# Using Knowledge of Human-Generated Code to Bias the Search in Program Synthesis with Grammatical Evolution

Dirk Schweim
schweim@uni-mainz.de
Johannes Gutenberg University
Mainz, Germany

Erik Hemberg
hembergerik@csail.mit.edu
MIT
Cambridge, MA, USA

Dominik Sobania
dsobania@uni-mainz.de
Johannes Gutenberg University
Mainz, Germany

Una-May O'Reilly
unamay@csail.mit.edu
MIT
Cambridge, MA, USA

Franz Rothlauf
rothlauf@uni-mainz.de
Johannes Gutenberg University
Mainz, Germany

## ABSTRACT

Recent studies show that program synthesis with GE produces code that has different structure compared to human-generated code, e.g., loops and conditions are hardly used. In this article, we extract knowledge from human-generated code to guide evolutionary search. We use a large code-corpus that was mined from the open software repository service GitHub and measure software metrics and properties describing the code-base. We use this knowledge to guide the search by incorporating a new selection scheme. Our new selection scheme favors programs that are structurally similar to the programs in the GitHub code-base. We find noticeable evidence that software metrics can help in guiding evolutionary search.

## CCS CONCEPTS

• **Software and its engineering** → **Search-based software engineering**; **Genetic programming**.

## KEYWORDS

Program Synthesis, Software Synthesis, Grammar Guided Genetic Programming, Genetic Programming, Grammatical Evolution, Mining Software Repositories

**ACM Reference Format:**
Dirk Schweim, Erik Hemberg, Dominik Sobania, Una-May O'Reilly, and Franz Rothlauf. 2021. Using Knowledge of Human-Generated Code to Bias the Search in Program Synthesis with Grammatical Evolution. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3449726.3459548

## 1 INTRODUCTION

A recent study [4] identified several problems in program synthesis with grammatical evolution (GE, [2]). For example, conditionals or loops are often not effectively used since the fitness signal does not guide the search towards these complex structures [4]. Instead, small building blocks are combined and the search iteratively evolves very specialized programs. The authors come to the conclusion that "the current problem specification and especially the definition of the fitness functions do not allow guided search, as the resulting problem constitutes a needle-in-a-haystack problem" [4]. They state that a main challenge for future research in program synthesis is to find new ways that help to guide the search.

In this article we focus on the question of how knowledge gained from human-generated code can be used as an additional bias to guide program synthesis with GE. In current approaches, general programming knowledge is only incorporated into the evolutionary search process via the BNF grammar. The evolved solutions are in many cases unreadable as well as "bloated" and thus hardly maintainable or testable [3]. We extend the current approaches and investigate the possibility to use software metrics from an existing code-base to guide the search with GE for program synthesis problems. Our work is a first step to evaluate the question if general programming knowledge can be used to bias an evolutionary search towards programs that are similar to human-generated programs.

Therefore, we mined a code-corpus, consisting of 211,060 real-world and high-quality Python functions. We use this human-generated code and measure the frequencies of software metrics that describe properties of the code in the code-base. Then, we propose multiple GE variants where the additional knowledge is used as an additional signal to guide the search. Our results show evidence that additional information can help in guiding the search. Furthermore, we gain valuable insight on how future approaches can be improved. For example, we learn that setting too many additional objectives is detrimental, because the conventional fitness signal is obfuscated.

## 2 GITHUB CODE CORPUS

In our experiments we use human-generated code in the programming language Python that was mined from the software repository hosting service GitHub. On GitHub, users are able to rate software repositories. We only use high-quality code, i.e., repositories with 150 or more positive ratings ("stars"). We cloned a total of 10,723 repositories that met the aforementioned search criteria. We use widely known software metrics to perform a descriptive analysis of


*GECCO '21 Companion, July 10–14, 2021, Lille, France*

ACM ISBN 978-1-4503-8351-6/21/07.
https://doi.org/10.1145/3449726.3459548

the code-base, investigating the question if there are certain similarities in high-quality human-generated Python code that could be useful to help in guiding an evolutionary search. In summary, code that received positive ratings ("stars") by human programmers has several similarities in its structural properties. Often, it consists of short code with a low complexity where certain concepts are used repeatedly. This is in line with the goal of this article: we seek to evolve simple, understandable code that looks like human-generated code. Therefore, it seems appropriate to use the metrics evaluated in this section to guide evolutionary search with the goal to improve the readability and maintainability of the generated code.

## 3 EXPERIMENTS

It is an open question how additional information can be appropriately added to an evolutionary search to increase search performance. Furthermore, it is not clear what kind of information is helpful. In our experiments, we use the frequency distributions of the software metrics gained from the GitHub code-base and test multiple methods how this additional knowledge can be added to the search. We compare five GE variants with a standard GE algorithm. Overall, the additional information gained from the GitHub repository does not improve the search performance. However, our goal is to improve readability and maintainability of the code. We can see that the best programs found with standard GE are very large compared to many other variants. Interestingly, the high parsimony pressure in one of the settings does not effectively prevent bloat. An explanation for this behavior is that, due to elitism, bloated solutions will always stay in the population when their performance is better compared to other programs that were optimized for their size. The search has to evolve small and highly fit solutions from time to time to effectively prevent bloat.

In the last part of our analysis we want to focus on the AST node types that are used in the evolved programs. It is very interesting to see that GE with additional information from software metrics often evolves programs that use the same node types that are also used in a hand-coded program. Other node types are rarely used and bloat results from very few meaningless function calls, e.g., "min(in0,in0)". This is a good property if the search gets a clear fitness signal like in the median problem. On the other hand, it limits the possibility to find more complex solutions. For example, the grammars that we used would also allow more complex solutions where loops, conditionals, and comparisons are necessary. To find such a complex solution will be hard when following the conventional fitness signal.

If we take into account both, the performance and the size of the evolved programs, a standard GE with a depth limit is the most favorable setting. With $d_{max} = 17$, bloat is limited very effectively while the performance is comparable to a standard GE without depth limit. However, the approach does not help to find complex programs. For example, our results indicate that more complex non-linear program structures like conditionals and loops are not effectively used by the current GE approaches (with "general grammars" that are not optimized for a problem instance)

In summary, in our experiments, we find evidence that software metrics and $n$-gram frequencies can help evolving good solutions, while helping to limit bloat and allowing for more complexity. Arguably, meta information like the software metrics investigated in this work is possibly not enough of additional signal to be effectively exploited by evolutionary search. This is no big surprise, since it is also hard for humans to quantify code quality. Overall, our work is a first step towards the goal to effectively use additional input signals to guide the search towards meaningful complex programs.

## 4 CONCLUSION

Creating high-quality code is a complex task, even for humans. The goal of this paper was to discuss and evaluate the idea of how additional knowledge gained from a large amount of high-quality human-generated code can help in an evolutionary search. We find noticeable evidence that meta-information can help in guiding the search.

Future work has to investigate how information from the existing code-base can be used more effectively to further improve the search performance. Furthermore, these new approaches should be combined with other additional sources of information, e.g., domain-specific knowledge and problem knowledge as proposed in [1].

## ACKNOWLEDGMENTS

The authors thank Jordan Wick for sharing his expertise, the insightful discussions, and his help on our project. This work was supported by a fellowship within the IFI programme of the German Academic Exchange Service (DAAD).

## REFERENCES

[1] Erik Hemberg, Jonathan Kelly, and Una-May O'Reilly. 2019. On Domain Knowledge and Novelty to Improve Program Synthesis Performance with Grammatical Evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '19)*. ACM, New York, NY, 1039–1046.

[2] Conor Ryan, J. J. Collins, and Michael O'Neill. 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language. In *Proceedings of the First European Workshop on Genetic Programming, EuroGP 1998*, Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty (Eds.). Springer, Berlin, Heidelberg, 83–96.

[3] Dominik Sobania and Franz Rothlauf. 2019. Teaching GP to Program Like a Human Software Developer: Using Perplexity Pressure to Guide Program Synthesis Approaches. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2019)*. ACM, New York, NY, 1065–1074.

[4] Dominik Sobania and Franz Rothlauf. 2020. Challenges of Program Synthesis With Grammatical Evolution. In *European Conference on Genetic Programming (LNCS, Vol. 12101)*, Ting Hu, Nuno Lourenço, Eric Medvet, and Federico Divina (Eds.). Springer International Publishing, Cham, 211–227.