Benchmarking Parent Selection for Program Synthesis by Genetic Programming

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

Amr Abdelhady
Hamilton College
Clinton, New York, USA
aabdelha@hamilton.edu

ABSTRACT

In genetic programming, the parent selection method determines which individuals in the population are selected to be parents for the next generation, and how many children they create. This process directly impacts the search performance by determining on which areas of the search space genetic programming focuses its attention and how it balances exploration and exploitation. Many parent selection methods have been proposed in the literature, with aims of improving problem-solving performance or other characteristics of the GP system. This paper aims to benchmark many recent and common parent selection methods by comparing them within a single system and set of benchmark problems. We specifically focus on the domain of general program synthesis, where solution programs must make use of multiple data types and control flow structures, and use an existing benchmark suite within the domain. We find that a few methods, all variants of lexicase selection, rise to the top and demand further study, both within the field of program synthesis and in other domains.

CCS CONCEPTS

• **Computing methodologies** → **Genetic programming**;

KEYWORDS

genetic programming, parent selection, benchmark, program synthesis

ACM Reference Format:
Thomas Helmuth and Amr Abdelhady. 2020. Benchmarking Parent Selection for Program Synthesis by Genetic Programming. 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/10.1145/3377929.3389987

1 INTRODUCTION

In this paper, we benchmark the performance of a set of 21 parent selection methods from the literature on 12 problems from the "General Program Synthesis Benchmark Suite" [3]. These problems expect a synthesis system to produce the types of programs that humans write, and require multiple data types and the use of control flow structures to solve. While our results only examine performance in the program synthesis domain, we expect that the findings will roughly translate to other domains as well.

This study considers some parent selection methods that have previously been tested on these program synthesis benchmark problems, as well as many selection methods that have not. Many of the methods we test are variants of either tournament selection or lexicase selection [4]. We include many selection methods that are popular in GP and evolutionary computation more generally, as well as some recently-published methods.

Others have benchmarked parent selection in GP and evolutionary computation more generally. Early work in genetic algorithms compared tournament selection, fitness-proportionate selection, ranking selection, and steady-state evolution [2]. More recently, a study of selection methods that sample the training set compares some of the methods we consider in this work [7]. While others have benchmarked some of the methods we consider, our study contains a larger breadth of parent selection methods than any study of which we are aware. Additionally, to our knowledge this is the first study to extensively benchmark the effects of parent selection on general program synthesis problems.

Our results show that lexicase selection variants perform far and away better than other methods, with lexicase-based methods dominating the top half of the ranking. In particular, down-sampled lexicase [1, 5] gave the best performance, placing in the top few positions on almost all of our 12 benchmark problems.

2 BENCHMARKING RESULTS

We use the PushGP genetic programming system to evaluate each parent selection method [9]. We measure performance as the number of successful runs out of 100 for each method. For this shortened paper, we will just give the average ranking of each method across the benchmark problems. We also do not have enough room to describe each method, though many can be found in the literature.

We present the average rankings of each method across the tested benchmark problems in Table 1. The main thing to notice is that the 11 algorithms that use or are based on lexicase selection fall within the first 12 spots in the list. Only pooled lexicase selection had a worse average ranking than the best tournament-based approach, which was batch tournament selection.

To give more detail about the best-performing methods, we plot the success rates of the top four lexicase selection variants as well as standard lexicase selection on each problem in Figure 1. One highlight here is that while down-sampled lexicase performs about as well as others on easier problems, it gives much better results on the five most difficult problems than any of the other methods, leading it to top the list of average ranks.

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

ACM ISBN 978-1-4503-7127-8/20/07.
https://doi.org/10.1145/3377929.3389987

Table 1: Average ranking of each of the 21 parent selection methods across 10 benchmark problems, not including the Mirror Image or Smallest problems.

Method	Average Ranking
Down-sampled lexicase	2.2
MADCAP ϵ-lexicase	4.7
ALPS with lexicase	5.4
Median lexicase	5.8
Knobelty	6.1
Ranked shuffle lexicase	6.2
Lexicase	6.2
Novelty-lexicase	6.6
ϵ-lexicase	7.3
Batch lexicase	8.8
Batch tournament	10.5
Pooled lexicase	10.8
Pareto tournament, age	14.6
Implicit fitness sharing	15.1
Tournament	15.2
ALPS tournament	15.5
Interleaved sampling	16.0
Pareto tournament, size	18.0
Fitness-proportionate	18.3
Novelty search	18.8
Uniform	18.9

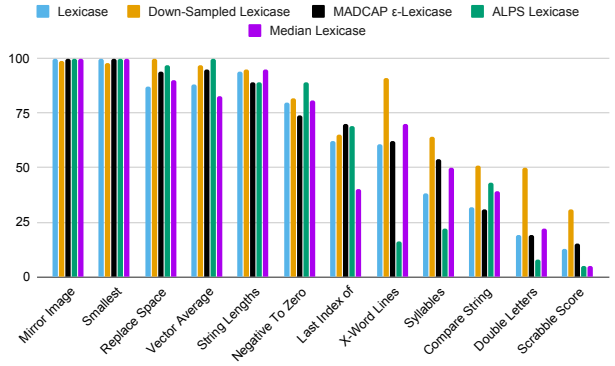

Figure 1: Number of successes out of 100 PushGP runs. This figure shows standard lexicase and the four best lexicase variants.

3 DISCUSSION AND CONCLUSIONS

This paper benchmarks 21 GP parent selection methods on 12 problems from a program synthesis benchmark suite, finding that some methods perform much better than others in this domain. In particular, lexicase selection variants outperform all others, and down-sampled lexicase gave the best success rates of all. We find that many common parent selection methods, such as tournament selection, fitness-proportionate selection, and novelty search perform poorly in this setting.

Down-sampled lexicase selection [1, 5] performed best of any of the methods tested in this paper. We attribute down-sampled lexicase selection's success to it using a smaller subsample of the training cases to evaluate each individual, resulting in fewer program executions per generation and therefore more generations per run. Unless stuck in a local optima, having more generations usually means better performance on the given problem, as more individuals are considered over the course of the evolutionary process.

The second best method in our experiments, MADCAP ϵ-lexicase selection [8], is a variant of lexicase selection that falls somewhere between standard lexicase and ϵ-lexicase [6]. This method uses normal lexicase half of the time, and the other half of the time lets any individual survive that is within some ϵ of the best individual. The relaxation of lexicase may allow some individuals to be selected that have poor errors in a few cases.

The dominance of lexicase-based variants in our benchmarking compared to tournament-based variants clearly shows the benefits of the underlying algorithm compared to methods that aggregate fitness into a single value. While the differences were only significant for some specific comparisons, the fact that 11 of the first 12 methods (and all of the first 10) in our average rankings are based on lexicase selection shows that it, at this point in time, has no peer when using GP to solve program synthesis problems. That said, our results suggest that some lexicase variants are *worse* than others for program synthesis; we would recommend against using ϵ-lexicase, batch lexicase, or pooled lexicase in this domain.

REFERENCES

[1] 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.

[2] David E. Goldberg and Kalyanmoy Deb. 1991. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. Foundations of Genetic Algorithms, Vol. 1. Elsevier, 69 – 93. https://doi.org/10.1016/B978-0-08-050684-5.50008-2

[3] 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

[4] Thomas Helmuth, Lee Spector, and James Matheson. 2015. Solving Uncompromising Problems with Lexicase Selection. *IEEE Transactions on Evolutionary Computation* 19, 5 (Oct. 2015), 630–643. https://doi.org/10.1109/TEVC.2014.2362729

[5] Jose Guadalupe Hernandez, Alexander Lalejini, Emily Dolson, and Charles Ofria. 2019. Random subsampling improves performance in lexicase selection. In *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, Prague, Czech Republic, 2028–2031. https://doi.org/doi:10.1145/3319619.3326900

[6] William La Cava, Thomas Helmuth, Lee Spector, and Jason H. Moore. 2019. A probabilistic and multi-objective analysis of lexicase selection and epsilon-lexicase selection. *Evolutionary Computation* 27, 3 (2019), 377–402. https://doi.org/doi:10.1162/evco_a_00224

[7] Yuliana Martinez, Enrique Naredo, Leonardo Trujillo, Pierrick Legrand, and Uriel Lopez. 2017. A comparison of fitness-case sampling methods for genetic programming. *Journal of Experimental & Theoretical Artificial Intelligence* 29, 6 (2017), 1203–1224. https://doi.org/doi:10.1080/0952813X.2017.1328461

[8] Lee Spector, William La Cava, Saul Shanabrook, Thomas Helmuth, and Edward Pantridge. 2017. Relaxations of Lexicase Parent Selection. In *Genetic Programming Theory and Practice XV (Genetic and Evolutionary Computation)*, Wolfgang Banzhaf, Randal S. Olson, William Tozier, and Rick Riolo (Eds.). Springer, University of Michigan in Ann Arbor, USA, 105–120. https://doi.org/doi:10.1007/978-3-319-90512-9_7

[9] Lee Spector and Alan Robinson. 2002. Genetic Programming and Autoconstructive Evolution with the Push Programming Language. *Genetic Programming and Evolvable Machines* 3, 1 (March 2002), 7–40. https://doi.org/doi:10.1023/A:1014538503543