Random subsampling improves performance in lexicase selection

Jose Guadalupe Hernandez
Michigan State University
East Lansing, Michigan
herna383@msu.edu

Alexander Lalejini
Michigan State University
East Lansing, Michigan
lalejini@msu.edu

Emily Dolson
Michigan State University
East Lansing, Michigan
dolsonem@msu.edu

Charles Ofria
Michigan State University
East Lansing, Michigan
ofria@msu.edu

ABSTRACT

Lexicase selection has been proven highly successful for finding effective solutions to problems in genetic programming, especially for test-based problems where there are many distinct test cases that must all be passed. However, lexicase (as with most selection schemes) requires all prospective solutions to be evaluated against most test cases each generation, which can be computationally expensive. Here, we propose reducing the number of per-generation evaluations required by applying random subsampling: using a subset of test cases each generation (down-sampling) or by assigning test cases to subgroups of the population (cohort assignment). Tests are randomly reassigned each generation, and candidate solutions are only ever evaluated on test cases that they are assigned to, radically reducing the total number of evaluations needed while ensuring that each lineage eventually encounters all test cases. We tested these lexicase variants on five different program synthesis problems, across a range of down-sampling levels and cohort sizes. We demonstrate that these simple techniques to reduce the number of per-generation evaluations in lexicase can substantially improve overall performance for equivalent computational effort.

CCS CONCEPTS

• **Software and its engineering → Genetic programming**; *Automatic programming*;

KEYWORDS

parent selection, lexicase selection, down-sampled lexicase, cohort lexicase, cohorts, genetic programming, program synthesis

ACM Reference Format:
Jose Guadalupe Hernandez, Alexander Lalejini, Emily Dolson, and Charles Ofria. 2019. Random subsampling improves performance in lexicase selection. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3319619.3326900

1 INTRODUCTION

We often apply evolutionary computation to test-based problems where the quality of a candidate solution is assessed by evaluating it on a large set of test cases. For such problems, we must select parents (*i.e.*, genetic source material) for each generation based on how well individuals solve each test case. In many test-based problems, the space of possible test cases is either infinite or so large that it is not computationally feasible to evaluate a candidate solution on every possible test case. In the absence of extensive domain knowledge, it can be challenging to find an optimal test set size. Too small, and we risk overfitting. Too large, and the demand on computational resources will bring adaptive evolution to a crawl.

Lexicase selection is a more recent technique developed for genetic programming (GP) that has been demonstrated to be particularly effective for solving challenging test-based problems [8, 11, 13]. The lexicase algorithm chooses each parent for the next generation by sequentially applying test cases, in a random order. Only the best performers on each test case are kept, until a single individual is identified. This sequential filtering approach is a departure from traditional parent-selection methods that calculate an absolute fitness metric by summing an individual's performance across all test cases. Because lexicase changes the ordering of test cases for every parent-selection event, individuals that perform well on different subsets of test cases can co-exist. This dynamic allows lexicase selection to maintain specialists on tests that the majority of the population fail, preserving potentially important genetic material [2, 6] and thus searching for a perfect solution from many directions at once.

The drawback of lexicase (and many other test-based selection schemes) is that assessing candidate solutions on a large set of test cases can be computationally expensive, especially if individual evaluations are costly. A simple speed-up might seem to be cutting down the number of evaluations by limiting the number of successive filtering steps taken during each lexicase selection event, shifting to a random selection if multiple solutions are still available (*e.g.*, truncated lexicase [14]). However, in practice, each candidate solution must still be evaluated on most test cases every generation.

We could trivially decrease the number of evaluations per generation by statically reducing the total number of test cases used during evolutionary search. For example, a 50% reduction in test cases would allow us to run our search for about twice as many generations. However, simply reducing the total number of tests is more likely to result in prospective solutions overfitting the reduced test set. Reducing computational effort on test-based problems is a long-standing endeavour for GP [4]. Many techniques have been proposed that dynamically subsample the set of tests (from a large pool) used for candidate solution assessment and selection (see [9, 11] for recent reviews). Subsampling techniques have been employed to reduce computational effort in GP [1, 4] and to improve the generalizability of evolved programs [5, 11]. Can we apply test-case subsampling techniques to lexicase selection?

Here, we examine two lexicase selection variants that leverage random subsampling to reduce the number of evaluations per generation: *down-sampled lexicase* and *cohort lexicase*. Down-sampled lexicase selects parents based on a random subset of test cases each generation, guaranteeing that individuals are only evaluated against test cases in the subset. Cohort lexicase uses all test cases each generation, but divides both tests and individuals into cohorts, ensuring that each individual is evaluated against only a subset of tests. By reshuffling which test cases are experienced every generation, *lineages* will eventually encounter all test cases. We compare the results of different configurations of down-sampled and cohort lexicase across five program synthesis problems. Additionally, we compare the performance of our proposed lexicase variants to that of standard lexicase with a reduced number of total tests.

2 LEXICASE SELECTION

Lexicase selection is a method for choosing a candidate solution from a population to use as a parent (*i.e.*, to provide genetic source material for a new individual in the next generation). Each such parent is selected individually, with replacement, such that individuals may be chosen multiple times. In lexicase, a large number of test cases are used as criteria for evaluation. Unlike many traditional parent-selection methods, lexicase does not aggregate performance across test cases to calculate a single fitness score. Instead, each time a parent is needed, test cases are successively applied in a random order, keeping only the most fit candidates on each. This process continues until the population is filtered down to either a single candidate or a set of equivalent candidates (at which point one is selected randomly). Because the ordering of test cases changes for every parent-selection event, individuals that perform well on different subsets of test cases are able to co-exist [2, 6]. A more detailed description of lexicase selection can be found in [8, 13].

Spector [13] initially proposed lexicase selection as a GP selection scheme for modal problems where qualitatively different modes of response are required for inputs from different regions of the problem domain. Subsequent work demonstrated lexicase selection's efficacy relative to traditional parent-selection algorithms on *uncompromising* problems where solutions must perform optimally over the entire space of possible test cases [8]. Part of lexicase selection's success is attributed to its effectiveness at diversity maintenance; lexicase maintains specialists on test cases that the majority of the population fail, preserving potentially important genetic material [2, 6]. For an analysis of lexicase selection in the context of ecological theory, see [2].

Several variants of lexicase selection have previously been proposed [14]. We propose two new lexicase variants that relax the need to evaluate all candidate solutions against most test cases, thus allowing computational resources to be reallocated to additional search time, larger population sizes, *et cetera*.

3 DOWN-SAMPLED LEXICASE SELECTION

In each generation of standard lexicase selection, every test in the test case set is available as evaluation criteria for selection events; thus, all individuals must be evaluated against *most* test cases each generation. Assuming we can store and reuse previously computed performances for each repeated application of a test case during parent selection events, lexicase selection's worst-case number of per-generation evaluations is equal to the size of the test case set multiplied by the population size (*i.e.*, every member of the population is evaluated against every test case once).

Down-sampled lexicase applies the random subsampling technique [5] to lexicase selection. Each generation, down-sampled lexicase selects a random subset of the test cases to use for all selection events, guaranteeing that unselected test cases are not evaluated against at all. Here, we refer to the our 'down-sample factor' (the subsample rate) as D. For example, $D = 10$ implies a tenfold subsample rate (*i.e.*, each generation, we use $\frac{1}{10}$ of the total test case set to evaluate individuals). This down-sampling divides the worst-case number of evaluations performed each generation by D, allowing us to run our evolutionary search for more generations (or with a larger population size) than standard lexicase selection. Here, we exclusively apply random subsampling every generation; however, as discussed by Gonçalves *et al.* [5], we could also vary the number of generations at which we apply random subsampling.

Why is down-sampling the test case set preferable to simply reducing the number of test cases? In down-sampled lexicase selection, lineages are likely to be tested against a large portion of the full test set over several generations. Each generation, a candidate solution will encounter a proportion of test cases equal to $\frac{1}{D}$; thus, $1 - \frac{1}{D}$ gives the proportion of test cases *not* encountered by a candidate solution in a given generation. The *expected* proportion of test cases not encountered by a *lineage* after G generations is $(\frac{D-1}{D})^G$. To calculate the expected number of generations for a lineage to be evaluated against proportion T of the test cases for a known down-sampling rate ($\frac{1}{D}$), we can solve for G in Equation 1.

$$G = \frac{log(1-T)}{log(D-1) - log(D)} \tag{1}$$

Note that a lineage will always encounter proportion $T \leq \frac{1}{D}$ in a single generation, and T asymptotically approaches 1.0 as the number of generations increases.

4 COHORT LEXICASE SELECTION

Cohort lexicase selection makes use of the full test case set each generation but ensures that each prospective solution is evaluated against only a subset of them. Every generation, cohort lexicase randomly partitions both the population and test case set into K equally-sized sub-groups (cohorts). Each of the K candidate solution cohorts is then paired with a test case cohort, and each candidate solution in a cohort is evaluated against all test cases in the associated test case cohort. This means that the number of evaluations performed each generation (relative to standard lexicase selection) is divided by K. Candidate solutions only compete within their cohort, and within-cohort competition is arbitrated by the test cases in the associated cohort of tests. In this way, cohorts impose a sort of island model [15] on standard lexicase selection where each island's membership (candidate solutions) and environment (test cases) is transient, randomized every generation.

Our formulation of cohort lexicase follows the same expectations as down-sampled lexicase for the number of generations before a lineage is expected to encounter proportion T test cases (given by Equation 1). Cohort lexicase's K and down-sampled lexicase's D create an equivalent down-sampling rate. Note, however, in our implementation of cohort lexicase, tests are not repeated across cohorts; though, there is no reason why they could not be repeated.

5 METHODS

To test the utility of down-sampled and cohort lexicase selection, we used both selection schemes to evolve linear genetic programs to solve five test-based problems from the program synthesis benchmark suite [7]: Small or Large, For Loop Index, Compare String Lengths, Median, and Smallest. A description of our GP system (including source code) can be found in supplemental material [10].

5.1 Program Synthesis Problems

Problems in the general program synthesis benchmark suite were selected from sources for introductory computer science programming problems; while not particularly challenging for experienced human programmers, they can be challenging for current GP systems [3, 7]. These benchmarks have been used to compare lexicase selection against other, more traditional selection schemes [7]. Previous studies (using PushGP [7] and G3P [3]) have shown standard lexicase selection to be capable of solving the five problems used in this work, making them good choices for evaluating random test subsampling in the context of lexicase selection.

Each problem is defined by a set of test cases in which programs are given input data and are scored on how well their output matches the correct output (assigning scores on a gradient or pass-fail basis as appropriate). During an evaluation, the total number of steps (instructions) a problem could execute varied by problem.

During evolution, programs were assessed using a training set of test cases, which defined the selection criteria used for lexicase selection. To qualify as a solution, a program needed to perfectly pass all test cases in a separate testing set (withheld generalization examples) in addition to passing all tests in the training set used during evaluation. For all problems, we used the same training and testing sets (100 training cases and 1,000 testing cases) and the same input constraints as in [7]. The exact training and testing sets used can also be found in our supplemental material [10].

For a more detailed description of the five benchmark problems used here (Small or Large, For Loop Index, Compare String Lengths, Median, and Smallest), see [7] or our supplemental material [10]. For each problem, we added problem-specific instructions (see [10]) to our GP instruction set to allow programs to load test case inputs into memory and submit output.

5.2 Experimental Design

We evolved populations of 1,000 programs under a range of subsampling levels (*i.e.*, the percent of the training set used to assess candidate solutions) using both down-sampled and cohort lexicase: 5%, 10%, 25%, 50%, and 100% (no reduction). Additionally, we evolved programs using standard lexicase selection (no subsampling) with 5%, 10%, 25%, 50%, and 100% (no reduction) of the training set; when reducing the training set for standard lexicase selection runs, we randomly selected the appropriate percentage of test cases from the full training set (*e.g.*, 5 of the 100 total test cases when using 5% of the training set), and the reduced training set remained static for the duration of evolutionary search.

We ran 100 replicates of all conditions, each for a fixed budget of 30,000,000 evaluations (*i.e.*, 300 generations when using the full training set). Conditions where we subsampled or reduced the training set ran for more generations than conditions using the full training set (5%: 6,000 generations; 10% 3,000 generations; 25%: 1,200 generations; 50%: 600 generations). For each problem and selection condition, we compared the problem solving success rates (*i.e.*, the number of runs in which a perfect solution evolved) of using fewer training cases (via cohorts, down-sampling, or static reduction) versus using the full training set during selection (Fisher's exact test with a significance level of 0.05 and a Holm-Bonferonni correction for multiple comparisons). All statistical analyses were performed using the R Statistical Computing Platform [12]. The source code for our analyses and data visualizations can be found in our supplemental material [10].

6 RESULTS AND DISCUSSION

Figure 1 shows the problem solving success for all experimental conditions across all five problems after a fixed number of test case evaluations; see our supplemental material [10] for more detailed statistical analyses. With the exception of the For Loop Index problem, reducing the size of the training set for standard lexicase selection (resulting in more generations of evolution) did not improve (by a statistically significant amount) problem solving success. Indeed, on the Compare String Lengths, Median, and Smallest problems, reducing the training set beyond a critical threshold (which varied by problem) when using standard lexicase selection significantly reduced problem solving success relative to using the the full test case set (*e.g.*, Compare String Lengths, 50% training: $p < 0.021$; Median, 10% training: $p < 3.68\text{e-}10$; Smallest, 25% training: $p < 0.003$). These reduced success rates are likely due to overfitting: we sufficiently reduced the training set such that it does not adequately represent the full space of test cases, and as a result, evolved programs fail to generalize. On the For Loop Index problem, using standard lexicase with only 25% of the full training set has a significantly *higher* success rate than when using the full training set ($p = 0.017$); in this case, reducing the size of the training set to rapidly progress through more generations pays off, which suggests that the full training set for this problem is unnecessarily large to thoroughly assess candidate solutions.

Multiple configurations of down-sampled lexicase significantly improved problem solving success relative to standard lexicase across all problems except Compare String Lengths where improvements are not statistically significant (*e.g.*, Small or Large, 50% training:

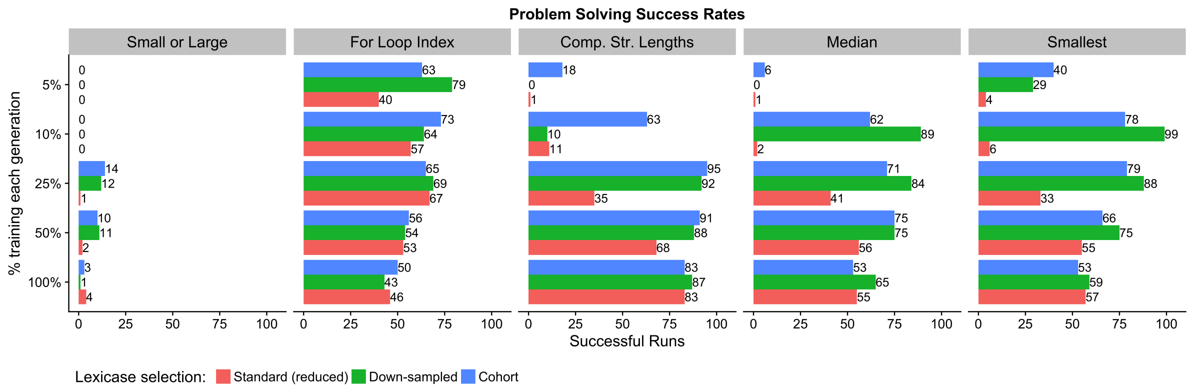

Figure 1: Problem solving success rates (*i.e.*, the number of runs in which a perfect solution evolved) for each program synthesis problem. Note that, here, all conditions using 100% of the training set (regardless of lexicase variant) are qualitatively identical conditions.

$p < 0.015$; For Loop Index, 25% training: $p < 0.002$; Median, 25% training: $p < 0.007$; Smallest, 50% training: $p < 0.024$). Similarly, at least one configuration of cohort lexicase significantly improved problem solving success relative to standard lexicase across all problems (*e.g.*, Small or Large, 25% training: $p < 0.034$; For Loop Index, 10% training: $p < 0.006$; Compare String Lengths, 25% training: $p < 0.023$; Median, 50% training: $p < 0.006$; Smallest, 25% training: $p < 0.001$). The particular configurations of down-sampled and cohort lexicase that work best depend on the problem. Neither cohort or down-sampled lexicase consistently outperformed the other on any of the five problems.

These results suggest that: (1) random subsampling can be used to improve the problem solving performance of lexicase selection, and (2) both cohort and down-sampled lexicase are successful approaches for applying random subsampling to standard lexicase.

7 CONCLUSION

We presented two extensions of the lexicase parent selection algorithm that incorporate random subsampling techniques: down-sampled lexicase and cohort lexicase. Using these techniques, we confirm that random subsampling can be successfully applied lexicase selection, allowing evolutionary search to more rapidly progress through generations and improving problem solving success rates. Our experimental results suggest that the best configuration of down-sampled and cohort lexicase depends on the problem. Future studies will tease apart how different levels of subsampling impact lexicase selection (*e.g.*, diversity maintenance).

ACKNOWLEDGMENTS

We thank Wolfgang Banzhaf and the Digital Evolution Lab at Michigan State University (MSU) for discussions and feedback. This research was supported by the National Science Foundation through the BEACON Center (Coop. Agreement No. DBI-0939454), a Graduate Research Fellowship to AL and ED (Grant No. DGE-1424871), and Grant No. DEB-1655715 to CO.

REFERENCES

[1] Robert Curry and Malcolm Heywood. 2004. Towards Efficient Training on Large Datasets for Genetic Programming. In *Advances in Artificial Intelligence*, Ahmed Y. Tawfik and Scott D. Goodwin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 161–174.
[2] Emily L Dolson, Wolfgang Banzhaf, and Charles Ofria. 2018. Ecological theory provides insights about evolutionary computation. *PeerJ Preprints* 6 (Nov. 2018), e27315v1. https://doi.org/10.7287/peerj.preprints.27315v1
[3] Stefan Forstenlechner, David Fagan, Miguel Nicolau, and Michael O'Neill. 2018. Towards Understanding and Refining the General Program Synthesis Benchmark Suite with Genetic Programming. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–6. https://doi.org/10.1109/CEC.2018.8477953
[4] Chris Gathercole and Peter Ross. 1994. Dynamic training subset selection for supervised learning in Genetic Programming. In *Parallel Problem Solving from Nature — PPSN III*, Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 312–321.
[5] Ivo Gonçalves, Sara Silva, Joana B. Melo, and João M. B. Carreiras. 2012. Random Sampling Technique for Overfitting Control in Genetic Programming. In *Genetic Programming*. 218–229. https://doi.org/10.1007/978-3-642-29139-5_19
[6] Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. 2016. Effects of Lexicase and Tournament Selection on Diversity Recovery and Maintenance. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO '16 Companion*. ACM Press, New York, New York, USA, 983–990. https://doi.org/10.1145/2908961.2931657
[7] Thomas Helmuth and Lee Spector. 2015. General Program Synthesis Benchmark Suite. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*. ACM Press, New York, New York, USA, 1039–1046. https://doi.org/10.1145/2739480.2754769
[8] Thomas Helmuth, Lee Spector, and James Matheson. 2015. Solving Uncompromising Problems With Lexicase Selection. *IEEE Transactions on Evolutionary Computation* 19, 5 (10 2015), 630–643. https://doi.org/10.1109/TEVC.2014.2362729
[9] Hmida Hmida, Sana Ben Hamida, Amel Borgi, and Marta Rukoz. 2017. Sampling Methods in Genetic Programming Learners from Large Datasets: A Comparative Study. In *Advances in Big Data*. Springer International Publishing, Cham, 50–60.
[10] Alexander Lalejini and Jose Hernandez. 2019. GECCO-2019-cohort-lexicase GitHub Repository. (March 2019). https://github.com/amlalejini/GECCO-2019-cohort-lexicase DOI: 10.5281/zenodo.2603050.
[11] Yuliana Martínez, Enrique Naredo, Leonardo Trujillo, Pierrick Legrand, and Uriel López. 2017. A comparison of fitness-case sampling methods for genetic programming. *Journal of Experimental and Theoretical Artificial Intelligence* 29, 6 (2017), 1203–1224. https://doi.org/10.1080/0952813X.2017.1328461
[12] R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/
[13] Lee Spector. 2012. Assessment of problem modality by differential performance of lexicase selection in genetic programming. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion - GECCO Companion '12*. ACM Press, New York, New York, USA, 401. https://doi.org/10.1145/2330784.2330846
[14] Lee Spector, William La Cava, Saul Shanabrook, Thomas Helmuth, and Edward Pantridge. 2018. Relaxations of Lexicase Parent Selection. In *Genetic Programming Theory and Practice XV*, Wolfgang Banzhaf, Randal S. Olson, William Tozier, and Rick Riolo (Eds.). Springer International Publishing, Cham, 105–120.
[15] Sewall Wright. 1943. Isolation by Distance. *Genetics* 28, 2 (1943), 114–138. http://www.genetics.org/content/28/2/114