Semantic Fitness Function in Genetic Programming Based on Semantics Flow Analysis*

Pak-Kan Wong
The Chinese University of Hong Kong
Hong Kong
pkwong@cse.cuhk.edu.hk

Man-Leung Wong
Lingnan University
Hong Kong
mlwong@ln.edu.hk

Kwong-Sak Leung
The Chinese University of Hong Kong
Hong Kong
ksleung@cse.cuhk.edu.hk

ABSTRACT

The search performance of conventional Genetic Programming (GP) methods is strongly guided by the performance of the fitness function. In each generation, the fitness function evaluates every program in the population and measures the distance between the final output of the programs and the desired output. Human programmers often rely on the feedback from the intermediate execution states, which are the semantics, to localize and resolve software bugs. However, the semantics of a program is seldom explicitly considered in the fitness function to assess the quality of a program in GP. In this paper, we invent methods to improve fitness evaluation leveraging semantics in GP. We propose semantics flow analysis for programs using information theoretic concepts. Next, we develop a novel semantic fitness evaluation technique to rank programs using semantics based on the semantics flow analysis. Our evaluation results show that adopting our method can improve the success rates in Grammar-Based GP.

CCS CONCEPTS

• **Software and its engineering** → **Genetic programming**; • **Computing methodologies** → *Neural networks*;

KEYWORDS

Genetic Programming, Semantics, Fitness, Semantics Flow

ACM Reference Format:
Pak-Kan Wong, Man-Leung Wong, and Kwong-Sak Leung. 2019. Semantic Fitness Function in Genetic Programming Based on Semantics Flow Analysis. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3319619.3321960

1 INTRODUCTION

Conventional Genetic Programming (GP) and Grammar-Based Genetic Programming (GBGP) methods search on the syntactical structure of a program [2, 5, 6]. The semantics of a program involves the Intermediate Execution States (IESs), which are the values computed by the subtrees of a program tree and are sometimes called program traces, and concerns the meaning of the program execution. Conventional methods ignore the semantics of the program. The first analytical study on semantics in the context of GP was conducted by Langdon [4] and more recently by Krzysztof [3]. It is observed that human programmers often rely on the feedback from the IESs and then they can effectively localize the bugs and assess the quality of different parts of the programs. In this paper, we show that the IESs of a program can be divided into two groups: those can be found in an optimal program and those cannot. As such, program structures associated with the former group may be more reusable during evolution than those associated with the latter group. Programs composed of more reusable program structures should be given a higher chance, which can be controlled by the fitness function, to breed new programs in the next population. Therefore, we propose a method to utilize the semantic information to rank the programs.

The paper is organized as follows. First of all, we introduce the concept of semantics flow and present a brief description of semantics flow analysis in Section 2. Our semantic fitness function is presented in Section 3. Experiments are briefly presented in Section 4. Conclusion and future work are discussed in Section 5.

2 SEMANTICS FLOW IN A PROGRAM

To understand how a program works, it is often useful to inspect the IESs which are called the semantics. They are stored in the memory during program execution. This is essential to study how the semantics interact. Let us focus on the candidate programs in GP that can be executed sequentially. Initially, the inputs to a program are the data stored in the memory of a computer as shown in Figure 1. We call the data as the *input semantics* of the program. A program instructs the computer how to transform the data stored in the memory at each step. The program in Figure 1 needs transform the input semantics (i.e. x_0, x_1, x_2, x_3, and x_4) five times by applying *and* functions and *or* functions twice. Immediately after each step of the function execution, the IES stored in the memory is the semantics of the step. By construction, the programs generated by GP will stop after a finite number of steps for parse trees with a limited height. The final output of the program is the data in the memory after the completion of all steps The data is called the *output semantics*.

In order to track the changes in semantics during program execution, we formally represent all semantics appeared in the memory using a set of tuples. Denote the semantics at step k for a program p as $s_p(k)$, where $s_p(0)$ is a tuple storing the initial inputs, i.e. the input semantics. Suppose the program p terminates after

*Produces the permission block, and copyright information

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3321960

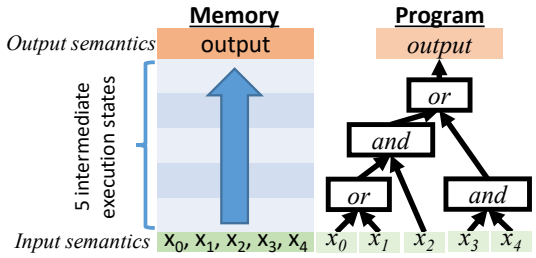

Figure 1: Illustration of the formulation of semantics flow.

step K, the semantics at step $s_p(K)$, is the output semantics of the program. A *semantics* flow of this program can be denoted by $s_p(0) \rightarrow s_p(1) \rightarrow \ldots \rightarrow s_p(K)$.

3 SEMANTICS FITNESS FUNCTION

Given that a program is defective, some parts of it should not be included or the way of composing these components should be wrong. We propose $SemanticsCompare(p_1, p_2)$ (Algorithm 1) which is a semantic fitness function to compare the programs. Algorithm 1 is based on two objectives:

1) Optimal Program Execution. It can be proved that the semantics flow of the optimal program is constrained which can be exploited during evolution. Let T be the random variable for the desired output semantics and is uniquely defined by all given test cases. As such, the random variable $S_{opt}(K)$ is equal to the random variable T given that the program *opt* is an optimal program.

Corollary 1. Optimal program execution inequality. For any semantics flow of any optimal program *opt*, and the random variable T corresponds to the target output semantics, we have $I(S_{opt}(i); T) \leq I(S_{opt}(j); T) \leq I(T; T) \leq H(T)$, where $i \leq j$ and $I(\cdot;\cdot)$ is the mutual information function.

The proof applies data processing inequality [1]. Moreover, the smallest i satisfying Corollary 1 is called a *critical step*. Programs with a critical step are more preferable than those without.

2) Parsimony Principle of Program Size. We want to keep the size of the potentially correct part of the program to be small. To measure the program complexity of this part of the program, we consider the following quantity: $\Delta B_{critical}(p_1, p_2) = B_{p_1}(i_{critical,p_1}) - B_{p_2}(i_{critical,p_2})$, where $B_p(i)$ is the total number of branches of p involved in the semantics $s_p(i)$. If the total number of branches of p_1 is smaller than that of p_2, it is more likely that p_1 will use fewer number of steps to achieve the target outcome than that p_2 will use. In addition, we define $V_p(i)$ as the total number of nodes in the branches involved in the i-th step of a program p. $V_p(i)$ is an alternative measure of program complexity.

4 EVALUATION

Due to page limit, we present a part of our results about using GP with our semantics fitness function to solve parity problems. The semantics fitness function and the canonical fitness function (CFF), i.e. using the total number of correct cases as the fitness value, were executed independently for 30 runs per problem. Our results show that the success rates of the semantics fitness function and CFF were respectively 93% and 90% for the parity 5 problem. For the parity 6 problem, those values were 50% and 47% respectively. To summarize, the success rates of the semantics fitness function were 3% higher than those values of CFF in these problems.

Algorithm 1 SemanticsCompare(p_1,p_2)

Input: program p_1 and program p_2
Output: a Boolean variable $better$ (i.e. p_1 ranks higher than p_2).

if $B_{p_1}(i_{critical,p_1}) \neq 0 \wedge B_{p_2}(i_{critical,p_2}) == 0$ **then**
  $better$ = TRUE
  return $better$
end if
if $B_{p_2}(i_{critical,p_2}) \neq 0 \wedge B_{p_1}(i_{critical,p_1}) == 0$ **then**
  $better$ = FALSE
  return $better$
end if
if $p_1.fitness \neq p_2.fitness$ **then**
  $better = p_1.fitness > p_2.fitness$
  return $better$
end if
if $\Delta B_{critical}(p_1, p_2) \neq 0$ **then**
  $better = \Delta B_{critical}(p_1, p_2) < 0$
  return $better$
end if
$better = V_{p_1}(i_{critical,p_1}) < V_{p_2}(i_{critical,p_2})$
return $better$

5 CONCLUSION AND FUTURE WORK

This paper proposes the concept of semantics flow in GP and designs a fitness function for a set of defective programs based on their semantic flows. Our method does not introduce extra parameters in GP framework. Our experimental results show that our method can attain a higher success rate than the canonical fitness function can achieve. One shortcoming of our method is that more rounds of fitness evaluation may be needed.

In the future, multi-objective optimization can be applied so as to minimize the program errors at the final output and the seriousness of the program errors within the defect programs. Apart from Boolean functions, our method will be generalized to a function set and a terminal set of countable and finite sets. Lastly, we believe that the theory of semantics flow will lead to innovation of semantic crossover and semantic mutation.

ACKNOWLEDGMENTS

This research is supported by Institute of Future Cities of The Chinese University of Hong Kong.

REFERENCES

[1] Thomas M Cover and Joy A Thomas. 2012. *Elements of information theory.* John Wiley & Sons.
[2] John R Koza. 1992. *Genetic Programming: vol. 1, On the programming of computers by means of natural selection.* Vol. 1. MIT press.
[3] Krzysztof Krawiec. 2016. *Behavioral Program Synthesis with Genetic Programming.* Vol. 618. Springer.
[4] William B Langdon. 2002. How many Good Programs are there? How Long are they?. In *Proceedings of the 2002 Foundations of Genetic Algorithms.* 183–202.
[5] Peter Alexander Whigham. 1995. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-world Applications*, Vol. 16. 33–41.
[6] Man Leung Wong and Kwong Sak Leung. 1995. Applying Logic Grammars to Induce Sub-Functions in Genetic Programming. In *Proceedings of the 1995 IEEE Conference on Evolutionary Computation*, Vol. 2. IEEE, 737–740.