Exploiting Subprograms in Genetic Programming

Steven Fine
Massachusetts Institution of Technology, USA
sfine@mit.edu

Erik Hemberg
Massachusetts Institute of Technology, USA
hembergerik@csail.mit.edu

Una-May O'Reilly
Massachusetts Institute of Technology, USA
unamay@csail.mit.edu

ABSTRACT

KEYWORDS

Keyword 1, Keyword 2

ACM Reference format:
Steven Fine, Erik Hemberg, and Una-May O'Reilly. 2017. Exploiting Subprograms in Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference 2017, Berlin, Germany, July 15–19, 2017 (GECCO '17),* 1 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Our general goal is to improve the level of program complexity that genetic programming, GP, can routinely evolve. This is toward fulfilling its potential to occupy a significant niche in the ever advancing field of program synthesis. Behavioral genetic programming (BGP) is an extension to GP that advances toward this compelling goal[1, 2]. The intuition of the BGP paradigm is that, during evolutionary search and optimization, information characterizing programs by behavioral properties that extend beyond how accurately they match their target outputs can be effectively integrated into extensions of the algorithm's fundamental mechanisms of fitness-based selection and genetic variation.

To identify useful subprograms BGP commonly exploits program *trace* information (first introduced in TODO:[**?**]) which is a capture of the output of every subprogram within the program for every test data point during fitness evaluation. The trace is stored in a matrix where the number of rows is equal to the number of test suite data points, and the number of columns is equal to the number of subtrees in a given program. The trace captures a full snapshot of all of the intermediate states of the program evaluation.

With the trace BGP next identifies the merit of each subprogram by treating its column as a feature or explanatory variable in a *model regression* on the desired program outputs. The accuracy and complexity of the model reveals how useful the subprograms are. The model, if it has feature selection capability, also reveals specific subprograms within the tree that are partially contributing to the program's fitness. BGP uses this information in two ways. It can integrate *model error* and *model complexity* into the program's fitness. Second, it maintains an archive of the most useful subtrees identified by modeling each program and uses them in an *archive-based crossover*. BGP has a number of variants that collectively yield impressive results, see [1, 2]. It demonstrates that we should look at how a program behaves during execution because partial behavior, if it can be identified and isolated, can be used to direct GP toward better program synthesis.

GECCO '17, Berlin, Germany

DOI: 10.1145/nnnnnnn.nnnnnnn

REFERENCES

[1] Krzysztof Krawiec. 2015. *Behavioral Program Synthesis with Genetic Programming.* Studies in Computational Intelligence, Vol. 618. Springer International Publishing. DOI:http://dx.doi.org/doi:10.1007/978-3-319-27565-9
[2] Krzysztof Krawiec and Una-May O'Reilly. 2014. Behavioral programming: a broader and more detailed take on semantic GP. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation.* ACM, 935–942.