Counterexample-Driven Genetic Programming without Formal Specifications

Thomas Helmuth
Hamilton College
Clinton, New York, USA
thelmuth@hamilton.edu

Lee Spector
Amherst College, Hampshire College,
and U. Massachusetts, Amherst
Amherst, Massachusetts, USA
lspector@amherst.edu

Edward Pantridge
Swoop
Cambridge, Massachusetts, USA
ed@swoop.com

ABSTRACT

Counterexample-driven genetic programming (CDGP) uses specifications provided as formal constraints in order to generate the training cases used to evaluate the evolving programs. It has also been extended to combine formal constraints and user-provided training data to solve symbolic regression problems. Here we show how the ideas underlying CDGP can also be applied using only user-provided training data, without formal specifications. We demonstrate the application of this method, called "informal CDGP," to software synthesis problems.

CCS CONCEPTS

• **Computing methodologies → Genetic programming**;

KEYWORDS

genetic programming, program synthesis, counterexamples

ACM Reference Format:
Thomas Helmuth, Lee Spector, and Edward Pantridge. 2020. Counterexample-Driven Genetic Programming without Formal Specifications. In *Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3377929.3389983

1 INTRODUCTION

The bulk of the computational effort required for genetic programming is expended in the evaluation of the errors of programs in the evolving population. Typically, each program is evaluated on many inputs, which are generally referred to as "fitness cases" or "training cases." In most prior work, the set of fitness cases that will be used for program evaluation during evolution is specified in advance of the genetic programming run, and all available cases are used for each program evaluation.

A recent method [1, 2, 5, 6] for decreasing the number of cases needed to evaluate each individual generates new cases on the basis of formal specifications for the problem that genetic programming is being employed to solve, that are not yet correctly handled by the programs in the population. These "counterexamples" provide more focused guidance to the evolutionary process than do random test cases, and appear to direct evolution more specifically to master aspects of the target problem that are not properly handled by individuals in the current population. This approach is known as "counterexample-driven genetic programming," or CDGP.

In this paper we describe a new method, inspired by CDGP, that does not require formal specifications. The approach that we describe, "informal CDGP," (iCDGP) evaluates individuals during evolution using only a small sub-sample of the user-provided fitness cases, allowing more individuals to be assessed within the same computational budget. The fitness cases that are used are not chosen randomly, but rather are chosen to be counterexamples for the best individuals in the current population. This allows iCDGP to direct evolution in much the same way as CDGP, but without requiring that the user provide formal specifications for solutions to the target problem. Since we do not have formal specifications, we instead expect the problem to be defined by a full training set of input/output examples, typically numbering 100 or more, which we call T.

In informal CDGP, we use an active training set, $T_A \subseteq T$, that GP uses to evaluate the individuals in the population. In all of our experiments, T_A initially contains 10 random training cases from T, although other sizes could be used. During evolution, if an individual is found that passes all of the cases in T_A, we test the individual on all of the cases in T; if it also passes all of them, then it is a training set solution and GP terminates. Otherwise, we select a random case in T that the individual does not pass, add it to T_A, and continue evolution. Note that if multiple individuals in a generation pass all of the cases in T_A, each of them goes through this process, potentially adding multiple new cases to T_A for the next generation.

Given that we already have a set of training cases, why does informal CDGP use a smaller, likely less-informative set of active training cases instead of just using all available training data? As with other approaches based on the sub-sampling of fitness cases ([3]), a smaller active training set allows us to perform fewer program executions per generation, making each generation computationally cheaper than if using the full set of training cases. In our experiments, we compare methods based on the same maximum number of program executions, allowing informal CDGP to run for more generations than standard GP while using the same total program executions. Additionally, the informal CDGP idea of adding a counterexample case to T_A that the best individual doesn't pass (borrowed from formal CDGP) allows it to augment the training set in ways that specifically direct GP to solve difficult parts of the problem.

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7127-8/20/07.
https://doi.org/10.1145/3377929.3389983

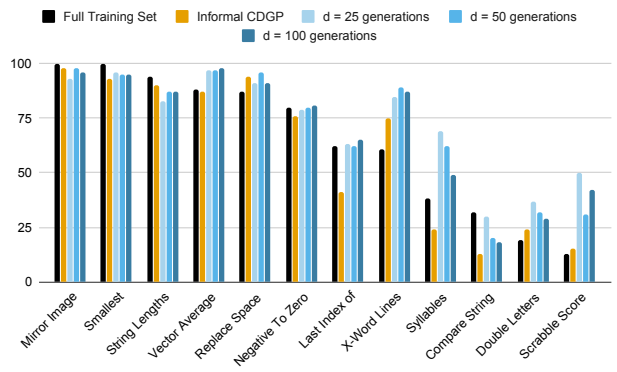

Figure 1: Number of successes out of 100 GP runs for three settings of the number of generations d between case additions in generational-based informal CDGP.

2 EXPERIMENT

We conducted experiments using the PushGP genetic programming system [7] and problems from the "General Program Synthesis Benchmark Suite" [4].

We explored several variants of iCDGP, each of which provided a different approach to the question of when new cases should be added to T_A *aside* from the condition that normally triggers case additions, in which an individual is found that passes all of the cases currently in T_A. If no progress is being made on the current set of cases for some time, then we would like to add new cases nonetheless, in the hope that the augmented set of cases will provide more guidance to the search. There are several options for when and how such supplementary additions might be made.

Of the many variants of iCDGP we tried, the most promising one uses generation-based additions, where we add a new case to T_A every time d generations have passed without a new case being added otherwise; this case is one at which the population performs worst. We tried three settings for d: 25, 50, and 100 generations. Note that failed informal CDGP runs often finished after 1000 to 3000 generations, depending on how many cases are added to T_A. We present results out of 100 runs using generation-based addition with iCDGP in Figure 1. This shows that on many problems, especially more difficult problems that are solved less often overall, iCDGP with generation-based additions significantly outperformed using the full training set as well as plain iCDGP.

3 CONCLUSIONS AND FUTURE WORK

We conclude that informal counterexample-driven genetic programming (informal CDGP) advances the state of the art for software synthesis by genetic programming. The results documented here are better, as far as we know, than any previously published for this set of benchmark program synthesis problems. The informal CDGP approach builds on the recent advance provided by formal CDGP, but it is likely to be more widely applicable because it does not require a formal specification of solutions to the target problem. The same set of test inputs that would be used for traditional genetic programming can be used for informal CDGP, with the only difference being that they will be used differently. Specifically, informal CDGP begins with a small initial subset of the cases, and augments the subset with counterexamples whenever an individual passes all of the current cases, or whenever some number of generations has passed.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1617087. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Iwo Błądek and Krzysztof Krawiec. 2019. Solving symbolic regression problems with formal constraints. In *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, Prague, Czech Republic, 977–984. https://doi.org/doi:10.1145/3321707.3321743

[2] Iwo Błądek, Krzysztof Krawiec, and Jerry Swan. 2018. Counterexample-Driven Genetic Programming: Heuristic Program Synthesis from Formal Specifications. *Evolutionary Computation* 26, 3 (Fall 2018), 441–469. https://doi.org/doi:10.1162/evco_a_00228

[3] Austin J Ferguson, Jose Guadalupe Hernandez, Daniel Junghans, Emily Dolson, and Charles Ofria. 2019. Characterizing the effects of random subsampling on Lexicase selection. In *Genetic Programming Theory and Practice XVII*, Wolfgang Banzhaf, Erik Goodman, Leigh Sheneman, Leonardo Trujillo, and Bill Worzel (Eds.). East Lansing, MI, USA.

[4] Thomas Helmuth and Lee Spector. 2015. General Program Synthesis Benchmark Suite. In *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, Madrid, Spain, 1039–1046. https://doi.org/10.1145/2739480.2754769

[5] Krzysztof Krawiec, Iwo Błądek, and Jerry Swan. 2017. Counterexample-driven Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. ACM, Berlin, Germany, 953–960. https://doi.org/doi:10.1145/3071178.3071224 Best paper.

[6] Krzysztof Krawiec, Iwo Błądek, Jerry Swan, and John H. Drake. 2018. Counterexample-Driven Genetic Programming: Stochastic Synthesis of Provably Correct Programs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, Jerome Lang (Ed.). International Joint Conferences on Artificial Intelligence, Stockholm, 5304–5308. https://doi.org/doi:10.24963/ijcai.2018/742

[7] Lee Spector, Jon Klein, and Maarten Keijzer. 2005. The Push3 execution stack and the evolution of control. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, Vol. 2. ACM Press, Washington DC, USA, 1689–1696. https://doi.org/doi:10.1145/1068009.1068292