output (GMean),

$$O(T) = \sqrt[k]{\prod_{i=1}^{k} o(p_i)}$$

- Mean of each predictor output weighted by their error (EMean),

$$O(T) = \frac{1}{\sum_{i=1}^{k} w_i} \sum_{i=1}^{k} w_i o(p_i)$$

  with $w_i = e^{-\beta E(p_i)}$, β a positive scaling factor and $E(p_i)$ the training error of p_i
- Winner predictor output Takes All (WTA),

$$O(T) = o(argmin_{p_i, i \in [1,k]}(E(p_i)))$$

  with $E(p_i)$ the training error of p_i
- Top of stack (Top), this is the classical way to evaluate a program where only output of the predictor at the top of the stack are taken into account.

$$O(T) = o(p_k)$$

2 Experimental Results

We choose the Poly 10 problem [4], where the target function is the 10-variate cubic polynomial $x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$. In this study, the fitness is the classical Root Mean-Square Error. The dataset contains 50 test points and is generated by randomly assigning values to the variables x_i in the range $[-1, 1]$. We perform 50 independent runs with various mutation and crossover rates. Populations of 500 individuals are randomly created according to a maximum creation size of 50. The instructions set contains: the four arithmetic instructions ADD, SUB, MUL, DIV, the ten variables $X_1 \ldots X_{10}$ and one stack-based GP specific instruction DUP which duplicates the top of the operand stack. The evolution, with elitism, maximum program size of 500, 16-tournament selection, and steady-state replacement, takes place over 100 generations [3]. We use a statistical unpaired, two-tailed t-test with 95% confidence to determine if results are significantly different.

In Table 1, the best performances on the Poly 10 problem with different combinations of predictors are presented, using the best settings of evolutionary parameters found (crossover rate varying from 0 to 1.0 and mutation rate from 0 to 2.0). In the first row, results obtained using a Tree GP implementation are reported in order to give an absolute reference. We see that the classical Top of stack and WTA methods work badly, which is to be expected, since in this case, teams' outputs correspond to single predictor outputs. The Sum and AMean methods report good results compared to the Tree method. In contrast, the Pi and GMean methods are not suitable for this problem, perhaps because the Poly 10 problem is easiest to decompose as a sum. Finally, EMean undoubtedly outperforms other methods. We note that results presented here correspond to a scaling factor β of 10.

[3] In a steady state system, the generation concept is somewhat artificial and is used only for comparison with generational systems. Here, a generation corresponds to a number of replacement equal to the number of individual in the population, i.e. 500.

Table 1. Results on Poly 10.

Team	Mean	Std Dev	Best	Worst
Tree	0.250	0.072	0.085	0.407
Top	0.353	0.105	0.172	0.520
WTA	0.443	0.048	0.347	0.541
Sum	0.173	0.031	0.120	0.258
AMean	0.165	0.034	0.109	0.251
Pi	0.361	0.063	0.220	0.456
GMean	0.222	0.027	0.151	0.301
EMean	**0.066**	**0.017**	**0.029**	**0.141**

Conclusion

In this paper, we start with a known drawback of stack-based GP systems : they provide many outputs, that is sub-programs, instead of only one. Taking the arithmetic mean or the sum of sub-programs outputs gives our system better performances than a tree-based GP system, which evolves individual predictors. The best results we obtain are when the output of the program is a weighted sum of the sub-programs, where each sub-program receives a weight depending on its individual performance. Moreover, this supplementary degree of freedom promotes the emergence of dynamically sized teams.

This paper represents the first step of our work and empirical results should be confirmed by experiments on other problems and theoretical studies. Moreover, some choices we have made are arbitrary, such as the use of a linear combination of sub-programs. Other types of combinations, e.g. logarithmic ones may improve performances. Also, we need to better understand how the scaling factor β influences performance, and its effect on team composition.

REFERENCES

[1] Markus Brameier and Wolfgang Banzhaf, 'A comparison of linear genetic programming and neural networks in medical data mining', *IEEE Transactions on Evolutionary Computation*, **5**(1), 17–26, (2001).

[2] Markus Brameier and Wolfgang Banzhaf, 'Evolving teams of predictors with linear genetic programming', *Genetic Programming and Evolvable Machines*, **2**(4), 381–407, (2001).

[3] Tim Perkis, 'Stack-based genetic programming', in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pp. 148–153, Orlando, Florida, USA, (27-29 1994). IEEE Press.

[4] Riccardo Poli, 'A simple but theoretically-motivated method to control bloat in genetic programming', in *Genetic Programming, Proceedings of EuroGP'2003*, eds., Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, volume 2610 of *LNCS*, pp. 200–210, Essex, (14-16 April 2003). Springer-Verlag.

[5] W. Sarle. Stopped training and other remedies for overfitting, 1995.

[6] Byoung-Tak Zhang and Je-Gun Joung, 'Time series prediction using committee machines of evolutionary neural trees', in *Proceedings of the Congress of Evolutionary Computation*, eds., Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, volume 1, pp. 281–286. IEEE Press, (1999).