Problem-Solving Benefits of Down-Sampled Lexicase Selection

Thomas Helmuth*
Hamilton College
thelmuth@hamilton.edu

Lee Spector
Amherst College
Hampshire College
University of Massachusetts Amherst
lspector@amherst.edu

Abstract In genetic programming, an evolutionary method for producing computer programs that solve specified computational problems, 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 training 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. This can be seen as modeling the fact that individual organisms encounter only subsets of the possible environments and that environments 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 the evolutionary process to examine more individuals within the same computational budget, even though each individual is examined less completely.

Keywords
Genetic programming, parent selection, lexicase selection, down-sampled lexicase selection, program synthesis

1 Introduction

Genetic programming is an evolutionary method for producing computer programs that solve specified computational problems (Koza, 1992). When used as a supervised learning technique, genetic programming defines a problem's specifications by a set of *training cases*. It then judges the ability of evolved programs to solve the problem by running each program on each training case, and measuring the distance between the program's output and the desired output. Genetic programming

* Corresponding author.

uses these *error values* during *parent selection* 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 is analogous to the interaction between a biological organism and the challenges presented by its environment. Organisms that are better equipped to handle these challenges have better reproductive success, and in genetic programming the programs that produce outputs closer to the desired outputs should produce more children.

Many parent selection methods have been developed for genetic programming, and they vary in the ways that they model the interactions that biological organisms have with their environments. In most, the performance of a program on all of the training cases is aggregated into a single value, referred to as a *fitness measure* or *total error*, and the probability that a program will produce offspring is partially or entirely determined by this aggregate value. Even multi-objective optimization methods, which select on the basis of multiple objectives, generally nonetheless aggregate performance across training cases into one objective (Deb et al., 2002; Kotanchek et al., 2006, 2008; Schmidt & Lipson, 2010a). Similarly, the recent development of quality diversity algorithms (Cully, 2019; Cully & Demiris, 2018) such as MAP-Elites (Mouret & Clune, 2015; Vassiliades et al., 2018) use aggregate fitness as part of the basis for selection.

The aggregation of performance is akin to exposing all organisms to all challenges that they could possibly face, and allowing those that perform best on average to produce more children. In biology, by contrast, each organism may face different challenges, and it will produce offspring if it survives the challenges that it happens to face before it has the opportunity to reproduce.

The lexicase parent selection method differs from most other parent selection methods in that it avoids the aggregation of performance on different training cases into a single value (Helmuth et al., 2015; Spector, 2012). Instead, it filters individuals by performance on training cases that are presented in different random orders for each parent selection event, with the result that different parents will be selected on the basis of good performance on different sequences of training cases. Additionally, children in the next generation will face different randomly shuffled cases than their parents did. For these reasons, lexicase selection can be thought of as more faithfully modeling interactions between biological organisms and their environments.

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 for 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. Crucially, both methods reduce the amount of computational effort required to evaluate each individual, since they run each program only on a subsample of the training cases. These computational savings can be recouped by evaluating more individuals throughout evolution. 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 article 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 selection 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.

One area where down-sampling (without lexicase selection) has proven useful is in avoiding overfitting and improving generalization, both in genetic programming (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 article extends a preliminary report that was presented at the 2020 Artificial Life conference (Helmuth & Spector, 2020). Aside from general improvements to the clarity and completeness of the presentation in the conference paper, this article covers experiments involving more subsampling levels and more benchmark problems, with both of these extensions producing significant new results. One key area we explore is in using extremely small subsampled sets of training cases, resulting in surprisingly good performance with some notable drawbacks.

Our presentation below continues as follows: We first discuss lexicase selection and subsampling of training cases in more detail. Once we have covered these fundamental algorithms, we describe our experimental methods and present our benchmark results. We then address each of the above-described hypotheses in turn, and conclude with our interpretation of the results and suggestions for future work.

2 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 Algorithm 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 evolutionary computation systems and problem domains (Aenugu & Spector, 2019; Forstenlechner et al., 2017; Helmuth et al., 2015; Helmuth & Spector, 2015; La Cava et al., 2019; Liskowski et al., 2015; Oksanen & Hu, 2017; Orzechowski et al., 2018; Metevier et al., 2019; Moore & Stanton, 2017, 2018, 2019, 2020).

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 resampled each generation. This method reduces the per-individual computational effort, which can 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

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 training 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 individual from candidates
  end if
  Remove the first case from cases
end loop
```


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 & 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 (Curry & Heywood, 2004; Gathercole & Ross, 1994; Hmida et al., 2016; Martinez et al., 2017; Zhang & Joung, 1999). Others have proposed subsampling as a technique to reduce overfitting and improve generalization (Goncalves & Silva, 2013; Martinez et al., 2017; Schmidt & Lipson, 2006, 2008, 2010b). Additionally, subsampling data is common in machine learning for similar reasons (often referred to as mini-batches), 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 to 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 & 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.

3 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 (Ferguson et al., 2019; Hernandez et al., 2019). In particular, we use problems from the General Program Synthesis Benchmark Suite (Helmuth & 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

Table 1. 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

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. We additionally compare down-sampled lexicase selection to standard lexicase selection on the 25 problems of PSB2, the second iteration of general program synthesis benchmark problems (Helmuth & Kelly, 2021).[1]

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.[2] For each problem we use a small set of *training cases* to evaluate each individual: between 25 and 300 cases per run (see Table 1) and 200 cases for every problem in PSB2. We use 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.

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 (using it as a validation 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. As we will detail in Section 4.2, with an extremely low subsampling level that leaves the subsampled training set with 1 or 2 cases, it is easier for GP to generate individuals that perfectly pass those cases without passing the full training set; with enough of these individuals, the process of verifying that they pass the full training set may dominate the running time of evolution. 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

1 More information can be found at the benchmark suite's website: https://cs.hamilton.edu/~thelmuth/PSB2/PSB2.html.

2 Datasets for these problems can be found at https://git.io/fjPeh.

Table 2. PushGP system parameters.

Parameter	Value
population size	1,000
max generations for runs using full training set	300
genetic operator	UMAD
UMAD addition rate	0.09

Note. UMAD = uniform mutation with additions and deletions.

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 & Robinson, 2002). Push was designed with GP 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 & 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.[3]

We present the PushGP system parameters used in our experiments in Table 2. 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 & 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 1, to ensure that each method receives equal training time.

4 Benchmarking Down-Sampled Lexicase Selection

In the work introducing down-sampled lexicase selection, experiments benchmarked down-sampled lexicase selection with subsampling levels of 0.05, 0.1, 0.25, and 0.5 on five program synthesis problems (Ferguson et al., 2019; Hernandez et al., 2019). We expand on those benchmarks by testing three additional subsampling levels, 0.01, 0.02, and 0.175, with the first two explicitly trying to gauge how low the subsampling rate can get before having deleterious effects. Our experiments increase the number of benchmark problems to 12, and additionally test the subsampling level of 0.25 on 39 other program synthesis benchmark problems to broaden our assessment. As described above, our experiments use PushGP, showing that the benefits of down-sampling generalize beyond the linear GP system used in the initial experiments (Ferguson et al., 2019; Hernandez et al., 2019).

3 https://github.com/lspector/Clojush

4.1 Subsampling Levels

Table 3 presents the success rates for down-sampled lexicase selection using seven 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 therefore represents standard lexicase selection. For these runs, we proportionally increase the maximum number of generations that evolution can run to keep a constant number of program executions; for example, while standard lexicase selection runs for at most 300 generations, the runs with a subsampling level of 0.02 run for at most $\frac{1}{0.02} = 50$ times as many, at 15,000 generations. For each problem, we calculate the rank of each subsampling level, and average those to calculate the mean rank, where lower values are better. Six sets of runs (five at the subsampling level of 0.01 and one at level 0.02) were not able to complete in a reasonable amount of time, as discussed in section 4.2.

The subsampling level of 0.02 performed the best on average, propelled by its significantly better results on the difficult Double Letters and Last Index of Zero problems. However, every subsampling level performed well, and all considerably better than standard lexicase (i.e., the subsampling level of

Table 3. Number of successes out of 100 GP runs of down-sampled lexicase selection with proportional increases in maximum generations per run across seven different subsampling levels, as well as 1.0, which is equivalent to standard lexicase selection.

	Subsampling level							
Problem	0.01	0.02	0.05	0.1	0.175	0.25	0.5	1.0
CSL	0/4	48/97	38	25	60	51	40	32
Double Letters	5/42	85	87	72	55	50	29	19
LIOZ	94	90	72	68	61	65	63	62
Mirror Image	100	100	100	99	99	99	100	100
Negative To Zero	96	84	84	86	86	82	78	80
RSWN	100	100	99	96	97	100	93	87
Scrabble Score	1/98	7	18	19	24	31	28	13
Smallest	100	100	100	99	100	98	100	100
SLB	100	100	99	96	96	95	94	94
Syllables	11/60	47	48	61	68	64	54	38
Vector Average	100	100	100	98	99	97	95	88
X-Word Lines	25/60	96	98	95	94	91	86	61
Mean	61.0	79.8	78.6	76.2	78.3	76.9	71.7	64.5
Mean Rank	4.8	3.2	3.4	4.2	3.7	4.0	5.8	7.1

Note. *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. For six sets of runs at subsampling levels 0.01 and 0.02, we were not able to finish all 100 runs, as described in section 4.2; the number of finished runs is given after the /.

1.0). The level of 0.5 performed worst of the subsampling levels, likely because it only runs for twice as many generations as standard lexicase selection, whereas the other subsampling levels run longer.

It is surprising that the subsampling level of 0.02 performed best, as it only uses 2 training cases per generation for seven of the problems, significantly limiting the information contained in the errors on which lexicase selection bases selection. In fact, with only 2 training cases, lexicase selection can only select individuals with 2 different error vectors corresponding to the 2 possible orderings of the cases! Even so, this extreme constraint on selection introduced by down-sampling seems to be largely outweighed by increasing the maximum number of generations manyfold.

Even though a subsampling level of 0.02 performed best, subsampling levels 0.02, 0.05, 0.1, 0.175, and 0.25 performed nearly identically, showing that down-sampled lexicase selection is robust to a wide variety of subsampling levels across an order of magnitude.

4.2 Lower Bounds of Subsampling Level

Since down-sampled lexicase selection performs well at quite low levels of subsampling, are there any drawbacks? Is there a lower bound to the benefits of subsampling?

First, we will examine the results on our lowest subsampling level, 0.01. We see that it performed excellently on 7 out of 12 problems, including 4 where it operated on a single training case (Mirror Image, Replace Space with Newline, String Lengths Backwards, and Smallest) and 3 others with only 2 or 3 cases. These results include producing the absolute best results on 2 problems, Last Index of Zero and Negative To Zero. However, it gave polarized performance, producing the worst results on the remaining 5 problems. For 2 of these problems (Scrabble Score and Syllables), there is a clear trend toward worse performance with the lowest subsampling levels, but for the other 3, down-sampled lexicase selection performs well even at the 0.02 subsampling level. We interpret these findings to suggest that, at least for some problems, 1 to 3 cases is not sufficient information to drive evolution toward solutions, likely resulting in either catastrophic lack of diversity, thrashing of the population between trying to solve different cases, or other detriments.

Beyond the problem-solving performance considerations, using extremely low subsampling levels results in other unwanted behaviors of the GP system. Typically in GP, we consider the program executions to be the time limiting factor of running GP, and therefore tune our experiments to use the same number of program executions regardless of down-sampling. However, as we proportionally increase the number of maximum generations to make up for fewer program executions per generation, the remaining components of the GP system (such as genetic operators and data logging) take up a larger proportion of the running time in practice. Additionally, if we run evolution for many generations (for example, 100 times as many with the subsampling level of 0.01), we will require that many times more hard drive space to log data from runs. Similar issues exist with reserving sufficient RAM when increasing the population size instead of the maximum generations.

In Table 3, six of our sets of runs at low subsampling levels were not able to finish all 100 runs in a reasonable amount of time and were cutoff before finishing. Some of the extreme length of these runs is likely attributable to the effects discussed in the previous paragraph. However, a subtler and potentially more harmful effect is at play as well. As described in section 3, when GP finds a program that passes all of the subsampled training cases, we must test it on the remaining training cases before calling it a potential solution and halting evolution; if it does not pass all training cases, evolution continues. With extremely small subsampled sets, it becomes easier for evolution to find (many) individuals that pass all of the subsampled data, requiring us to fully evaluate those individuals, which many times do not pass the full training set. This problem is compounded for problems that have Boolean outputs (such as Compare String Lengths), since even if the entire population chooses between `True` and `False` randomly, if there is only one case in the subsampled set, half of the population will answer that case correctly and need to be evaluated on every training case every generation, negating the benefits of quick evaluation per generation. This certainly impacted the low number of finished runs of the Compare String Lengths problem at the 0.01 subsampling level, and likely contributed to unfinished runs on other problems at that level.

With these drawbacks in mind, we see subsampling levels between 0.05 and 0.25 producing good compromises between problem-solving performance and real running times. In the following section, we benchmark down-sampled lexicase selection using a subsampling level of 0.25, though we expect the results would look similar at a variety of subsampling levels.

4.3 Expanding Benchmarking of Down-Sampled Lexicase Selection to More Problems

After extensively testing a variety of subsampling levels on 12 benchmark problems, we want to exhibit its performance on a larger set of benchmark problems. We only had the computational resources to test one subsampling level on this larger set of problems, and chose 0.25. While the subsampling level of 0.25 did not produce the best results in Table 3, it performed almost as well as any level, and was less computationally demanding than much lower subsampling levels for the reasons discussed in section 4.2.

Table 4 compares standard lexicase selection (i.e., the 1.0 column in Table 3) to down-sampled lexicase selection with a subsampling level of 0.25 on 26 benchmark problems from Helmuth and Spector (2015), including the 12 from Table 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 2 of the problems, neither of which were significantly different.

Table 5 continues the comparison from Table 4 on 25 new problems from PSB2 (Helmuth & Kelly, 2021). These problems were designed to be a step more difficult than those from Helmuth and Spector (2015), and show lower success rates for both standard lexicase selection and down-sampled lexicase selection. However, down-sampled lexicase selection continues to clearly outperform standard lexicase selection, solving 4 problems that standard lexicase never solved, and performing significantly better on 8 problems. In fact, down-sampled lexicase never produced fewer solutions than standard lexicase on any of the 25 problems. This expanded benchmarking confirms previous findings that down-sampled lexicase selection creates great improvements in performance compared to lexicase selection.

4.4 Comparison With Static Subsample of Cases

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 as 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 subsampling level, but is often equal to a subsampling level of 0.1 or 0.05. We compare down-sampled lexicase selection with the subsampling level of 0.1 to lexicase selection using a static set of 10 cases in Table 6. 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).

5 Hypotheses for Down-Sampled Lexicase Selection's Performance

All of our 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 & Abdelhady, 2020). We therefore turn to the question of what makes down-sampled lexicase selection better than other parent selection methods. In this section, we present three distinct hypotheses examining the origins of the

Table 4. Number of successful runs comparing lexicase selection to down-sampled lexicase selection with a subsampling level of 0.25 on 26 benchmark problems.

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

Table 4. (*continued*)

Problem	Down-sampled	Lexicase
X-Word Lines	91	61
Problems Solved	25	22

Note. Underlined values indicate significant improvement of down-sampled lexicase over lexicase using a chi-square test. Lexicase was never significantly better than down-sampled lexicase. *Problems Solved* counts the number of problems each method solved at least once.

Table 5. Number of successful runs comparing lexicase selection to down-sampled lexicase selection with a subsampling level of 0.25 on the 25 new benchmark problems of PSB2.

Problem	Down-sampled	Lexicase
Basement	2	1
Bouncing Balls	3	0
Bowling	0	0
Camel Case	4	1
Coin Sums	39	2
Cut Vector	0	0
Dice Game	1	0
Find Pair	20	4
Fizz Buzz	74	25
Fuel Cost	67	50
GCD	20	8
Indices of Substring	4	0
Leaders	0	0
Luhn	0	0
Mastermind	0	0
Middle Character	79	57
Paired Digits	17	8
Shopping List	0	0

Table 5. (*continued*)

Problem	Down-sampled	Lexicase
Snow Day	7	4
Solve Boolean	5	5
Spin Words	0	0
Square Digits	2	0
Substitution Cipher	86	61
Twitter	52	31
Vector Distance	0	0
Problems Solved	17	13

Note. Underlined values indicate significant improvement of down-sampled lexicase over lexicase using a chi-square test. Lexicase was never significantly better than down-sampled lexicase. *Problems Solved* counts the number of problems each method solved at least once.

Table 6. Number of successful runs 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.

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

Note. Underlined successes are significantly better using a chi-square test.

benefits bestowed by down-sampled lexicase selection, and conduct experiments to provide evidence for or against these hypotheses.

5.1 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, p. 3) argue in particular that "deeper evolutionary 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, the *search space 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.

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 subsampling level of 0.25, we will increase the population size by 4 times, from 1,000 to 4,000. This experiment has GP evaluate the same number of points in the search space as using increased maximum generations, but will not allow for longer evolutionary lineages than standard lexicase selection, as each run is limited to 300 generations. We tested three representative subsampling levels for increased population size, and compared them to the equivalent subsampling levels with increased maximum generations, using the same data as in Table 3.

We present results using down-sampled lexicase selection with increased population sizes in Table 7. We compare results at the same subsampling 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 versus 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. While these conclusions reflect the general results, there are some interesting problem-specific trends to note in Table 7. Increasing generations produced significantly better results on the Last Index of Zero problem at all three subsampling levels, and the inverse was true on Scrabble Score for two of the three subsampling levels. Keeping this in mind, we recommend utilizing the bonus program evaluations allowed by down-sampling on increasing the maximum generations 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 or to a particular problem.

5.2 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

Table 7. Number of successes out of 100 GP runs of down-sampled lexicase selection at three different subsampling levels.

	Population			Generations		
Problem	0.05	0.1	0.25	0.05	0.1	0.25
Compare String Lengths	48	32	42	38	25	51
Double Letters	53	42	35	87	72	50
Last Index of Zero	76	72	77	72	68	65
Mirror Image	100	100	100	100	99	99
Negative To Zero	86	86	91	84	86	82
Replace Space with Newline	99	100	95	99	96	100
Scrabble Score	18	50	64	18	19	31
Smallest	99	100	100	100	99	98
String Lengths Backwards	100	100	98	99	96	95
Syllables	24	55	76	48	61	64
Vector Average	100	93	99	100	98	97
X-Word Lines	94	96	84	98	95	91
Mean	74.7	77.2	80.1	78.6	76.2	76.9
Mean Rank	3.2	3.4	3.2	3.3	4.0	4.0

Note. This compares increasing population size to increasing maximum generations, with the latter being identical to the data in Table 3. Underlined results are significantly better than the corresponding results at the same subsampling level using a chi-square test. *Mean Rank* gives the average rank of each of the six treatments, so that ranks vary from 1 to 6.

in a different order for each selection. Changing environments often have interesting effects on evolutionary dynamics (Levins, 1968), and empirical studies of evolving populations of *Saccharomyces cerevisiae* yeast (Boyer et al., 2021), logic functions (Kashtan et al., 2007), and digital organisms (Canino-Koning et al., 2019; Nahum et al., 2017) have demonstrated that the speed and effectiveness of adaptive evolution can be affected, and in some cases enhanced, by environmental variation. This led us to ask whether environmental variation might be responsible for the benefits of 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, as with lexicase selection, but has each selection use only a subsample of those cases, as 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 compared down-sampled lexicase selection at the 0.1 subsampling 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

Table 8. 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 3,000 generations.

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

Note. Underlined results are significantly better using a chi-square test.

selection uses every training case each generation, where down-sampled lexicase selection uses the same subsample for every selection.[4]

In our experiment, we ran both down-sampled lexicase and truncated lexicase selections for 3,000 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 environment*s hypothesis holds, then down-sampled lexicase selection should produce better results than truncated lexicase selection, since its environment changes each generation whereas 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 8 presents the number of successful runs of down-sampled lexicase selection and truncated lexicase selection with a maximum of 3,000 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.

5.3 Hypothesis: Better Generalization

As discussed in section 2 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 run 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 9. 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-square 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.

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 9. Comparing generalization rates of lexicase selection and down-sampled lexicase selection with a subsampling level of 0.25.

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

Table 9. (*continued*)

Problem	Down-sampled	Lexicase
Vectors Summed	0.95	0.92
X-Word Lines	0.98	1.00

Note. These generalization rates are for the success rates in Table 4. None of the differences in generalization were significant.

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.

6 Conclusions

In this article we shed additional light on the performance and mechanisms of down-sampled lexicase selection. We conducted more extensive benchmarking of down-sampled lexicase selection than has been previously done, and found that it performs well across a large range of benchmark problems and subsampling levels. We described some of the drawbacks of using very low subsampling levels, despite their ability to produce competitive problem-solving performance. We found 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 examined 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 (i.e., individuals) that GP considers throughout evolution. In this article we pushed the abilities of down-sampled lexicase selection to increase the number of individuals considered to the extreme, and found that at the 0.01 and 0.02 subsampling levels, problem-solving performance remains surprisingly good, while actual processor performance diminishes. We would be interested to see what effects such low subsampling levels have on population dynamics such as diversity, considering that they allow lexicase to select only a tiny fraction of the individuals in the population.

Other methods that increase the number of individuals considered by GP without sacrificing information about individuals' performances (or even ones that do sacrifice some information, as in down-sampled lexicase selection) could provide additional benefits. Exploring this avenue illuminated

by down-sampled lexicase selection may yield other techniques that, possibly in combination with down-sampled lexicase selection, could continue to drive the field forward.

Acknowledgments

We thank Emily Dolson, Amr Abdelhady, and the Hampshire College Computational Intelligence Lab for discussions that improved this work.

Funding Information

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

Aenugu, S., & Spector, L. (2019). Lexicase selection in learning classifier systems. In M. López-Ibáñez (Ed.), *GECCO '19: Proceedings of the genetic and evolutionary computation conference* (Prague, Czech Republic, 13 July 2019, pp. 356–364). ACM. https://doi.org/10.1145/3321707.3321828

Boyer, S., Hérissant, L., & Sherlock, G. (2021). Adaptation is influenced by the complexity of environmental change during evolution in a dynamic environment. *PLOS Genetics*, *17*(1), e1009314. https://doi.org/10.1371/journal.pgen.1009314, PubMed: 33493203

Canino-Koning, R., Wiser, M. J., & Ofria, C. (2019). Fluctuating environments select for short-term phenotypic variation leading to long-term exploration. *PLOS Computational Biology*, *15*(4), e1006445. https://doi.org/10.1371/journal.pcbi.1006445, PubMed: 31002665

Cully, A. (2019). Autonomous skill discovery with quality-diversity and unsupervised descriptors. In M. López-Ibáñez (Ed.), *GECCO '19: Proceedings of the genetic and evolutionary computation conference companion* (Prague, Czech Republic, July 2019, pp. 81–89). ACM. https://doi.org/10.1145/3321707.3321804

Cully, A., & Demiris, Y. (2018). Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, *22*(2), 245–259. https://doi.org/10.1109/TEVC.2017.2704781

Curry, R., & Heywood, M. I. (2004). Towards efficient training on large datasets for genetic programming. In A. Y. Tawfik & S. D. Goodwin (Eds.), *Advances in artificial intelligence: Canadian AI 2004* (pp. 161–174). Springer. https://doi.org/10.1007/978-3-540-24840-8_12

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197. https://doi.org/10.1109/4235.996017

Ferguson, A. J., Hernandez, J. G., Junghans, D., Lalejini, A., Dolson, E., & Ofria, C. (2019). Characterizing the effects of random subsampling and dilution on lexicase selection. In W. Banzhaf, E. Goodman, L. Sheneman, L. Trujillo, & B. Worzel (Eds.), *Genetic programming theory and practice XVII* (pp. 1–23). Springer. https://doi.org/10.1007/978-3-030-39958-0_1

Forstenlechner, S., Fagan, D., Nicolau, M., & O'Neill, M. (2017). A grammar design pattern for arbitrary program synthesis problems in genetic programming. In J. McDermott, M. Castelli, L. Sekanina, E. Haasdijk, & P. García-Sánchez (Eds.), *Genetic programming: 20th European conference: EuroGP 2017* (pp. 262–277). Springer. https://doi.org/10.1007/978-3-319-55696-3_17

Gathercole, C., & Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In Y. Davidor, H. P. Schwefel, & R. Männer (Eds.), *Parallel problem solving from nature—PPSN III. PPSN 1994* (pp. 312–321). Springer. https://doi.org/10.1007/3-540-58484-6_275

Goncalves, I., & Silva, S. (2013). Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In K. Krawiec, A. Moraglio, T. Hu, A. Ş. Etaner-Uyar, & B. Hu (Eds.), *Genetic Programming: EuroGP 2013* (pp. 73–84). Springer. https://doi.org/10.1007/978-3-642-37207-0_7

Helmuth, T., & 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* (Cancún, Mexico, July 2020, pp. 237–238). ACM. https://doi.org/10.1145/3377929.3389987

Helmuth, T., & Kelly, P. (2021). PSB2: The second program synthesis benchmark suite. In F. Chicano (Ed.), *GECCO '21: Proceedings of the genetic and evolutionary computation conference* (Lille, France, June 2021, pp. 785–794). ACM. https://doi.org/10.1145/3449639.3459285

Helmuth, T., McPhee, N. F., Pantridge, E., & Spector, L. (2017). Improving generalization of evolved programs through automatic simplification. In *GECCO '17: Proceedings of the genetic and evolutionary computation conference* (Berlin, Germany, July 2017, pp. 937–944). ACM. https://doi.org/10.1145/3071178.3071330

Helmuth, T., McPhee, N. F., & Spector, L. (2016). The impact of hyperselection on lexicase selection. In T. Friedrich (Ed.), *GECCO '16: Proceedings of the 2016 annual conference on genetic and evolutionary computation* (Denver, CO, July 2016, pp. 717–724). ACM. https://doi.org/10.1145/2908812.2908851

Helmuth, T., McPhee, N. F., & Spector, L. (2018). Program synthesis using uniform mutation by addition and deletion. In H. Aguirre (Ed.), *GECCO '18: Proceedings of the genetic and evolutionary computation conference* (Kyoto, Japan, July 2018, pp. 1127–1134). ACM. https://doi.org/10.1145/3205455.3205603

Helmuth, T., & Spector, L. (2015). General program synthesis benchmark suite. In S. Silva (Ed.), *GECCO '15: Proceedings of the 2015 annual conference on genetic and evolutionary computation* (Madrid, Spain, July 2015, pp. 1039–1046). ACM. https://doi.org/10.1145/2739480.2754769

Helmuth, T., & Spector, L. (2020). Explaining and exploiting the advantages of down-sampled lexicase selection. In J. Bongard, J. Lovato, L. Hebert-Dufrésne, R. Dasari, & L. Soros (Eds.), *Proceedings of the ALIFE 2020: The 2020 conference on artificial life* (Online, pp. 341–349). MIT Press. https://doi.org/10.1162/isal_a_00334

Helmuth, T., Spector, L., & Matheson, J. (2015). Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, *19*(5), 630–643. https://doi.org/10.1109/TEVC.2014.2362729

Hernandez, J. G., Lalejini, A., Dolson, E., & Ofria, C. (2019). Random subsampling improves performance in lexicase selection. In M. López-Ibáñez (Ed.), *GECCO '19: Proceedings of the genetic and evolutionary computation conference companion* (Prague, Czech Republic, July 2019, pp. 2028–2031). ACM. https://doi.org/10.1145/3319619.3326900

Hmida, H., Ben Hamida, S., Borgi, A., & Rukoz, M. (2016). 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: Proceedings of the 2nd INNS conference on big data* (Thessaloniki, Greece, October 2016, pp. 50–60). Springer. https://doi.org/10.1007/978-3-319-47898-2_6

Kashtan, N., Noor, E., & Alon, U. (2007). Varying environments can speed up evolution. *Proceedings of the National Academy of Sciences*, *104*(34), 13711–13716. https://www.pnas.org/content/104/34/13711. https://doi.org/10.1073/pnas.0611630104, PubMed: 17698964

Kleinberg, R., Li, Y., & Yuan, Y. (2018). An alternative view: When does SGD escape local minima? In *Proceedings of the 35th international conference on machine learning, PMLR 80* (pp. 2698–2707). PMLR.

Kotanchek, M., Smits, G., & Vladislavleva, E. (2006). Pursuing the pareto paradigm tournaments, algorithm variations & ordinal optimization. In R. L. Riolo, T. Soule, & B. Worzel (Eds.), *Genetic programming theory and practice IV* (pp. 167–185). Springer. https://doi.org/10.1007/978-0-387-49650-4_11

Kotanchek, M., Smits, G., & Vladislavleva, E. (2008). Exploiting trustable models via pareto GP for targeted data collection. In R. L. Riolo, T. Soule, & B. Worzel (Eds.), *Genetic programming theory and practice VI* (pp. 145–163). Springer.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. MIT Press. https://mitpress.mit.edu/books/genetic-programming

La Cava, W., Helmuth, T., Spector, L., & Moore, J. H. (2019). A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. *Evolutionary Computation*, *27*(3), 377–402. https://doi.org/10.1162/evco_a_00224, PubMed: 29746157

Levins, R. (1968). *Evolution in changing environments: Some theoretical explorations* (Monographs in Population Biology). Princeton University Press. https://doi.org/10.1515/9780691209418

Liskowski, P., Krawiec, K., Helmuth, T., & Spector, L. (2015). Comparison of semantic-aware selection methods in genetic programming. In S. Silva (Ed.), *GECCO '15: Semantic methods in genetic programming (SMGP'15) workshop* (Madrid, Spain, July 2015, pp. 1301–1307). ACM. https://doi.org/10.1145/2739482.2768505

Martinez, Y., Naredo, E., Trujillo, L., Legrand, P., & Lopez, U. (2017). A comparison of fitness-case sampling methods for genetic programming. *Journal of Experimental & Theoretical Artificial Intelligence*, *29*(6), 1203–1224. https://doi.org/10.1080/0952813X.2017.1328461

Metevier, B., Saini, A. K., & Spector, L. (2019). Lexicase selection beyond genetic programming. In W. Banzhaf, L. Spector, & L. Sheneman L. (Eds.), *Genetic programming theory and practice XVI* (pp. 123–136). Springer. https://doi.org/10.1007/978-3-030-04735-1_7

Moore, J. M., & Stanton, A. (2017). Lexicase selection outperforms previous strategies for incremental evolution of virtual creature controllers. In *Proceedings of the ECAL 2017, the fourteenth European conference on artificial life* (Lyon, France, September 2017, pp. 290–297). https://doi.org/10.1162/isal_a_050

Moore, J. M., & Stanton, A. (2018). Tiebreaks and diversity: Isolating effects in lexicase selection. In *Proceedings of the ALIFE 2018: The 2018 conference on artificial life* (Tokyo, Japan, July 2018, pp. 590–597). MIT Press. https://doi.org/10.1162/isal_a_00109

Moore, J. M., & Stanton, A. (2019). The limits of lexicase selection in an evolutionary robotics task. In *Proceedings of the ALIFE 2019: The 2019 conference on artificial life* (Online, July 2019, pp. 551–558). MIT Press. https://doi.org/10.1162/isal_a_00220

Moore, J. M., & Stanton, A. (2020). When specialists transition to generalists: Evolutionary pressure in lexicase selection. In *Proceedings of the ALIFE 2020: The 2020 conference on artificial life* (Online, July 2020, pp. 719–726). MIT Press. https://doi.org/10.1162/isal_a_00254

Mouret, J.-B., & Clune, J. (2015). Illuminating search spaces by mapping elites. ArXiv:1504.04909. https://arxiv.org/abs/1504.04909

Nahum, J. R., West, J., Althouse, B. M., Zaman, L., Ofria, C., & Kerr, B. (2017). Improved adaptation in exogenously and endogenously changing environments. In *Proceedings of the ECAL 2017, the fourteenth European conference on artificial life* (Lyon, France, September 2017, pp. 306–313). MIT Press. https://doi.org/10.1162/isal_a_052

Oksanen, K., & Hu, T. (2017). Lexicase selection promotes effective search and behavioural diversity of solutions in linear genetic programming. In J. A. Lozano (Ed.), *2017 IEEE congress on evolutionary computation (CEC)* (Donostia-San Sebastian, Spain, June 2017, pp. 169–176). IEEE. https://doi.org/10.1109/CEC.2017.7969310

Orzechowski, P., La Cava, W., & Moore, J. H. (2018). Where are we now? A large benchmark study of recent symbolic regression methods. In H. Aguirre (Ed.), *GECCO '18: Proceedings of the genetic and evolutionary computation conference* (Kyoto, Japan, July 2018, pp. 1183–1190). ACM. https://doi.org/10.1145/3205455.3205539

Schmidt, M. D., & Lipson, H. (2006). Co-evolving fitness predictors for accelerating and reducing evaluations. In R. L. Riolo, T. Soule, & B. Worzel (Eds.), *Genetic programming theory and practice IV* (pp. 113–130). Springer. https://doi.org/10.1007/978-0-387-49650-4_8

Schmidt, M. D., & Lipson, H. (2008). Coevolution of fitness predictors. *IEEE Transactions on Evolutionary Computation*, *12*(6), 736–749. https://doi.org/10.1109/TEVC.2008.919006

Schmidt, M. D., & Lipson, H. (2010a). Age-fitness pareto optimization. In R. Riolo, T. McConaghy, & E. Vladislavleva (Eds.), *Genetic programming theory and practice VIII* (pp. 129–146). Springer. https://doi.org/10.1007/978-1-4419-7747-2_8

Schmidt, M. D., & Lipson, H. (2010b). Predicting solution rank to improve performance. In *GECCO '10: Proceedings of the 12th annual conference on genetic and evolutionary computation* (Portland, Oregon, July 2010, pp. 949–956). ACM. https://doi.org/10.1145/1830483.1830652

Spector, L. (2012). Assessment of problem modality by differential performance of lexicase selection in genetic programming: A preliminary report. In T. Soule (Ed.), *GECCO '12: Proceedings of the 14th annual conference companion on genetic and evolutionary computation, Session: Understanding problems (GECCO-UP)* (Philadelphia, PA, July 2012, pp. 401–408). ACM. https://doi.org/10.1145/2330784.2330846

Spector, L., Klein, J., & 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* (Washington, DC, June 2005, Vol. 2, pp. 1689–1696). ACM. https://doi.org/10.1145/1068009.1068292

Spector, L., La Cava, W., Shanabrook, S., Helmuth, T., & Pantridge, E. (2017). Relaxations of lexicase parent selection. In W. Banzhaf, R. Olson, W. Tozierm, & R. Riolo (Eds.), *Genetic programming theory and practice XV* (pp. 105–120). Springer. https://doi.org/10.1007/978-3-319-90512-9_7

Spector, L., McPhee, N. F., Helmuth, T., Casale, M. M., & Oks, J. (2016). Evolution evolves with autoconstruction. In T. Friedrich (Ed.), *GECCO '16: Proceedings of the 2016 annual conference on genetic and evolutionary computation* (Denver, CO, July 2016, pp. 1349–1356). ACM. https://doi.org/10.1145/2908961.2931727

Spector, L., & Robinson, A. (2002). Genetic programming and autoconstructive evolution with the Push programming language. *Genetic Programming and Evolvable Machines*, *3*(1), 7–40. https://hampshire.edu/lspector/pubs/push-gpem-final.pdf. https://doi.org/10.1023/A:1014538503543

Vassiliades, V., Chatzilygeroudis, K., & Mouret, J. B. (2018). Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, *22*(4), 623–630. https://doi.org/10.1109/TEVC.2017.2735550

Zhang, B.-T., & Joung, J.-G. (1999). Genetic programming with incremental data inheritance. In W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, & R. Smith (Eds.), *GECCO-99: Proceedings of the genetic and evolutionary computation conference* (Orlando, FL, July 1999, Vol. 2, pp. 1217–1224). Morgan Kaufmann. https://gpbib.cs.ucl.ac.uk/gecco1999/GP-460.pdf