Explaining and Exploiting the Advantages of Down-sampled Lexicase Selection

Thomas Helmuth[1] and Lee Spector[2,3,4]

[1]Hamilton College, Clinton, NY 13323
[2]Amherst College, Amherst, MA 01002
[3]Hampshire College, Amherst, MA 01002
[4]University of Massachusetts, Amherst, MA 01003
thelmuth@hamilton.edu

Abstract

In genetic programming, parent selection is ordinarily based on aggregate measures of performance across an entire training set. Lexicase selection, by contrast, selects on the basis of performance on random sequences of test cases; this has been shown to enhance problem-solving power in many circumstances. Lexicase selection can also be seen as better reflecting biological evolution, by modeling sequences of challenges that organisms face over their lifetimes. Recent work has demonstrated that the advantages of lexicase selection can be amplified by down-sampling, meaning that only a random subsample of the training cases is used each generation, which can also be seen as modeling environmental change over time. Here we provide the most extensive benchmarking of down-sampled lexicase selection to date, showing that its benefits hold up to increased scrutiny. The reasons that down-sampling helps, however, are not yet fully understood. Hypotheses include that down-sampling allows for more generations to be processed with the same budget of program evaluations; that the variation of training data across generations acts as a changing environment, encouraging adaptation; or that it reduces overfitting, leading to more general solutions. We systematically evaluate these hypotheses, finding evidence against all three, and instead draw the conclusion that down-sampled lexicase selection's main benefit stems from the fact that it allows GP to examine more individuals within the same computational budget, even though each individual is examined less completely.

Introduction

When used as a supervised learning technique, genetic programming (GP) defines a problem's specifications by a set of *training cases*. It then judges the ability of evolved programs to solve the problem based on running each program on each training case, measuring the distance between the program's output and the desired output. GP uses these error values to determine which individuals in the population it selects to reproduce, and how many children they will produce.

The interaction between a program and the training cases can be thought of as the interaction between a biological individual and challenges presented by its environment. Individuals that are better equipped to handle these challenges have better reproductive success, similarly to how programs that perform closer to the desired outputs should produce more children.

Lexicase parent selection mimics this analogy better than most other GP parent selection techniques, by having each parent selection test individuals on the set of training cases in a different random order, in essence exposing individuals to different challenges for each selection. However, between all of the parent selections in a generation, each training case will appear early in the random ordering for a few selections. In this sense, the environment remains static between generations, unlike environments that change over time in biological evolution.

Hernandez et al. (2019) recently proposed two methods for subsampling the training set each generation when using lexicase selection, which were further studied by Ferguson et al. (2019). Down-sampled lexicase selection uses a different random subsample of cases each generation. Cohort lexicase selection groups individuals into cohorts, and exposes each cohort to a different random subsample of the training cases. Both methods effectively change the environment from generation to generation by exposing individuals to different training cases. Both methods also crucially reduce the amount of computational effort required to evaluate each individual, since they only run each program on a subsample of the training cases. Results from Hernandez et al. (2019) and Ferguson et al. (2019) indicate that both of these methods improve problem-solving performance compared to standard lexicase selection.

In this paper we concentrate on down-sampled lexicase selection, as it is simpler in concept and implementation, and both Hernandez et al. (2019) and Ferguson et al. (2019) found its benefits to be comparable to cohort lexicase selection. We first conduct a more expansive benchmarking of down-sampled lexicase than has been conducted previously, using more benchmark problems and subsample sizes. These results confirm earlier findings that down-sampled lexicase selection produces substantial improvements over lexicase selection, and that it is robust to a range of subsample sizes.

We then turn to developing a better understanding of why

down-sampled lexicase selection performs so well. One hypothesis put forward by Ferguson et al. (2019) is that down-sampled lexicase selection's success hinges on it enabling deeper evolutionary searches for more generations given the same computational effort. We compare this hypothesis to the hypothesis that simply evaluating more individuals in the search space is more important than deeper evolution specifically. We conduct experiments using increased maximum generations and increased population sizes (with non-increased generations), and find that they perform commensurately, indicating that deeper evolutionary lineages are not crucial to down-sampled lexicase selection's success.

We then examine the idea that by randomly down-sampling, we change the environment encountered by individuals each generation. In biology, many theorists believe that changing environments play an important role in evolutionary adaptation and speciation (Levins, 1968). We hypothesize that changing the training cases on which down-sampled lexicase selection evaluates individuals each generation contributes to the evolvability of the system, resulting in improved performance. We test this hypothesis with an experiment that mimics down-sampled lexicase selection, except that it uses different training cases in every selection, meaning that every training case gains exposure each generation. The results of this experiment provide evidence against our hypothesis that "changing environments" are important for down-sampled lexicase selection, leading us to believe its main benefit is in increasing the number of individuals evaluated within the same computational budget.

One area where down-sampling (without lexicase selection) has proven useful is in avoiding overfitting and improving generalization, both in GP and in machine learning more generally. We explore the hypothesis that down-sampled lexicase selection's improved performance is driven by better generalization, and find that it does not hold up to the results of our experiments.

This paper continues as follows: we next discuss lexicase selection and subsampling of training cases in more detail, and then describe our experimental methods. We then benchmark down-sampled lexicase, followed by sections tackling each of the hypotheses described above, and end with our conclusions.

Related Work

Unlike many evolutionary computation parent selection methods, lexicase selection does not aggregate the performance of an individual into a single fitness value (Helmuth et al., 2015). Instead, it considers each training case separately, never conflating the results on different cases. We give pseudocode for the lexicase selection algorithm in Figure 1. After randomly shuffling the training cases, lexicase selection goes through them one by one, removing any individuals that do not give the best performance on each case until either a single individual or a single case remains. Lexicase selection has produced better performance than other parent selection methods in a variety of GP systems and problem domains (Helmuth et al., 2015; Helmuth and Spector, 2015; La Cava et al., 2018; Forstenlechner et al., 2017; Liskowski et al., 2015; Oksanen and Hu, 2017; Moore and Stanton, 2017, 2018).

Algorithm 1: Lexicase Selection *(to select a parent)*

```
Inputs: candidates, the entire population;
        cases, a list of training cases
Shuffle cases into a random order
loop
  Set first be the first case in cases
  Set best be the best performance of any individual in
      candidates on the first test case
  Set candidates to be the subset of candidates that
      have exactly best performance on first
  if |candidates| = 1 then
    Return the only individual in candidates
  end if
  if |cases| = 1 then
    Return a randomly selected ind. from candidates
  end if
  Remove the first case from cases
end loop
```

Hernandez et al. (2019) introduced down-sampled lexicase selection, a variant of lexicase selection that was developed further by Ferguson et al. (2019). Down-sampled lexicase selection aims to reduce the number of program executions used to evaluate each individual by only running each program on a random subsample of the overall set of training cases, which are changed each generation. This method reduces the per-individual computational effort, which can either either be saved for decreased runtimes, or can be allocated in other ways, such as increases in population size or maximum number of generations. In order to compare with methods that do not subsample the training cases, we take the latter approach, always comparing methods equitably by limiting their total program executions per GP run.

Others have used subsampling of training data in GP, for reducing computation per individual or for improving generalization. To our knowledge, the only other work that has combined subsampling with lexicase selection besides Hernandez et al. (2019) and Ferguson et al. (2019) is in evolutionary robotics, where subsampling is necessary for improving runtimes because of slow simulation speeds, though this research did not include comparisons with non-subsampled methods (Moore and Stanton, 2017, 2018). Outside of lexicase selection, subsampling has been used largely to reduce the computational load of evaluating each individual, especially when considering large datasets (Hmida et al., 2016; Martinez et al., 2017; Curry

and Heywood, 2004; Gathercole and Ross, 1994; Zhang and Joung, 1999). Others have proposed subsampling as a technique to reduce overfitting and improve generalization (Goncalves and Silva, 2013; Martinez et al., 2017; Schmidt and Lipson, 2008). Additionally, subsampling data is common in machine learning in general for similar reasons, as in stochastic gradient descent for improving generalization (Kleinberg et al., 2018).

The work we present here, along with that of Hernandez et al. (2019), Ferguson et al. (2019) and Moore and Stanton (2017), is novel in its application of subsampling when using lexicase selection, as well as applying subsampling to an already relatively small set of training data. To the latter point, many previous applications of subsampling aim to subsample a large set of example data (thousands or millions of cases) to a manageable size, say hundreds of cases. In our case, we start with a set of about 100-200 cases, and subsample to a set of 50 or less. When using a small set of n training cases, lexicase selection can select parents with at most $n!$ different error vectors, since this is the number of different shufflings of cases. When n is as small as 4 or 5, this limits selection to a small portion of the population, and often even less in practice. Lexicase selection typically requires 8 to 10 cases minimum to produce performance benefits, though others have successfully used it with as few as 4 cases (Moore and Stanton, 2017). With this in mind, it is not self-evident whether or not lexicase selection can maintain empirical benefits such as increased population diversity and problem-solving performance with such few cases.

Experimental Methods

To explore the effects of down-sampled lexicase selection, we use benchmark problems from the domain of automatic program synthesis, which previous studies of down-sampled lexicase selection have used (Hernandez et al., 2019; Ferguson et al., 2019). In particular, we use problems from the "General Program Synthesis Benchmark Suite" (Helmuth and Spector, 2015), which require solution programs to manipulate a variety of data types and control flow structures. These problems originate from introductory computer science textbooks, allowing us to test the ability of evolution to perform the same types of programming we expect humans to perform. We use a core set of 12 problems with a range of difficulties and requirements for many of our experiments, and expand that set to 26 problems (all of the problems from the suite that have been solved by at least one program synthesis system) for one experiment.

As in Helmuth and Spector (2015), we define each problem's specifications as a set of input/output examples, so that GP has no knowledge of the underlying problems besides these examples.[1] For each problem we use a small set of *training cases* to evaluate each individual (between 100 and 250 cases per run, see Table 2), and a larger set of unseen *test cases*, which are used to determine whether an evolved program that passes all of the training cases generalizes to unseen data. Before testing a potential solution for generalization, we use an automatic simplification procedure that has been shown to improve generalization (Helmuth et al., 2017); finding a simplified program that passes all of the unseen test cases is considered a successful GP run. We test the significance of differences in numbers of successes between sets of runs using a chi-square test with a 0.05 significance level, using Holm's correction for multiple comparisons whenever there are more than two methods run on a single problem in one experiment.

Table 1: PushGP system parameters.

Parameter	Value
population size	1000
max generations for runs using full training set	300
parent selection	lexicase
genetic operator	UMAD
UMAD addition rate	0.09

When a run using down-sampled lexicase selection finds a program that passes all of the subsampled training cases, we do not immediately terminate the run. Instead, we run the program on the full training set, and terminate the run if the program passes all of those cases. If it does not, we continue to the next generation, as the individual (or its children) may not pass some of the cases in the newly subsampled set of cases. Note that if only a single individual passes all cases in the subsampled training set but evolution continues, it will receive every single parent selection in that generation. These *hyperselection events* (Helmuth et al., 2016) may have strong effects on population diversity, a potential avenue for future study.

We evolve programs with the PushGP genetic programming system, which uses programs represented in the Push programming language (Spector et al., 2005; Spector and Robinson, 2002). Push was designed with genetic programming in mind, in particular to enable *autoconstruction*, in which evolving programs not only need to try to solve a problem, but are also run to produce their children (Spector and Robinson, 2002; Spector et al., 2016). Push programs utilize a handful of typed stacks, from which instructions pop their arguments and to which instructions push their results. Push programs can be any hierarchically-nested list of instructions and literals, the latter of which the interpreter pushes onto the relevant stack. We use the Clojush, the Clojure implementation of PushGP, for our experiments.[2]

We present the PushGP system parameters used in our

[1] The datasets for these problems can be found at `https://git.io/fjPeh`.

[2] `https://github.com/lspector/Clojush`

Table 2: Full training set size and program execution limit for each problem.

Problems	Training Set Size	Executions
Number IO	25	7,500,000
Sum Of Squares	50	15,000,000
Compare String Lengths, Digits, Double Letters, Even Squares, For Loop Index, Median, Mirror Image, Replace Space With Newline, Smallest, Small Or Large, String Lengths Backwards, Syllables	100	30,000,000
Last Index of Zero, Vectors Summed, X-Word Lines	150	45,000,000
Count Odds, Grade, Negative to Zero, Pig Latin, Scrabble Score, String Differences, Super Anagrams	200	60,000,000
Vector Average	250	75,000,000
Checksum	300	90,000,000

experiments in Table 1. Our only genetic operator, uniform mutation with additions and deletions (UMAD), adds random genes before each gene in a parent's genome at the *UMAD addition rate*, and then deletes random genes at a rate to remain size-neutral on average. We use UMAD to produce 100% of the children, instead of also using a crossover operator, since thus far it has produced the best results of any operator tested on these problems (Helmuth et al., 2018).

Each problem in the benchmark suite prescribes a number of training cases to use (Helmuth and Spector, 2015). In our default configuration, we run every individual on every training case, meaning the total number of program executions allowed in one GP run is the number of training cases multiplied by the population size and generations. Since our down-sampled lexicase selection experiments use fewer cases to evaluate each individual, we limit our GP runs by a program execution limit, as given in Table 2.

Benchmarking Down-sampled Lexicase Selection

Previous work compared down-sampled lexicase selection using subsampling levels of 0.05, 0.1, 0.25, and 0.5 on four or five program synthesis benchmark problems (Hernandez et al., 2019; Ferguson et al., 2019). Here, we test each of those levels on 12 benchmark problems to get a more detailed idea of their utility. Additionally, we use a different genetic programming system, PushGP.

Table 3: Number of successes out of 100 GP runs of down-sampled lexicase selection with four different subsampling levels, as well as 1.0, which is equivalent to standard lexicase selection. Mean is across problems. The mean rank calculates the average rank of each method among all methods across the problems, excluding Mirror Image and Smallest, easy problems where results differ only in random changes in solution generalization.

Problem	0.05	0.1	0.25	0.5	1.0
CSL	38	25	51	40	32
Double Letters	87	72	50	29	19
LIOZ	72	68	65	63	62
Mirror Image	100	99	99	100	100
Negative to Zero	84	86	82	78	80
RSWN	99	96	100	93	87
Scrabble Score	18	19	31	28	13
Smallest	100	99	98	100	100
SLB	99	96	95	94	94
Syllables	48	61	64	54	38
Vector Average	100	98	97	95	88
X-Word Lines	98	95	91	86	61
Mean	78.6	76.2	76.9	71.7	64.5
Mean Rank	2.0	2.4	2.2	3.7	4.8

Table 3 presents the success rates for down-sampled lexicase selection using four different subsampling levels across twelve representative benchmark problems, along with the mean number of successes. The last column of 1.0 performs no down-sampling, and is therefore standard lexicase selection. For each problem, we calculate the rank of each subsampling level, and average those to calculate the mean rank, where lower values are better.

The subsampling level of 0.05 performed the best on average, propelled by its significantly better results on the difficult Double Letters problem. However, every subsampling level performed well, and all considerably better than standard lexicase (i.e. subsampling level of 1.0). The level of 0.5 performed worst of the subsampling levels, likely because it only runs for at most twice as many generations as standard lexicase selection, where the other methods run for even more generations. It is surprising that the subsampling level of 0.05 performed best, as it only uses 5 training cases per generation for seven of the problems, limiting the information contained in the errors on which lexicase selection bases selection. We assume that reducing the subsampling level further would result in reduced performance, though further study will be needed to find the lower bound on effective subsampling levels. That said, subsampling levels 0.05, 0.1 and 0.25 performed nearly identically, showing that down-sampled lexicase selection is robust to a variety of subsampling levels.

Table 4: Comparing lexicase selection to down-sampled lexicase selection with a subsampling level of 0.25. Underlined values indicate significant improvement of down-sampled lexicase over lexicase. Lexicase was never significantly better than down-sampled lexicase.

Problem	Lexicase	Down-sampled
Checksum	1	18
CSL	32	51
Count Odds	8	11
Digits	19	28
Double Letters	19	50
Even Squares	0	2
For Loop Index	2	5
Grade	0	2
Last Index of Zero	62	65
Median	55	69
Mirror Image	100	99
Negative To Zero	80	82
Number IO	98	99
Pig Latin	0	0
RSWN	87	100
Scrabble Score	13	31
Small Or Large	7	22
Smallest	100	98
String Differences	0	1
SLB	94	95
Sum of Squares	21	25
Super Anagrams	4	4
Syllables	38	64
Vector Average	88	97
Vectors Summed	11	21
X-Word Lines	61	91

Table 4 compares standard lexicase selection (i.e. the column 1.0 in Table 3) to down-sampled lexicase selection with a subsampling level of 0.25 on a larger set of 26 benchmark problems.[3] Down-sampled lexicase selection produced significantly more successful runs than lexicase selection on 9 out of the 26 problems. It additionally found solutions to 3 of the problems that lexicase selection never solved, and had fewer successes on only two of the problems, neither of which were significantly different. This expanded benchmarking confirms previous findings that down-sampled lexicase selection creates great improvements in performance compared to lexicase selection.

One question raised by Ferguson et al. (2019) is whether down-sampled lexicase selection's method of randomly replacing the subsampled training cases each generation is beneficial, or if a static subsample of training cases would be just as good. To examine this question, we performed a set of runs that uses lexicase selection with a static, randomly subsampled set of 10 training cases that do not change during evolution; this uses an increased number of maximum generations like with down-sampled lexicase selection. Since each problem uses a different number of training cases (100 or 200 for most benchmark problems), this is not equal in number to any one subsample level, but is often equal to a subsample level of 0.1 or 0.05. We compare down-sampled lexicase selection with subsample level of 0.1 to lexicase selection using a static set of 10 cases in Table 5. Down-sampled lexicase performed significantly better on 11 of the 12 problems tested. This gives strong evidence for the importance of randomly changing the subsample each generation, which was the conclusion also found by Ferguson et al. (2019).

Table 5: Comparing down-sampled lexicase at a 0.1 subsampling level (DS 0.1) to using lexicase selection with a static set of 10 random training cases, which do not change during evolution. Underlined successes are significantly better.

Problem	DS 0.1	Static
Compare String Lengths	25	0
Double Letters	72	4
Last Index of Zero	68	7
Mirror Image	99	13
Negative To Zero	86	31
Replace Space with Newline	96	57
Scrabble Score	19	13
Smallest	99	40
String Lengths Backwards	96	35
Syllables	61	9
Vector Average	98	71
X-Word Lines	95	35

All of these results point to the considerable benefits of down-sampled lexicase selection compared to standard lexicase selection. Additional evidence comes from a recent benchmarking of parent selection techniques for program synthesis, which found down-sampled lexicase selection to perform best out of a field of 21 parent selection techniques (Helmuth and Abdelhady, 2020). We therefore turn to the question of what makes down-sampled lexicase selection better than other parent selection methods.

Hypothesis: Depth of Search

It seems clear that a primary (and possibly the only) benefit of down-sampled lexicase selection is that it allows GP to consider more individuals (i.e. points in the search space) within the same budget of program executions. Ferguson et al. (2019) argue in particular that "deeper evolutionary

[3] We only had the computational resources to test one subsampling rate on this larger set of problems. While subsampling rate of 0.25 did not produce the best results in Table 3, it performed almost as well as 0.05, which we tested too late to include comparisons on all 26 benchmark problems.

Table 6: Number of successes out of 100 GP runs of down-sampled lexicase selection using increases in population size instead of increases in maximum generations, at three different subsampling levels. Underlined results are significantly better (and ***italic/bold*** results are significantly worse) than the corresponding results at the same subsampling level with increased maximum generations (i.e. those in Table 3). The "Average Rank Pop" and "Average Rank Gen" give the average rank of down-sampled lexicase selection with increased population and increased generations respectively, so that ranks vary from 1 to 6.

Problem	0.05	0.1	0.25
Compare String Lengths	48	32	42
Double Letters	***53***	***42***	***35***
Last Index of Zero	76	72	77
Mirror Image	100	100	100
Negative To Zero	86	86	91
Replace Space With Newline	99	100	95
Scrabble Score	18	50	64
Smallest	99	100	100
String Lengths Backwards	100	100	98
Syllables	***24***	55	76
Vector Average	100	93	99
X-Word Lines	94	96	84
Mean	74.7	77.2	80.1
Average Rank Pop	3.2	3.4	3.2
Average Rank Gen	3.3	4.0	4.0

searches", i.e. having a larger maximum number of generations, leading to longer lineages of evolution, is responsible for improvements in performance—we call this the *generations hypothesis*. We present a competing hypothesis, that down-sampled lexicase selection's better performance is simply due to evaluating a larger number of individuals, but not related to the depth of the search, which we call the *search space hypothesis*.

To test these hypotheses, we devised an experiment in which we use down-sampled lexicase selection, but instead of increasing the maximum number of generations per run, we increase the population size while maintaining a fixed number of program executions. For example, with a subsample level of 0.25, we will increase the population size by 4 times, from 1000 to 4000. This experiment will have GP evaluate the same number of points in the search space as using an increased maximum generations, but will not allow for longer evolutionary lineages than standard lexicase selection, as each run is limited to 300 generations. We test the same subsample levels as in Table 3, except that we leave off 0.5, since it performed worse than the other levels.

We present results using down-sampled lexicase selection with increased population sizes in Table 6. We compare results at the same subsample level between increased generations and increased population sizes. Out of the 36 comparisons, 2 sets of runs were significantly better with increased population, and 4 were significantly worse. The mean success rates across problems are comparable to those with increased generations. We additionally present the average ranking of 6 down-sampled lexicase selection methods (3 that increase population size and 3 that increase maximum generations) across 10 of the problems, excluding the easy problems Mirror Image and Smallest, for which differences only reflect minor differences in generalization rate. The average ranks are all quite close to the overall average rank of 3.5, with increased population having a slightly better average rank across the three subsampling levels (3.3 vs. 3.8).

We take these results as evidence against the generations hypothesis, in that increasing population size while fixing the maximum number of generations produces very similar performance to increasing generations. These results give credence to the search space hypothesis, that we only need to have down-sampled lexicase selection increase the number of individuals we evaluate during evolution, whether that increase comes from increases in population size or more generations. We also recommend utilizing the bonus program evaluations on increasing the maximum generation or population size, as both lead to similarly good performance; the choice between the two may come down to other factors within the GP system.

Hypothesis: Changing Environment

One interesting aspect of down-sampled lexicase selection is that it changes the set of subsampled training cases every generation. If we think of the set of training cases as the challenges encountered by each individual, this corresponds to an environment that changes over time, requiring the evolving population to adapt to new circumstances (i.e. cases). In contrast, with a fixed set of training cases, lexicase selection provides a static environment, though one in which individuals encounter challenges in a different order for each selection. Changing environments often have interesting effects on evolutionary dynamics (Levins, 1968), leading us to wonder if they have a beneficial impact when using down-sampled lexicase selection. Here we explore the hypothesis that down-sampled lexicase selection changes the evolutionary dynamics in a positive way beyond increasing the number of individuals that are evaluated.

To test this hypothesis, we designed an experiment that uses a static set of training cases like with lexicase selection, but has each selection only use a subsample of those cases, like with down-sampled lexicase selection. In particular, we use *truncated lexicase selection*, which evaluates every individual on every training case each generation, but cuts off each lexicase selection after using a fixed number of cases (Spector et al., 2017). In our experiment, we compare down-sampled lexicase selection at the 0.1 subsample

Table 7: Number of successes out of 100 GP runs of down-sampled lexicase and truncated lexicase selections, both at the 0.1 level, and both over 3000 generations. Underlined results are significantly better than the comparison.

Problem	Down-sampled	Truncated
Double Letters	72	69
Scrabble Score	19	<u>90</u>
Vector Average	98	100

level with truncated lexicase selection also using only 10% of the cases for each selection. The main difference between the two is that across all selections, truncated lexicase selection uses every training case each generation, where down-sampled lexicase selection uses the same subsample for every selection. [4]

In our experiment, we run both down-sampled lexicase and truncated lexicase selections for 3000 generations. As truncated lexicase selection requires every individual to be evaluated on every training case each generation, this is not a fair comparison in terms of total program executions, but it is not meant to be. If the "changing environments" hypothesis holds, then down-sampled lexicase selection should produce better results than truncated lexicase selection, since its environment changes each generation where truncated lexicase selection's does not. We chose three problems for which down-sampled lexicase selection performed much better than standard lexicase selection over 300 generations, ensuring there is a possibility of performing worse than down-sampled lexicase selection.

Table 7 presents the number of successful runs of down-sampled lexicase selection and truncated lexicase selection with a maximum of 3000 generations. Over these three problems, truncated lexicase selection performed significantly better than down-sampled lexicase selection on the Scrabble Score problem, and very similarly on the other two problems. So, not only was down-sampled lexicase selection not better, it was a bit worse. This gives some evidence against the hypothesis that the "changing environment" of down-sampled lexicase selection contributes to its success, though we admit that there may be other beneficial evolutionary dynamics at play not captured by this experiment. We also want to emphasize that this experiment does not suggest that truncated lexicase selection should be preferred over down-sampled lexicase selection, or even standard lexicase selection for that matter; truncated lexicase selection used 10 times as many program executions in these runs as down-sampled lexicase selection, meaning they are not being compared on a level playing field.

[4] Ferguson et al. (2019) also conduct an experiment comparing truncated lexicase selection to down-sampled lexicase selection, but to address a different question; we see no contradiction between their results and the ones we present here.

Table 8: Comparing generalization rates of lexicase selection and down-sampled lexicase selection with a subsampling level of 0.25. These generalization rates are for the success rates in Table 4. None of the differences in generalization were significant.

Problem	Lexicase	Down-sampled
Checksum	1.00	1.00
CSL	0.49	0.61
Count Odds	1.00	1.00
Digits	0.66	0.60
Double Letters	0.95	0.98
Even Squares	-	1.00
For Loop Index	0.67	1.00
Grade	-	1.00
Last Index of Zero	0.67	0.66
Median	0.57	0.69
Mirror Image	1.00	0.99
Negative To Zero	0.84	0.83
Number IO	0.98	0.99
Pig Latin	-	-
RSWN	1.00	1.00
Scrabble Score	0.93	1.00
Small Or Large	0.32	0.42
Smallest	1.00	0.98
String Differences	-	1.00
SLB	1.00	1.00
Sum of Squares	1.00	1.00
Super Anagrams	0.80	1.00
Syllables	0.97	0.96
Vector Average	1.00	1.00
Vectors Summed	0.92	0.95
X-Word Lines	1.00	0.98

Hypothesis: Better Generalization

As discussed in the Related Work section above, down-sampling has been used (without lexicase selection) in both GP and machine learning more broadly as a method to combat overfitting and increase the generalization of solutions. There is plenty of room for improvement in generalization on some of our benchmark problems, with 6 problems having generalization rates below 0.7 when using lexicase selection. Does down-sampling improve generalization when using lexicase selection?

All of our successful rate counts above only include generalizing solutions that pass a large set of random, unseen test cases. We look at the proportion of solution programs that pass the training set that also pass the test set to calculate the *generalization rate* for each set of runs. For the extended set of 26 benchmark problems presented in Table 4, we present the generalization rate for each problem in Table 8. Even though there are some minor differences in

generalization between lexicase and down-sampled lexicase selections, none of them are significantly different using a chi-squared test. Problems that appear to have a large gap between the two, such as For Loop Index and Super Anagrams, do not have enough solutions to show significance.

At this point we have no evidence to suggest that down-sampling improves lexicase selection's generalization rate. In fact, down-sampled lexicase selection displays poor generalization on many of the same problems that lexicase selection does. Thus we cannot attribute the improved performance of down-sampled lexicase selection to avoiding overfitting and improving generalization.

Conclusions

In this paper we have shed more light on the performance and mechanisms of down-sampled lexicase selection. We conducted more extensive benchmarking of down-sampled lexicase selection than has been conducted before, finding that it performs well across a large range of benchmark problems. We find that it is important to change training cases every generation within a larger set of training cases, as a subsampling method that uses a static set of cases throughout evolution performed much worse than down-sampled lexicase selection.

We then considered the hypothesis that down-sampled lexicase selection performs well because of its ability to search for more generations, leading to deeper evolutionary lineages. Our experiment that makes use of down-sampled lexicase selection's extra program executions to increase the population size rather than extending evolutionary time provides evidence against this hypothesis, since approximately the same benefit is obtained with larger populations as with more generations. We also examine the hypothesis that down-sampled lexicase selection's changing of training cases every generation acts like an environment changing over evolutionary time, contributing to its success. Our experiment using truncated lexicase selection provides evidence against this hypothesis, though other environmental effects could be at play. A third experiment showed that down-sampled lexicase selection does not produce better generalization rates of solution programs compared to lexicase selection, despite this being a benefit of down-sampling in other machine learning systems. These experiments lead us to believe that the primary cause of down-sampled lexicase selection's success is that it allows evolution to consider more programs throughout evolution.

This work and that of Ferguson et al. (2019) and Hernandez et al. (2019) use problems from the same general program synthesis benchmark suite. We would certainly like to see similar experiments performed in other problem domains, where training set subsampling has been used previously, but not to our knowledge in conjunction with lexicase selection.

This research points to the importance of maximizing the number of points in the search space—individuals—that genetic programming considers throughout evolution. It would be difficult to push down-sampled lexicase selection further in this direction, as it would start to use too few training cases per generation to create a reasonable gradient for lexicase selection to follow, an effect that we see developing at the 0.05 subsampling level on some problems. That said, other methods that increase the number of individuals considered by genetic programming without sacrificing information about individuals' performances (or even ones that do sacrifice some information, as in down-sampled lexicase selection) could provide additional benefits.

Acknowledgements

We thank Emily Dolson, Amr Abdelhady, and the Hampshire College Computational Intelligence Lab for discussions that improved this work. This material is based upon work supported by the National Science Foundation under Grant No. 1617087. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Curry, R. and Heywood, M. I. (2004). Towards efficient training on large datasets for genetic programming. In *17th Conference of the Canadian Society for Computational Studies of Intelligence*, volume 3060 of *LNAI*, pages 161–174, London, Ontario, Canada. Springer-Verlag.

Ferguson, A. J., Hernandez, J. G., Junghans, D., Dolson, E., and Ofria, C. (2019). Characterizing the effects of random subsampling on lexicase selection. In *Genetic Programming Theory and Practice XVII*, East Lansing, MI, USA.

Forstenlechner, S., Fagan, D., Nicolau, M., and O'Neill, M. (2017). A grammar design pattern for arbitrary program synthesis problems in genetic programming. In *EuroGP 2017: Proceedings of the 20th European Conference on Genetic Programming*, volume 10196 of *LNCS*, pages 262–277, Amsterdam. Springer Verlag.

Gathercole, C. and Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 312–321, Jerusalem. Springer-Verlag.

Goncalves, I. and Silva, S. (2013). Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In *Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013*, volume 7831 of *LNCS*, pages 73–84, Vienna, Austria. Springer Verlag.

Helmuth, T. and Abdelhady, A. (2020). Benchmarking parent selection for program synthesis by genetic programming. In *GECCO '20: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation Companion*. ACM.

Helmuth, T., McPhee, N. F., Pantridge, E., and Spector, L. (2017). Improving generalization of evolved programs through automatic simplification. In *Proceedings of the Genetic and Evo-*

lutionary Computation Conference, GECCO '17, pages 937–944, Berlin, Germany. ACM.

Helmuth, T., McPhee, N. F., and Spector, L. (2016). The impact of hyperselection on lexicase selection. In Friedrich, T., editor, *GECCO '16: Proceedings of the 2016 Annual Conference on Genetic and Evolutionary Computation*, pages 717–724, Denver, USA. ACM.

Helmuth, T., McPhee, N. F., and Spector, L. (2018). Program synthesis using uniform mutation by addition and deletion. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 1127–1134, Kyoto, Japan. ACM.

Helmuth, T. and Spector, L. (2015). General program synthesis benchmark suite. In *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1039–1046, Madrid, Spain. ACM.

Helmuth, T., Spector, L., and Matheson, J. (2015). Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643.

Hernandez, J. G., Lalejini, A., Dolson, E., and Ofria, C. (2019). Random subsampling improves performance in lexicase selection. In *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 2028–2031, Prague, Czech Republic. ACM.

Hmida, H., Ben Hamida, S., Borgi, A., and Rukoz, M. (2016). Sampling methods in genetic programming learners from large datasets: A comparative study. In *INNS Conference on Big Data*, volume 529 of *Advances in Intelligent Systems and Computing*, pages 50–60.

Kleinberg, R., Li, Y., and Yuan, Y. (2018). An alternative view: When does SGD escape local minima?

La Cava, W., Helmuth, T., Spector, L., and Moore, J. H. (2018). A probabilistic and multi-objective analysis of lexicase selection and epsilon-lexicase selection. *Evolutionary Computation*.

Levins, R. (1968). *Evolution in Changing Environments: Some Theoretical Explorations*. Monographs in Population Biology. Princeton University Press.

Liskowski, P., Krawiec, K., Helmuth, T., and Spector, L. (2015). Comparison of semantic-aware selection methods in genetic programming. In *GECCO 2015 Semantic Methods in Genetic Programming (SMGP'15) Workshop*, pages 1301–1307, Madrid, Spain. ACM.

Martinez, Y., Naredo, E., Trujillo, L., Legrand, P., and Lopez, U. (2017). A comparison of fitness-case sampling methods for genetic programming. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(6):1203–1224.

Moore, J. M. and Stanton, A. (2017). Lexicase selection outperforms previous strategies for incremental evolution of virtual creature controllers. *Proceedings of the European Conference on Artificial Life*, pages 290–297.

Moore, J. M. and Stanton, A. (2018). Tiebreaks and diversity: Isolating effects in lexicase selection. *The 2018 Conference on Artificial Life*, pages 590–597.

Oksanen, K. and Hu, T. (2017). Lexicase selection promotes effective search and behavioural diversity of solutions in linear genetic programming. In Lozano, J. A., editor, *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 169–176, Donostia, San Sebastian, Spain. IEEE.

Schmidt, M. D. and Lipson, H. (2008). Coevolution of fitness predictors. *IEEE Transactions on Evolutionary Computation*, 12(6):736–749.

Spector, L., Klein, J., and Keijzer, M. (2005). The Push3 execution stack and the evolution of control. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1689–1696, Washington DC, USA. ACM Press.

Spector, L., La Cava, W., Shanabrook, S., Helmuth, T., and Pantridge, E. (2017). Relaxations of lexicase parent selection. In *Genetic Programming Theory and Practice XV*, Genetic and Evolutionary Computation, pages 105–120, University of Michigan in Ann Arbor, USA. Springer.

Spector, L., McPhee, N. F., Helmuth, T., Casale, M. M., and Oks, J. (2016). Evolution evolves with autoconstruction. In *GECCO '16 Companion: Proceedings of the Companion Publication of the 2016 Annual Conference on Genetic and Evolutionary Computation*, pages 1349–1356, Denver, Colorado, USA. ACM.

Spector, L. and Robinson, A. (2002). Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40.

Zhang, B.-T. and Joung, J.-G. (1999). Genetic programming with incremental data inheritance. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1217–1224, Orlando, Florida, USA. Morgan Kaufmann.