# Chapter 1
# Characterizing the Effects of Random Subsampling on Lexicase Selection


**Austin J. Ferguson, Jose Guadalupe Hernandez, Daniel Junghans, Alexander Lalejini, Emily Dolson, and Charles Ofria**

## 1.1 Introduction

Evolutionary computation is often used to solve complex, multi-faceted problems where the quality of a candidate solution is measured according to its performance on a large set of test cases. For these test-based problems, we must somehow meld performances across many test cases to select individuals to serve as parents for the next generation. In many test-based problems, we cannot exhaustively evaluate a candidate solution over the entire space of possible test cases. As a result, it can be challenging to balance the trade-off between using a large enough test set to thoroughly evaluate candidate solutions while keeping the test set small enough to preserve computational resources and rapidly progress through generations.

Lexicase selection is a relatively new parent-selection algorithm developed for genetic programming (GP) and has been demonstrated as an effective tool for solving difficult test-based problems [11, 12, 27]. Many traditional selection strategies for solving test-based problems score potential solutions by aggregating their fitness across all test cases. The lexicase algorithm, however, chooses each parent for the next generation by sequentially applying test cases in a random order, keeping only the best performers on each test case until the population has been winnowed to a single individual. Because the ordering of test cases changes for

A. J. Ferguson (✉) · J. G. Hernandez · D. Junghans · A. Lalejini · C. Ofria
The BEACON Center for the Study of Evolution in Action, Michigan State University, East Lansing, MI, USA
e-mail: fergu358@msu.edu; herna383@msu.edu; junghan2@msu.edu; lalejini@msu.edu; ofria@msu.edu

E. Dolson
Department of Translational Hematology and Oncology Research, Cleveland Clinic, Cleveland, OH, USA
e-mail: dolsonem@msu.edu



W. Banzhaf et al. (eds.), *Genetic Programming Theory and Practice XVII*, Genetic and Evolutionary Computation, https://doi.org/10.1007/978-3-030-39958-0_1

every parent selection event, individuals that perform well on different subsets of test cases are able to co-exist [4, 9].

The drawback of many test-based selection schemes, including lexicase, is that assessing individuals using a large set of test cases can be computationally expensive; this drawback is exacerbated when tests are costly to perform (e.g., robotics simulations). Using a large number of test cases constrains the number of generations we are able to run evolutionary search. Using too few test cases, however, may fail to accurately represent the problem domain and lead to overfitting. To combat this, many techniques dynamically subsample test cases (from a large pool representative of the problem domain) for candidate solution evaluation and selection (see [14, 20] for recent reviews). Indeed, subsampling has been used to reduce computational effort in GP [2, 7] and to improve the generalizability of evolved programs [8, 20].

In this chapter, we characterize the effects of random subsampling on the lexicase parent-selection algorithm. Previous work has shown that lexicase selection performs well when combined with random subsampling. Moore and Stanton applied random subsampling to lexicase selection in the context of an evolutionary robotics problem because evaluating robot controllers on test cases (simulation environments) was too costly to permit exhaustive assessments [23–25]. In [13], we proposed down-sampled and cohort lexicase selection, two variants of standard lexicase that employ random subsampling to reduce the number of per-generation evaluations required by lexicase selection. We demonstrated that both down-sampled and cohort lexicase could yield higher problem-solving success than standard lexicase on a fixed evaluation budget in the context of program synthesis [13].

Here, we explore *why* random subsampling can improve lexicase selection's problem-solving success. Additionally, we characterize the effect of subsampling on diversity and specialist maintenance, both of which have been shown to be important factors behind lexicase selection's efficacy [4, 9, 10, 24]. We show that the improvement in problem-solving success gained from subsampling is due to its facilitation of *deeper* evolutionary searches (i.e., consisting of more generations relative to standard lexicase) given a fixed evaluation budget. Moreover, we show that both down-sampled and cohort lexicase find solutions with less computational effort than standard lexicase. While we predicted that subsampling would degrade diversity, we find no evidence for systematic degradation of phenotypic diversity. However, as the level of subsampling increases, cohort lexicase generates and maintains more phylogenetic diversity than down-sampled lexicase. As expected, we find that random subsampling degrades specialist preservation relative to standard lexicase. Our phenotypic diversity results seem to contradict our specialist preservation findings; this could be because of the particular problems we are using or because of our choice of time to measure phenotypic diversity (at the time a solution was found). Future work will continue investigating how subsampling affects diversity maintenance in an expanded problem domain and with more fine-grained data collection and analysis.

## 1.2 Lexicase Selection

Spector [27] initially proposed the lexicase parent-selection algorithm for solving modal GP problems where programs may have to output qualitatively different responses to different inputs. To accomplish this, lexicase does not aggregate fitness across test cases like many selection schemes. Instead, for each selection event (where a single parent must be selected), lexicase randomly permutes the test cases in the training set. Each test case is then considered in this permuted order, keeping only those candidate solutions that solve the focal test case (or tie for highest fitness if no candidate solutions solve it). This process continues until either a single candidate solution remains or all test cases have been exhausted. If more than one candidate solution remains, the winner is chosen at random. Each selection event follows this pattern with a different permutation until all parents for the next generation have been selected. Because the order of test cases changes for every parent selection event, individuals that perform well on different subsets of test cases are able to co-exist [4, 9]. This dynamic creates niches in a lexicase population and encourages multiple co-existing solutions that focus on different subsets of test cases. See [12, 27] for a more detailed description of lexicase selection.

Since its conception, lexicase selection has been successfully applied in the field of genetic programming. Such applications include program synthesis [11] and regression [16]. Lexicase selection has also been in other areas such as evolutionary robotics [23], genetic algorithms [22], and learning classifier systems [1].

### *1.2.1 Applying Subsampling to Lexicase Selection*

Several variants of lexicase selection (and lexicase-inspired selection algorithms) exist, such as $\epsilon$-lexicase, truncated lexicase, batch-tournament, batch-lexicase, down-sampled lexicase, and cohort lexicase [1, 13, 21, 28]. Here, we investigate down-sampled and cohort lexicase, both of which leverage random subsampling to reduce the number of per-generation evaluations required for lexicase selection.

#### 1.2.1.1 Down-Sampled Lexicase

Down-sampled lexicase [13] applies the random subsampling technique [8] to lexicase selection. Each generation, down-sampled lexicase selects a random subset of the test-cases in the training set to use for all selection events, guaranteeing that unselected test cases are not evaluated. Here, we use $D$ to represent our 'down-sample factor', where each generation $\frac{1}{D}$ of the training set is used. For example, a $D$ of 5 implies a 20% subsampling rate (i.e., each generation we use one fifth of the training set to evaluate individuals). Down sampling divides the number of evaluations performed each generation by $D$. Given a fixed budget of evaluations,

the computational savings afforded by down sampling allows us to continue our evolutionary search for more generations (or with a larger population size) than standard lexicase selection.

#### 1.2.1.2 Cohort Lexicase

Cohort lexicase selection [13] makes use of the full set of training cases each generation while also ensuring that each prospective solution is evaluated against only a subset of tests. 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 paired with a test-case cohort, and each candidate solution is evaluated against only the test cases in its cohort. Thus, the number of evaluations performed each generation (relative to standard lexicase selection) is divided by $K$. Candidate solutions compete only within their cohort, and within-cohort competition is arbitrated by the test cases in the associated cohort. Because cohorts are shuffled each generation, offspring will be assessed with a different subset of test cases than their parents. Note that the down-sampling factor, $D$, is identical to the number of cohorts, $K$, in both the total number of evaluations and the number of training cases a candidate solution sees per generation. Thus, $K$ and $D$ provide equivalent subsampling rates for the two selection schemes, and hereafter, we substitute $D$ for $K$ to simplify comparisons between down-sampled and cohort lexicase.

## 1.3 Methods

We conducted a series of experiments to characterize the effects of applying random subsampling to lexicase selection. In all evolution experiments, we evolved populations of linear genetic programs to solve four program synthesis problems. Using this setup, we replicated previous results [13], tested the effect of the additional generations afforded by subsampling, and investigated how different types of subsampling affect the computational effort expended to solve problems. Additionally, we analyzed how these subsampling techniques affect both population diversity and specialist maintenance.

### *1.3.1 Evolutionary System*

For each of our evolution experiments, we evolved populations of 1000 linear genetic programs on four program synthesis problems (each described in detail in Sect. 1.3.2). Our linear-GP representation used:

- an instruction set that includes arithmetic, memory management, flow-control, and additional problem-specific instructions
- memory accessed with binary tags [18]
- modules referenced via binary tags [17, 29]

A more detailed description of our GP system (including source code) can be found in the supplemental material [5].

We propagated programs asexually, subjecting offspring to mutations. Single-instruction insertions, deletions, and substitutions were applied, each at a per-instruction rate of 0.005. Modules were duplicated and deleted at a per-module rate of 0.05. We also applied 'slip' mutations [19], which have the possibility of duplicating or deleting sequences of instructions, at a per-program rate of 0.05. Program-tags were mutated at a per-bit rate of 0.001. The run-termination criteria varied per experiment and is included in each experiment description.

### *1.3.2 Program Synthesis Problems*

For all evolution experiments, we evolved programs to solve problems from the general program synthesis benchmark suite [11]. To test our hypotheses, we needed a set of problems known to be challenging but not impossible for GP systems to solve. The general program synthesis benchmark suite comprises introductory-level computer science programming questions, many of which have been solved using lexicase selection [6, 11]. We used the following four program synthesis problems in our experiments: *Smallest*, *Median*, *For Loop Index*, and *Grade*. A description of each problem is given below:

**Smallest** Programs are given four integer inputs ($-100 \leq input_i \leq 100$) and must output the smallest value. We measured program performance on a pass-fail basis. We limited program length to a maximum of 64 instructions and also limited the maximum number of instruction-execution steps to 64.

**Median** Programs are given three integer inputs ($-100 \leq input_i \leq 100$) and must output the median value. We measured program performance against test cases on a pass-fail basis. We limited program length to 64 instructions and also limited the maximum number of instruction-execution steps to 64.

**For Loop Index** Programs receive three integer inputs *start* ($-500 \leq start \leq 500$), *end* ($-500 \leq end \leq 500$), ($start < end$), and *step* ($1 \leq step \leq 10$). Programs must output the following sequence:

$$n_0 = start$$

$$n_i = n_{i-1} + step$$

for each $n_i < end$. We limited program length to a maximum of 128 instructions and also limited the maximum number of instruction-execution steps to 256. Program performance against a test case was measured on a gradient, using the Levenshtein distance between the program's output and the correct output sequence.

**Grade** Programs receive five integers in the range [0, 100] as input: $A$, $B$, $C$, $D$, and $score$. $A$, $B$, $C$, and $D$ define the minimum score needed to receive that letter grade. These are specified such that $A > B > C > D$ (i.e., they are monotonically decreasing and unique). The program must read in these thresholds and return the appropriate letter grade for the given $score$, or $F$ if $score < D$. We limited program length to a maximum of 64 instructions and also limited programs' maximum instruction-execution steps to 64. On each test, we evaluated programs on a pass-fail basis.

For these experiments, the *Smallest*, *Median*, and *For Loop Index* problems have an associated training set of 100 test cases, and a separate validation set of 1000 test cases (withheld during fitness evaluations). We used 200 training cases and 2000 validation cases for the *Grade* problem. A program had to solve all test cases in both the training and validation sets to be considered a "perfect" solution. All training and validation sets can be found in the supplemental material [5].

### *1.3.3 Experimental Design*

We conducted five experiments: (1) we replicated a previous experiment [13] to evaluate subsampling's effect on lexicase selection's problem-solving success; (2) we tested whether or not subsampling improves problem-solving success because it facilitates deeper evolutionary searches; (3) we evaluated whether subsampling can reduce the computational effort expended by lexicase selection to solve problems; (4) we tested the effect of random subsampling on lexicase selection, comparing the diversity maintenance of standard, down-sampled, and cohort lexicase; (5) we compared each of standard, down-sampled, and cohort lexicase's capacity to maintain specialist candidate solutions (i.e., programs with low aggregate fitness that solve test cases that the majority of the population fails).

#### 1.3.3.1 Does Subsampling Improve Lexicase Selection's Problem-Solving Success Given a Fixed Computation Budget?

First, we replicated the experiment conducted in Hernandez et al. [13] where both down-sampled and cohort lexicase improved problem-solving success relative to standard lexicase selection. To evaluate whether subsampling improves lexicase's problem-solving success, we evolved programs using down-sampled, cohort, and standard lexicase selection to solve each of the four program synthesis problems (described in Sect. 1.3.2). While the sets of program synthesis problems are not

identical, the main difference between the two experiments is that our previous work included a test case that was designed to minimize program size of candidate solutions that solved all normal test cases; this minimizing test case was discarded for all experiments in this work. For a control, we also tested *reduced lexicase*: standard lexicase performed on a statically reduced training set that was randomly sampled at the beginning of the run. Reduced lexicase is similar to down-sampled lexicase, with the exception that test cases remain constant throughout the evolutionary search and are not sampled every generation.

All three of these lexicase variants were tested at five subsampling levels: 100% (identical to standard lexicase), 50, 25, 10 and, 5% ($D = 1, 2, 4, 10$, and 20, respectively). For standard lexicase and each variant, we limited each instance to a maximum computation budget of 30,000,000 evaluations.[1] Thus, standard lexicase ran for 300 generations, and the subsampled variants ran for 300, 600, 1200, 3000, and 6000 generations, respectively. We compared the problem-solving success (i.e., the number of replicates that produced a perfect solution) of each variant to standard lexicase. For each problem, we ran 50 replicates (each with a unique random seed) of each subsampled configuration, and 250 replicates (each with a unique random seed) of standard lexicase (50 replicates for each subsampling level).

#### 1.3.3.2 Does Subsampling Improve Lexicase Selection's Problem-Solving Success Because it Facilitates Deeper Searches?

Both down-sampled and cohort lexicase perform fewer test case evaluations per generation than standard lexicase, allowing us to run evolutionary searches for more generations given a fixed computation budget (i.e., a fixed number of total test case evaluations). We expected that subsampling improves lexicase's problem-solving success because it enables deeper searches. To test this hypothesis, we repeated the performance experiment (described previously in Sect. 1.3.3.1), except we evolved *all* populations (regardless of selection scheme and subsampling level) for 300 generations. We compared the number of successful replicates from each of down-sampled, cohort, and standard lexicase. If down-sampled and cohort lexicase lose their performance edge over standard lexicase, the distinction must come from the time after the 300 generation limit that they would have continued evolving. This finding would suggest that subsampling's improved problem-solving success results from its facilitation of deeper evolutionary searches.

[1] Evaluating a single program on a single test case is one test case evaluation.

#### 1.3.3.3 Does Random Subsampling Reduce the Computational Effort Required to Solve Problems with Lexicase Selection?

Our previous work [13] shows that subsampling can improve lexicase selection's problem-solving success given a fixed computational budget. Here, we are interested in whether or not subsampling reduces the total computational effort required to find solutions; that is, do down-sampled and cohort lexicase generally find solutions using fewer total evaluations than standard lexicase selection? We evolved programs on the four program synthesis problems described previously (Sect. 1.3.2) using down-sampled, cohort, and standard lexicase (at a 10% subsampling level for down-sampled and cohort lexicase). For each condition, we ran 50 replicate populations. Because we wanted to compare how much computational effort it generally took for a particular selection scheme to solve a problem, we only used data from the first 25 replicates of each condition to solve the problem (i.e., the 25 replicates per condition that used the least computational effort). We also included truncated lexicase [28], another lexicase selection variant that works to reduce the rigidness in lexicase selection by limiting the number of test cases used in a selection event before a candidate solution is selected. Truncated lexicase also has the potential to reduce the computational effort needed to find solutions. For our truncated lexicase condition, we used a truncation level equal to 10% of the training set.

#### 1.3.3.4 Does Subsampling Degrade Lexicase Selection's Diversity Maintenance?

Part of lexicase selection's success is known to be the result of its effectiveness at diversity maintenance [4, 9, 24]. Subsampling, however, is likely to degrade diversity maintenance because it both reduces the total number of niches available each generation (i.e., there are fewer possible orderings of test cases) *and* decreases niche stability from generation to generation (i.e., the set of possible test case permutations changes every generation). Thus, we expected populations evolved using down-sampled and cohort lexicase selection to have lower overall diversity and more frequent selective sweeps (coalescence events) than those evolved with standard lexicase selection. Additionally, cohort lexicase inherently buffers populations against selective sweeps, slowing down the rate at which a lineage can take over a population by limiting competition each generation to within cohorts. As such, we expected cohort lexicase to have fewer selective sweeps (and thus more phylogenetic diversity) than down-sampled lexicase.

To test our hypotheses, we replicated the experiment in Sect. 1.3.3.1, running both subsampling lexicase variants (at a range of subsampling levels) and standard lexicase for 30,000,000 total evaluations. In these runs, we collected data on genotypic, phenotypic, and phylogenetic diversity. We measured genotypic and phenotypic diversity with the Shannon diversity index. To assess phylogenetic diversity, we used a suite of phylogenetic diversity metrics (see [3] for a review). After all replicates terminated, we analyzed the results of each of these diversity

measures *at the time solutions were found*.[2] Within each subsampling level, we compared cohort, down-sampled, and standard lexicase selection.

#### 1.3.3.5 Does Subsampling Reduce Lexicase Selection's Capacity to Maintain *specialists*?

Recent work Helmuth et al. [10] demonstrates lexicase's tendency to select specialist individuals (i.e., individuals that have a low aggregate fitness but perform well on a subset of tests that the majority of the population fails). Helmuth et al. found that lexicase's ability to select specialists is a major driver behind its problem-solving success. Just as we expected subsampling to degrade lexicase selection's diversity maintenance, we also expected subsampling to inhibit specialist maintenance. Because specialists perform well on few test cases (and potentially poorly on the rest), a specialist's likelihood of being selected by lexicase selection is reduced if any of the test cases it passes are not sampled. Thus, we hypothesized that both down-sampled and cohort lexicase reduce lexicase selection's capacity to maintain specialist individuals.

To test our hypothesis, we investigated the extreme case of populations with a single specialist. We generated hypothetical populations, each containing a 'specialist' and many 'generalists'. In each generated population, the specialist individual was able to solve only one focal test case, and none of the generalists were allowed to solve the focal test case. We varied the probability at which generalists could solve each non-focal test case, ranging from 0.1 to 1.0 (where all generalists solved all non-focal test cases). We also varied population size and the total number of test cases. Table 1.1 shows all parameter values used in this experiment. We generated 100 populations for each combination of these parameters.

**Table 1.1** Generated population configurations

| Parameter | Values |
|---|---|
| Population size | 10, 20, and 100 |
| # test cases | 10, 20 |
| Generalist pass rate on non-focal tests | 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 |

We generated 100 populations for all combinations of the parameters given in this table

[2] Choosing when to measure diversity in evolutionary computation is an interesting problem. In evolutionary computation, diversity maintenance is often viewed as a mechanism to avoid premature convergence on suboptimal solutions. If our goal is to compare how well different selection schemes maintain diversity, *when* should we measure diversity? Measuring diversity *after* a global solution is found is not particularly meaningful, as finding the solution often causes the population to converge, decreasing diversity. We measured diversity at the time the solution is found to mitigate this problem. However, this solution only partially addresses the underlying problem: the process of evolution often involves many selective sweeps and subsequent divergences and we cannot know where in this cycle our measurements occurred.

For each population, we calculated the probability of each candidate solution being selected *at least once* to be a parent in the next generation under standard, down-sampled, and cohort lexicase selection. For standard lexicase selection, we calculated exact probabilities: we enumerated all possible orderings of test cases, counting the number of enumerations where each candidate solution is selected. This is intractable for the subsampled lexicase variants, so we took a sampling approach. To approximate the selection probability in the lexicase variants, we randomly subsampled the population according to the selection scheme being tested. After subsampling, down-sampled lexicase is equivalent to standard lexicase with fewer test cases, while cohort lexicase is equivalent to standard lexicase conducted separately on each cohort. Thus, we calculated the selection probabilities for each candidate solution with that particular random subsampling. This process was repeated 100,000 times to approximate the true selection probabilities under down-sampled and cohort lexicase. These calculations allowed us to compare the specialist's selection probability across configurations.

### *1.3.4 Statistical Analyses*

All statistics were calculated using the R statistical computing language v3.6.0 [26], and all figures in this work were created using the ggplot2 R package [31]. We compared problem-solving success rates among different independent conditions using Fisher's exact tests, and we corrected for multiple comparisons using the Holm–Bonferroni method where appropriate. For measures of computational effort and diversity, we performed a Kruskal–Wallis test to look for statistically significant differences among independent conditions. For comparisons in which the Kruskal–Wallis test was significant (significance level of 0.05), we performed a post-hoc Mann–Whitney test between relevant conditions (with a Holm–Bonferonni correction for multiple comparisons where appropriate). Statistical analyses for the specialist experiment also used a Kruskal–Wallis test, but swapped the Mann–Whitney test for a Wilcoxon test because the data were paired. Analysis and visualizations scripts can all be found in the supplemental material [5].

## 1.4 Results and Discussion

### *1.4.1 Subsampling Improves Lexicase Selection's Problem-Solving Success*

Figure 1.1 shows the fraction of replicates where a perfect solution evolved within 30,000,000 evaluations under each of down-sampled, cohort, reduced, and standard lexicase selection. For each program synthesis problem, we conducted a Fisher's

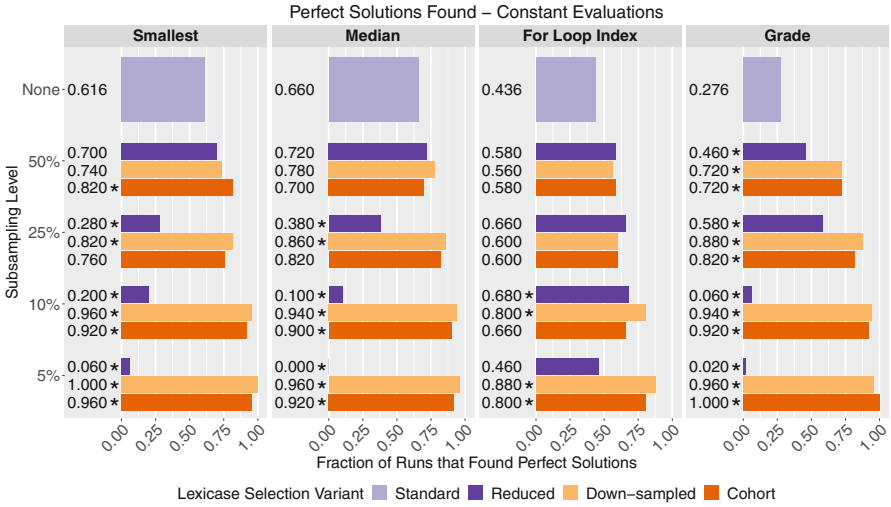


**Fig. 1.1** Problem-solving success after 30,000,000 evaluations. Bars show the fraction of replicates that found a perfect solution. An asterisk (*) to the left of a bar denotes a significant difference compared to the standard lexicase results (using a Holm–Bonferroni correction for multiple comparisons). Results for standard lexicase (light purple) consist of 250 replicates per problem, while results for reduced lexicase (dark purple), down-sampled lexicase (yellow), and cohort lexicase (orange) consist of 50 replicates for each configuration

exact test (0.05 significance level) between the 250 standard lexicase replicates and the 50 subsampled replicates of each experimental condition; we corrected for multiple comparisons using the Holm–Bonferonni method.

Our data are largely consistent with previous work [13]. For three of the four problems (Smallest, Median, and Grade), statically reducing the training set beyond a critical threshold significantly decreased problem-solving success. For example, at 5 and 10% subsampling levels, reduced lexicase performs significantly worse than standard lexicase in each of the Smallest, Median, and Grade problems. Reduced lexicase rarely outperformed standard lexicase, only doing so in three cases: Grade at 25 and 50% subsampling, and For Loop Index at 10% subsampling. Statically reducing the size of the training set did not inhibit our capacity to solve the For Loop Index problem; we suspect this is because the training set (100 test cases) is much larger than necessary. The same trend is true for 50- and 25%-reduced lexicase on the Grade problem.

Both down-sampled and cohort lexicase performed significantly better than standard lexicase on at least one subsampling level for every problem. Specifically, down-sampled lexicase significantly outperformed standard lexicase on all problems at the 5 and 10% subsampling levels, while cohort lexicase also outperformed standard lexicase at 5 and 10% subsampling on all problems except For Loop Index at the 10% subsampling level. Neither down-sampled nor cohort lexicase performed significantly worse than standard lexicase in any experimental configuration.

These results achieved better performance on more extreme subsampling levels than in [13]; this is because we removed all selection pressure to reduce program size. In this previous work, we included a single test case that favored small programs that only took effect when a program solved all other test cases *it was evaluated against*. At high subsampling levels (e.g., 5%), it is easy for programs that do not generalize well to prematurely trigger this size-minimization test case, which negatively impacted problem-solving success rates.

These results support our previous claim that subsampling can improve lexicase selection's problem-solving success. Although there is evidence that subsampling can improve solution rates, a different approach is needed to tease apart *why* this difference exists, or how down-sampled and cohort lexicase actually differ.

### *1.4.2 Deeper Evolutionary Searches Contribute to Subsampling's Success*

Figure 1.2 shows the fraction of replicates where a perfect solution evolved after 300 generations under each of down-sampled, cohort, and standard lexicase selection. After 300 generations, conditions with aggressive subsampling (e.g., 5%) have made fewer total evaluations than conditions with milder subsampling (e.g., 50%) or standard lexicase. To be exact, 50, 25, 10, and 5% subsampling complete 15,000,000,

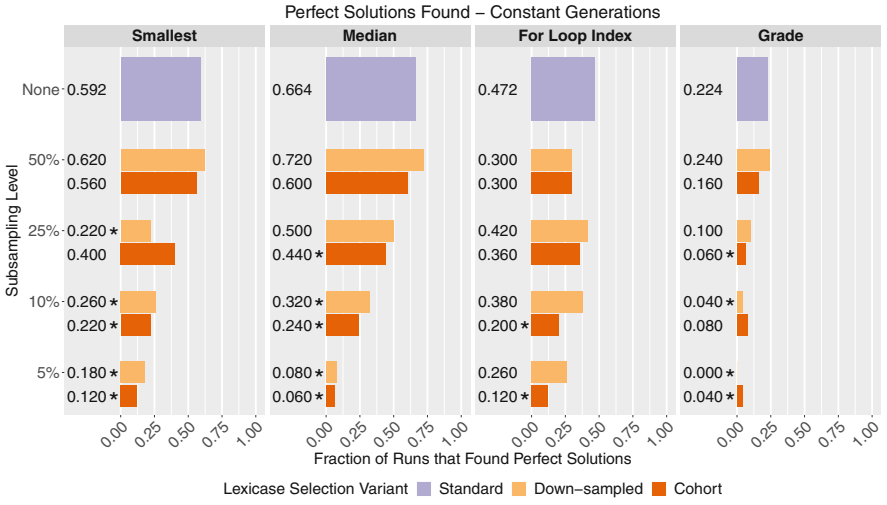


**Fig. 1.2** Evolutionary results at the end of 300 generations. Bars show the fraction of replicates that found a perfect solution on or before 300 generations. An asterisk (*) to the left of a bar denotes significant difference compared to the standard lexicase results. Results for standard lexicase (light purple) consist of 250 replicates per problem, while results for down-sampled lexicase (yellow) and cohort lexicase (orange) consist of 50 replicates for each experimental configuration

7,500,000, 3,000,000, and 1,500,000 evaluations, respectively. We hypothesized that random subsampling improves lexicase selection because it allows evolutionary searches to run for more generations given a fixed evaluation budget. By terminating all replicates after 300 generations, we expected subsampling to lose its advantage over standard lexicase.

Given a fixed number of generations, neither down-sampled nor cohort lexicase significantly outperformed standard lexicase at any subsampling level. In fact, down-sampled and cohort lexicase performed significantly worse than standard lexicase on all problems with 5 and 10% subsampling rates except in three cases: cohort at 10% subsampling on Grade, down-sampled at 10 and 5% subsampling on For Loop Index.

As shown in Sect. 1.4.1, when given equivalent computational budgets (i.e., total number of training case evaluations), subsampling significantly improves lexicase's problem-solving success. However, this experiment shows that when we restrict down-sampled and cohort lexicase to the same number of *generations* as standard lexicase, they both have significantly diminished success on the same problems. These data support our hypothesis that deeper evolutionary searches contribute to the success of the subsampled variations on lexicase selection.

### *1.4.3 Subsampling Reduces Computational Effort*

Next, we explored how subsampling affects the amount of computational effort required to solve problems in the context of lexicase selection. For this experiment, we removed all evaluation and generation termination criteria. Figure 1.3 shows the number of test case evaluations in each of the first 25 replicates for each condition in which a solution evolved (i.e., the 25 replicates that required the least computational effort to solve the problem). We performed a Kruskal–Wallis test (significance level 0.05) to look for significant differences among selection schemes for each program synthesis problem. For problems in which the Kruskal–Wallis test was significant,

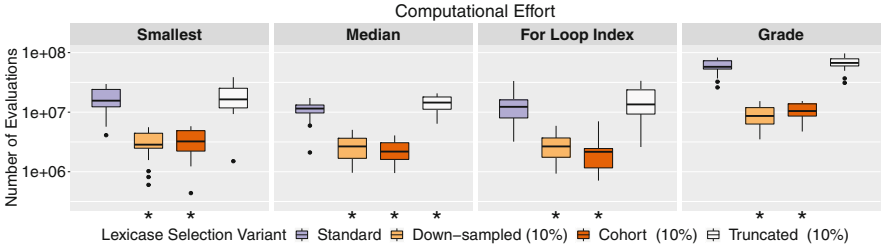


**Fig. 1.3** The number of evaluations required for each treatment to solve the specified problems. The 25 replicates with the fewest evaluations for each treatment are shown. An asterisk (*) under a box denotes significant difference between that treatment and standard lexicase

we performed a post-hoc Mann–Whitney test between standard lexicase and each of the down-sampled, cohort, and truncated lexicase (with a Holm–Bonferonni correction for multiple comparisons).

Both down-sampled and cohort lexicase used significantly fewer evaluations than standard lexicase on all four problems. Across all problems, truncated lexicase did not use significantly fewer evaluations than standard lexicase; on the Median problem, truncated lexicase actually used significantly *more* evaluations than standard lexicase. The data show a clear trend that 10% subsampling, whether via down-sampling or cohorts, can significantly reduce the number of evaluations needed to solve these program synthesis problems. However, truncated lexicase (using 10% of the training cases per selection event) causes either no effect or a significant increase in required evaluations.

### *1.4.4 Subsampling Does Not Systematically Decrease Phenotypic Diversity in Lexicase Selection*

Mutations to the binary tags used by the programs to reference modules and memory are often silent (i.e., the phenotype and fitness remain the same) allowing populations to endure high mutation rates that drive adaptive evolution. As a result, almost all replicates maximize genotypic diversity, rendering comparisons uninformative. Therefore, we examined the phenotypic diversity of lexicase and the two subsampled variants.

When evolution produced a candidate solution capable of solving all test cases in the training set, we immediately tested that solution on the cases in the reserved validation set as well. If this candidate solution continued to pass all test cases, we declared it a "perfect solution" and proceeded to measure the phenotypic diversity of the population it arose from. To do so, we tested all programs in the population on all test cases across both the training and validation sets. We designated each candidate solution's performances (in sequence) on all test cases as that solution's phenotype. Figure 1.4 shows the Shannon diversity of these results.

Minimal evidence was found to support our hypothesis that subsampling results in a reduction of phenotypic diversity. After comparing the phenotypic diversity of both down-sampled and cohort lexicase to the standard algorithm, only 2 of 32 configurations resulted in a significant decrease in phenotypic diversity, both of which were down-sampled configurations. Conversely, cohort lexicase actually had significantly *higher* phenotypic diversity than standard lexicase in two configurations. Further, cohort lexicase results had a significantly higher phenotypic diversity than down-sampled lexicase in 4 of 16 comparisons.

With only two configurations leading to decreased phenotypic diversity, we cannot conclude that there is a systematic decrease in phenotypic diversity due to subsampling for these program synthesis problems. However, these results hint

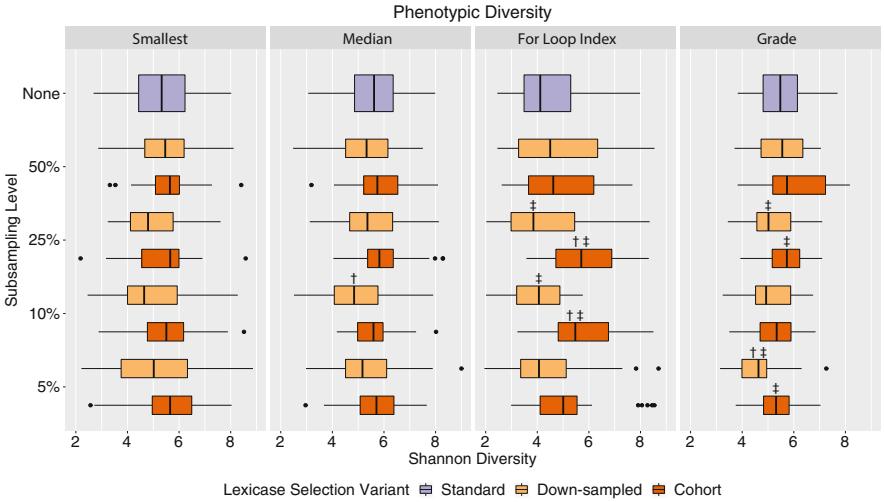


**Fig. 1.4** Shannon diversity of candidate solution phenotypes at the first generation a perfect solution was found; individual phenotypes were measured as a program's performance on each test from the training and validation sets. A dagger (†) above a box denotes significant difference with standard lexicase. A double dagger (‡) denotes significant difference between cohort lexicase and down-sampled lexicase at that subsampling level. Results consist of replicates that found a perfect solution out of 250 replicates for standard lexicase on each problem (purple boxes) and 50 replicates for each combination of problem and subsampling level for down-sampled lexicase (yellow boxes) and cohort lexicase (orange boxes)

at a difference between diversity due to down-sampled lexicase and cohort lexicase; we plan to explore this difference in future work.

### *1.4.5 Cohort Lexicase Enables More Phylogenetic Diversity Than Down-Sampled Lexicase*

As with phenotypic diversity, we recorded the phylogenetic diversity metrics at the time point when populations first found a perfect solution. This timing was necessary; the discovery of a perfect solution is likely to produce a selective sweep, radically altering the structure of the phylogeny. An unavoidable side effect is that the measurements are taken after different numbers of generations have elapsed in different replicates. This discrepancy is potentially concerning, as phylogenetic diversity measurements are sensitive to the number of generations represented within the phylogeny. Adding more generations will, in many cases, legitimately increase the diversity of evolutionary history that a population contains. However, the number of generations elapsed can have a disproportionately large effect on a phylogenetic diversity metric, swamping out other effects. In this case, it is these

other effects that we are most interested in, as we have already analyzed the causes and effects of the number of generations a population goes through. Fortunately, our results comparing down-sampled vs. cohort lexicase do not appear to be driven by variation in the number of generations elapsed, as the distribution of generations at which the first perfect solution was found did not vary consistently *within* any subsampling level. Because this distribution did vary *among* subsampling levels, we are not attempting to make any strong claims about the relationship between phylogenetic diversity and degree of subsampling. Here we examine only two of the pyhlogenetic metrics that were calculated; plots, descriptions, and statistics of all recorded metrics can be found in the supplemental material [5].

The most recent common ancestor (MRCA) is the most recently evolved candidate solution from which all extant candidate solutions descend. For this experiment we tracked the MRCA throughout the evolutionary search, and we examined the number of selective sweeps by counting the number of times the MRCA changed (see Fig. 1.5). For all problems tested, cohort lexicase has significantly fewer MRCA changes than down-sampled lexicase for 5, 10, and 25% subsampling levels. This pattern suggests that cohort lexicase inhibits selective sweeps in a way that down-sampled lexicase does not. A likely mechanism for this behavior is that, by explicitly fragmenting the population into groups, cohort lexicase prevents any single candidate solution from sweeping more than one cohort per generation.

Another phylogenetic measure we examined was the phylogenetic divergence (i.e., how distinct the extant taxa are from each other) [3]. Here we quantify

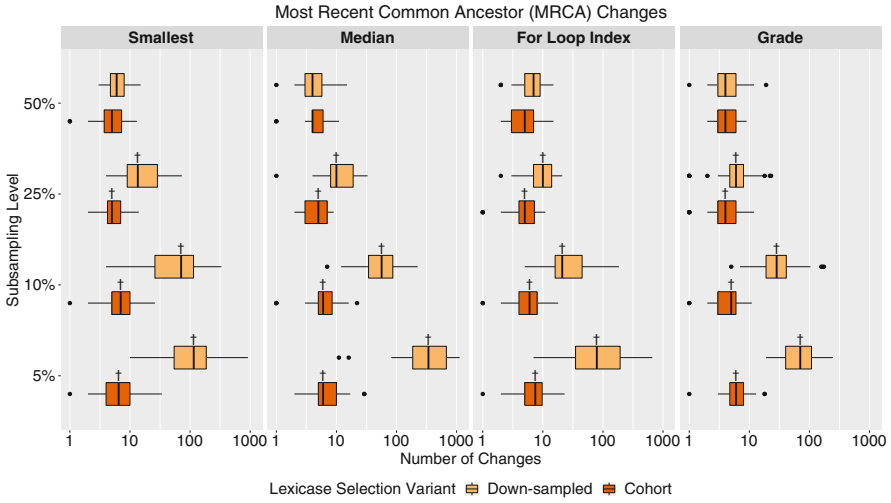


**Fig. 1.5** Number of times the most recent common ancestor (MRCA) of all extant candidate solutions changed for each evolutionary run. Changes shown on a logarithmic scale. A dagger (†) above a box denotes significant difference between cohort lexicase and down-sampled lexicase at that subsampling level. All results shown are from the replicates that found a perfect solution out of 50 replicates per experimental condition

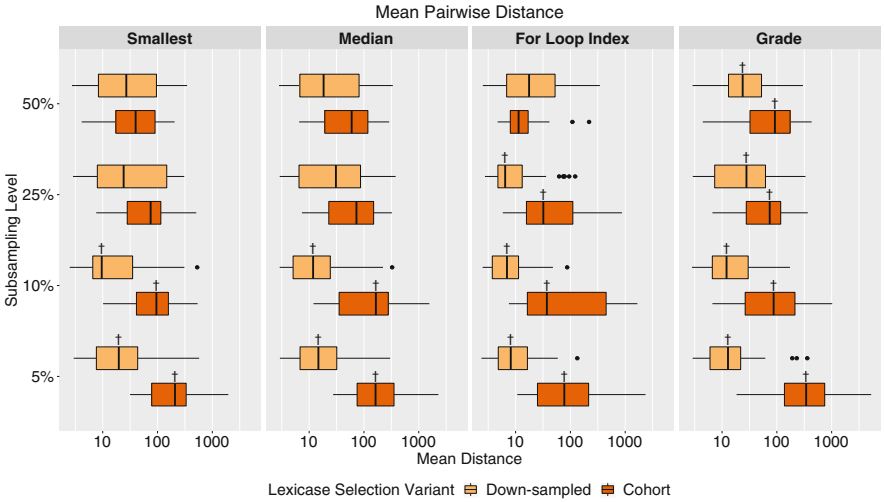


**Fig. 1.6** Mean distance between all pairs of extant taxa in the phylogenetic tree for runs of both subsampled lexicase variants at different subsampling levels. A dagger (†) above a box denotes significant difference between cohort lexicase and down-sampled lexicase at that subsampling level. All results shown consist of the replicates that found a perfect solution out of 50 replicates per experimental condition

phylogenetic divergence via mean pairwise distance of the extant solutions in the phylogeny. This metric is calculated as the average distance in the phylogenetic tree between each pair of extant candidate solutions (see Fig. 1.6) [30]. Cohort lexicase has significantly higher mean pairwise distance than down-sampled lexicase for all problems at the 5 and 10% subsampling levels. This result indicates that cohort lexicase has significantly higher phylogenetic divergence than down-sampled lexicase, providing further evidence that cohort lexicase is better than down-sampled lexicase at maintaining phylogenetic diversity. Other phylogenetic diversity metrics were consistent with these results.

Because the differing generation counts prevent us from meaningfully comparing phylogenetic diversity across subsampling levels, all we can say conclusively is that subsampling does not appear to decrease phylogenetic diversity. That said, it may well be the case that greater phylogenetic diversity helps produce better candidate solutions. If so, this factor could explain why more generations (as opposed to more evaluation thoroughness) increases the computational efficiency of lexicase selection. A more targeted investigation will be required to determine how important phylogenetic diversity is to the success of lexicase selection variants.

### 1.4.6 *Subsampling Degrades Specialist Maintenance*

Across experimental conditions, lexicase selection has a significantly higher probability of selecting the specialist than either subsampled variant (see Fig. 1.7). This result supports our hypothesis that subsampling degrades specialist preservation. Interestingly, down-sampled and cohort lexicase behave differently across the conditions. Exploring these differences can help us better understand the mechanisms that cause a lexicase variant to favor specialists.

When population size is large, down-sampled and cohort lexicase behave nearly identically. At higher subsampling rates specialists have a higher survival probability in both treatments. At smaller population sizes, higher subsampling rates continue to demonstrate a higher survival probability of specialists in down-sampled lexicase, but not always in cohort lexicase.

At the extreme, when population size, subsampling rate, and generalist pass rate are all small, cohort lexicase has a drastically higher probability of specialist survival than down-sampled lexicase. In this case, the specialist benefits from the low generalist pass rate, since many non-specialists will fail to solve many of the test cases. Specifically, if *all* candidate solutions competing against the specialist fail a given test case, it will be non-discriminatory and effectively ignored. This effect is more pronounced in cohort lexicase, when the specialist is competing only within


**Fig. 1.7** Bars show the median probability that a focal specialist will be selected as a parent in the next generation *at least once*; data are aggregated over 100 experimental populations. Error bars show the minimum and maximum probabilities across all populations for that configuration. The dashed lines show the expected probability for both subsampled lexicase variants for configurations where population size is 100. An asterisk (*) denotes a significant difference between cohort lexicase and down-sampled lexicase; standard lexicase was always significantly different. All configurations shown are for 20 test cases

its cohort (e.g., a cohort of size 2 for a population size of 20 with 10% subsampling), rather than the full population. At a population size of 100, this benefit is lessened because cohorts still contain a relatively large number of candidate solutions. In the remaining configurations, down-sampled lexicase has a higher probability of specialist survival than cohort lexicase.

To better understand these probabilities, consider a situation with two constraints: (1) the specialist solves only its one assigned test case, and (2) every other candidate solution can solve all test cases but the specialist's (i.e., the generalist pass rate is 1.0). While the situation is improbable, it is the worst-case scenario for selecting the specialist; relaxing either constraint could only increase the chance of selecting the specialist. In this situation, the specialist's odds of selection *in a single selection event* under lexicase selection is $\frac{1}{T}$ where $T$ is the number of test cases; that is, the probability of its focal test case being chosen first. The specialist's probability of selection for the entire next generation can be expressed as Eq. 1.1 where $N$ is the total population size [4] (for further discussion of selection probabilities under full lexicase selection, see [15]).

$$P_{lexicase} = 1 - (1 - \frac{1}{T})^N \tag{1.1}$$

We can modify Eq. 1.1 to accommodate down-sampled lexicase by accounting for two cases. First, the specialist's sole test case can be included in the test cases used for this generation, in which case the specialist has a $\frac{D}{T}$ chance of being selected (recall $D$ is the down-sample factor, which divides the number of training cases such that each organism sees $\frac{1}{D}$ of the full training set each generation). Otherwise the specialist's test case is not included, and the specialist has no chance of being selected. Thus, we arrive at Eq. 1.2.

$$P_{down\text{-}sampled} = \frac{1 - (1 - \frac{D}{T})^N}{D} \tag{1.2}$$

Finally, we can also account for cohort lexicase selection. Cohort lexicase also gives the specialist a $\frac{1}{D}$ chance of being evaluated against its sole test case. The only difference is in the number of selection events; cohort lexicase can be thought of as standard lexicase being conducted on each cohort. Thus, in the case where the specialist is in the same cohort as its test case, it does not have $N$ selection events to be selected, but instead $\frac{N}{D}$. This gives us the final equation, Eq. 1.3.

$$P_{cohort} = \frac{1 - (1 - \frac{D}{T})^{\frac{N}{D}}}{D} \tag{1.3}$$

Plotting these equations, we can see both that down-sampled and cohort lexicase approach a maximum specialist survival probability of $\frac{1}{D}$, and that down-sampled approaches that limit at lower population sizes than cohort lexicase (see Fig. 1.8). The plots also show that increasing the number of training cases increases the

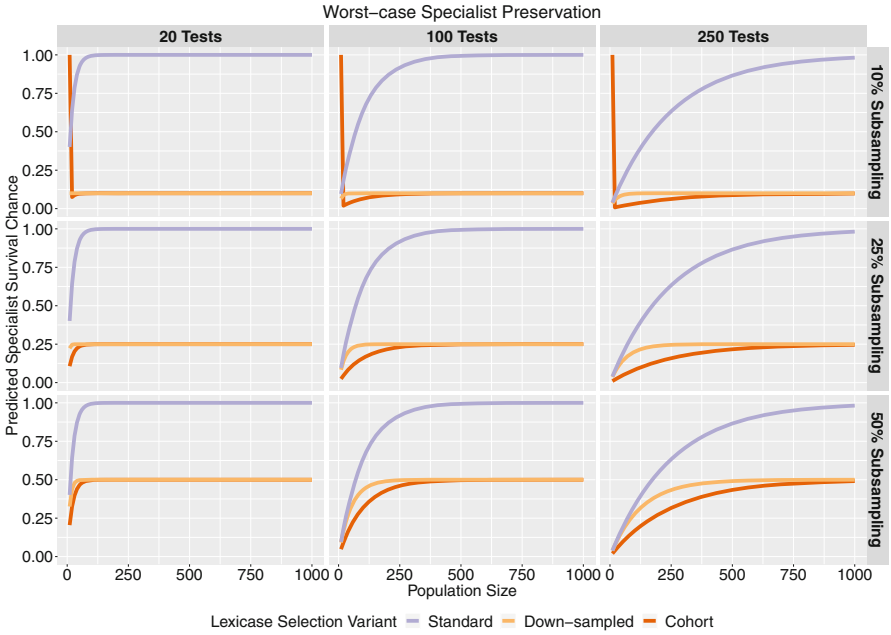


**Fig. 1.8** Probabilities that the focal specialist will be selected to be a parent in the next generation *at least once* in the situation where there is one specialist, which solves only one test case, but is also the only candidate solution to solve that specific test case. Meanwhile, all other candidate solutions solve all other test cases. Note the special case of a population size of 10 with 10% subsampling. Here, each cohort has one solution, which guarantees selection exactly once with no selective pressure

required population size to reach the $\frac{1}{D}$ limit. Thus the two subsampled lexicase variants have the same maximum specialist selection probability, but smaller populations will see a lower value for cohort lexicase. These theoretical findings help explain our empirical results.

Again, this is the worst-case scenario for the specialist. Further work is needed to see how specialist preservation changes under different situations (e.g., more copies of the specialist, less elite generalists, specialists that solve more than one test case, etc.). Figure 1.8 shows only the lower bound on the specialist selection probability.

## 1.5 Conclusion

Here, we investigated the effects of random subsampling on lexicase selection. We replicated previous results [13], demonstrating that subsampling improves lexicase's problem-solving success, and we have shown that subsampling's success is a result of it enabling deeper evolutionary searches (i.e., running searches for more genera-

tions). Moreover, we have shown that subsampling reduces the total computational effort required to evolve solutions in the context of lexicase selection. We expected that applying subsampling to lexicase selection would degrade phenotypic diversity, but have found no evidence of systematic degradation. However, we did find evidence that cohort lexicase is better at generating and preserving phylogenetic diversity than down-sampled lexicase. Finally, we have shown that subsampling does reduce lexicase's capacity to maintain specialist individuals.

Overall, our results highlight the value of random subsampling in lexicase selection, showing that it can improve problem-solving success and save computational effort. However, we also demonstrate that subsampling degrades specialist preservation, and as such, for problems where maintaining specialists is especially important, subsampling might have an overall negative effect on problem-solving success. Future work should explore how subsampling affects both overall population diversity and specialist maintenance at a fine-grained scale and on a wider range of problem types.

**Acknowledgements** This research was supported by the National Science Foundation through the BEACON Center (Coop. Agreement No. DBI-0939454), a Graduate Research Fellowship to AL (Grant No. DGE-1424871), and Grant No. DEB-1655715 to CO. Michigan State University provided computational resources through the Institute for Cyber-Enabled Research.

## References

1. Aenugu, S., Spector, L.: Lexicase selection in learning classifier systems. In: Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2019, pp. 356–364. ACM Press, Prague, Czech Republic (2019)
2. Curry, R., Heywood, M.: Towards efficient training on large datasets for genetic programming. In: A. Tawfik, S. Goodwin (eds.) Conference of the Canadian Society for Computational Studies of Intelligence, pp. 161–174. Springer (2004)
3. Dolson, E., Lalejini, A., Jorgensen, S., Ofria, C.: Quantifying the tape of life: Ancestry-based metrics provide insights and intuition about evolutionary dynamics. In: Artificial Life Conference Proceedings, pp. 75–82. MIT Press (2018)
4. Dolson, E.L., Banzhaf, W., Ofria, C.: Ecological theory provides insights about evolutionary computation. preprint, PeerJ Preprints (2018). URL https://peerj.com/preprints/27315
5. Ferguson, A.: FergusonAJ/gptp-2019-subsampled-lexicase: GPTP Chapter Companion (2020). https://doi.org/10.5281/zenodo.3679380, https://github.com/FergusonAJ/gptp-2019-subsampled-lexicase
6. Forstenlechner, S., Fagan, D., Nicolau, M., O'Neill, M.: Towards Understanding and Refining the General Program Synthesis Benchmark Suite with Genetic Programming. In: 2018 IEEE Congress on Evolutionary Computation (CEC), pp. 1–6. IEEE, Rio de Janeiro (2018)
7. Gathercole, C., Ross, P.: Dynamic training subset selection for supervised learning in Genetic Programming. In: Y. Davidor, H.P. Schwefel, R. Maenner (eds.) Parallel Problem Solving from Nature - PPSN III, vol. 866, pp. 312–321. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
8. Gonçalves, I., Silva, S., Melo, J.B., Carreiras, J.M.: Random sampling technique for overfitting control in genetic programming. In: A. Moraglio, S. Silva, K. Krawiec, P. Machado, C. Cotta (eds.) European Conference on Genetic Programming

9. Helmuth, T., McPhee, N.F., Spector, L.: Effects of lexicase and tournament selection on diversity recovery and maintenance. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, pp. 983–990. ACM (2016)
10. Helmuth, T., Pantridge, E., Spector, L.: Lexicase selection of specialists. In: Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO 2019, pp. 1030–1038. ACM Press, Prague, Czech Republic (2019)
11. Helmuth, T., Spector, L.: General program synthesis benchmark suite. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pp. 1039–1046. ACM (2015)
12. Helmuth, T., Spector, L., Matheson, J.: Solving uncompromising problems with lexicase selection. IEEE Transactions on Evolutionary Computation **19**(5), 630–643 (2015)
13. Hernandez, J.G., Lalejini, A., Dolson, E., Ofria, C.: Random Subsampling Improves Performance in Lexicase Selection. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, pp. 2028–2031. ACM, New York, NY, USA (2019). Event-place: Prague, Czech Republic
14. Hmida, H., Hamida, S.B., Borgi, A., Rukoz, M.: Sampling Methods in Genetic Programming Learners from Large Datasets: A Comparative Study. In: P. Angelov, Y. Manolopoulos, L. Iliadis, A. Roy, M. Vellasco (eds.) Advances in Big Data, vol. 529, pp. 50–60. Springer International Publishing, Cham (2017)
15. La Cava, W., Helmuth, T., Spector, L., Moore, J.H.: A Probabilistic and Multi-Objective Analysis of Lexicase Selection and $\epsilon$-Lexicase Selection. Evolutionary Computation **27**, 377–402 (2018)
16. La Cava, W., Spector, L., Danai, K.: Epsilon-Lexicase Selection for Regression. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO 2016, pp. 741–748. ACM, New York, NY, USA (2016). Event-place: Denver, Colorado, USA
17. Lalejini, A., Ofria, C.: Evolving event-driven programs with SignalGP. In: Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO 2018, pp. 1135–1142. ACM Press, Kyoto, Japan (2018)
18. Lalejini, A., Ofria, C.: Tag-accessed memory for genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion - GECCO 2019, pp. 346–347. ACM Press, Prague, Czech Republic (2019)
19. Lalejini, A., Wiser, M.J., Ofria, C.: Gene duplications drive the evolution of complex traits and regulation. In: Artificial Life Conference Proceedings 14, pp. 257–264. MIT Press (2017)
20. Martinez, Y., Naredo, E., Trujillo, L., Legrand, P., Lopez, U.: A comparison of fitness-case sampling methods for genetic programming. Journal of Experimental & Theoretical Artificial Intelligence **29**, 1203–1224 (2017)
21. Melo, V.V., Vargas, D.V., Banzhaf, W.: Batch Tournament Selection for Genetic Programming. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion - GECCO 2019, pp. 994–1002. ACM Press, Prague, Czech Republic (2019)
22. Metevier, B., Saini, A.K., Spector, L.: Lexicase selection beyond genetic programming. In: W. Banzhaf, L. Spector, L. Sheneman (eds.) Genetic Programming Theory and Practice XVI, pp. 123–136. Springer International Publishing, Cham (2019)
23. Moore, J.M., Stanton, A.: Lexicase selection outperforms previous strategies for incremental evolution of virtual creature controllers. In: Proceedings of the 14th European Conference on Artificial Life ECAL 2017, pp. 290–297. MIT Press, Lyon, France (2017)
24. Moore, J.M., Stanton, A.: Tiebreaks and Diversity: Isolating Effects in Lexicase Selection. In: The 2018 Conference on Artificial Life, pp. 590–597. MIT Press, Tokyo, Japan (2018)
25. Moore, J.M., Stanton, A.: The Limits of Lexicase Selection in an Evolutionary Robotics Task. In: The 2019 Conference on Artificial Life, pp. 551–558. MIT Press, Newcastle, United Kingdom (2019)
26. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2019). URL https://www.R-project.org/

27. Spector, L.: Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. In: Proceedings of the 14th annual conference companion on Genetic and evolutionary computation, pp. 401–408. ACM (2012)
28. Spector, L., Cava, W.L., Shanabrook, S., Helmuth, T., Pantridge, E.: Relaxations of Lexicase Parent Selection. In: W. Banzhaf, R.S. Olson, W. Tozier, R. Riolo (eds.) Genetic Programming Theory and Practice XV, pp. 105–120. Springer International Publishing, Cham (2018)
29. Spector, L., Martin, B., Harrington, K., Helmuth, T.: Tag-based modules in genetic programming. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO 2011, p. 1419. ACM Press, Dublin, Ireland (2011)
30. Webb, C.O.: Exploring the phylogenetic structure of ecological communities: an example for rain forest trees. The American Naturalist **156**(2), 145–155 (2000)
31. Wickham, H.: ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York (2016). URL https://ggplot2.tidyverse.org