Relaxations of Lexicase Parent Selection

Lee Spector, William La Cava, Saul Shanabrook, Thomas Helmuth, and Edward Pantridge

Abstract In a genetic programming system, the parent selection algorithm determines which programs in the evolving population will be used as the material out of which new programs will be constructed. The lexicase parent selection algorithm chooses a parent by considering all test cases, individually, one at a time, in a random order, to reduce the pool of possible parent programs. Lexicase selection is ordinarily strict, in that a program can only be selected if it has the best error in the entire population on the first test case considered, and the best error relative to all other programs that remain in the pool each time it is reduced. This strictness may exclude high-quality candidates from consideration for parenthood, and hence from exploration by the evolutionary process. In this chapter we describe and present results of four variants of lexicase selection that relax these strict constraints: epsilon lexicase selection, random threshold lexicase selection, MADCAP epsilon lexicase selection, and truncated lexicase selection. We present the results of experiments with genetic programming systems using these and other parent selection algorithms on symbolic regression and software synthesis problems. We also briefly discuss the relations between lexicase selection and work on many-objective optimization, and the implications of these considerations for future work on parent selection in genetic programming.

Lee Spector
Hampshire College, Amherst, MA, e-mail: lspector@hampshire.edu

William La Cava
University of Pennsylvania, Philadelphia, PA, e-mail: lacava@upenn.edu

Saul Shanabrook
University of Massachusetts, Amherst, MA, e-mail: s.shanabrook@gmail.com

Thomas Helmuth
Hamilton College, Clinton, NY, e-mail: thelmuth@hamilton.edu

Edward Pantridge
MassMutual, Amherst, MA, e-mail: EPantridge@MassMutual.com

1 Introduction

Almost all parent selection algorithms used in genetic programming systems involve not only comparisons among potential parent programs, but also random choices.

For example, in fitness-proportionate selection, we simulate the spinning of a roulette wheel, with the size of the pocket for each parent being inversely proportional to its total error over the set of test cases. It is possible for any program in the population to be selected as a parent, and the choice among potential parents, while random, is biased so that higher quality programs have higher probabilities of being selected.

In tournament selection, we first choose a tournament set randomly from the population, with each program in the population having equal probability of being chosen. We then select the best program in the tournament set to serve as the parent, where the "best" program is the one with the lowest total error. Here random choices are made first, to determine the tournament set, followed by a choice that is driven by the quality of the programs that have been chosen to participate in the tournament.

One could, in principle, avoid making any random choices in parent selection, but this is rarely done. For example, one could use a form of pure elitism, in which each program in the "best" $n\%$ of the population is selected as a parent some prespecified number of times.

In lexicase selection, randomization plays a particularly central role, but rather than being applied directly to choices of programs, it is applied to sequences of individual test cases by which programs are compared. In lexicase selection, parents are selected through a filtering process that is iterated over a sequence of test cases that is randomly shuffled for each parent selection event. Once randomly re-ordered, the test cases are considered one at a time, with only the best programs for each case retained at the corresponding filtering step.

Tournament selection can be thought of as subjecting a subset of the potential parents to a challenge of this form for each selection event: "Are you better, overall, than these other randomly-chosen programs?" By contrast, lexicase selection can be thought of as subjecting each potential parent to a sequence of challenges—"Are you the best on *this* test case? And of those of you who are, are you best on *this* one? Etc."—among which the order is randomized.

Lexicase selection has been shown to be advantageous in several contexts. It often allows problems to be solved more quickly and reliably than they can be without it [7, 17], and in some cases allows for the solution of problems that cannot otherwise be solved at all [6].

The power of lexicase selection appears to stem from the way in which it leverages multiple, randomized challenges to guide search. The randomization of test case order allows the parent selection process to be sensitive to more information about the strengths and weaknesses of programs in the population than it can be under the approach used in tournament selection. In fact, recent experiments with weighted shuffling of test cases produced similar or worse results, suggesting that the uniform shuffling of test cases allows lexicase selection to better sample useful programs in the population [22]. This randomization of challenges allows lexicase

selection to be sensitive not only to the performance of programs on all test cases considered in aggregate, but also to their performance on all subsets of the test cases; in this way, lexicase selection often selects individuals that specialize in some test cases while performing poorly on others. Considering all subsets of test cases explicitly would require exponential resources, but randomization allows them to be considered implicitly, through random sampling.

When a parent selection algorithm is sensitive to more information about the strengths and weaknesses of programs, then that information may be used to provide better guidance to evolutionary search in different ecological circumstances. Semantic- or behavior-aware genetic programming methods (such as lexicase selection) take into account information about a program's execution or its individual outputs/errors on test cases, going beyond methods that simply use a single fitness value [18, 9, 8, 15].

Might variations of lexicase selection perform even better on problems of specific kinds? In this chapter we describe and present data on four "relaxed" forms of lexicase selection, each of which allows some programs to be selected that would not be selected by ordinary lexicase selection; these are epsilon lexicase selection, random threshold lexicase selection, MADCAP epsilon lexicase selection, and truncated lexicase selection. Among the motivations for considering these forms of relaxation is the hypothesis that ordinary lexicase selection can sometimes be too strict, insofar as it eliminates any opportunity to serve as a parent for some programs that are quite good in many respects.

In the following sections, we first describe the most basic form of lexicase selection, on which the other selection methods described in this chapter are based. We then describe each of the four relaxed versions of lexicase selection in turn. Following the descriptions of the algorithms, we present and discuss the results of experiments involving all of the described algorithms, along with a few others from the literature to facilitate broader comparisons. These experiments involve eight symbolic regression problems and five software synthesis problems. We conclude with a brief discussion of the relation between work on lexicase selection and work on many-objective optimization, and we discuss the implications of our results for future research.

2 Lexicase selection

Lexicase selection is designed for problems in which candidate solutions are assessed with multiple test cases. In most other parent selection methods, a candidate solution's performance over multiple test cases is aggregated into a single measure, for example, an average error value, and this single aggregate measure is used as the basis of selection. In lexicase selection, no aggregation is performed; the measures for each individual test case are retained, and they may all be used, individually, in the parent selection process.

Although lexicase selection can be used for other kinds of performance measures as well, for the sake of simplicity we will refer to measures of performance on individual test cases as "errors," and we will assume that we are seeking a solution that minimizes all errors.

With the most basic form of lexicase selection ("global pool, uniform random sequence, elitist lexicase parent selection" [20], which will also refer to below as "ordinary lexicase selection"), when the genetic programming system requires a parent to use for the production of offspring, we first shuffle a sequence of the test cases that are being used to assess programs in the population. We then form a pool that initially contains all of the programs in population. We will winnow this pool down to a single selected program by considering each test case in turn. When each test case is considered, we first note the lowest error that any program in the pool has for that test case. We then eject all programs that have a higher error on that test case from the pool. If these ejections ever reduce the pool to a single program, then we return that program as the selected parent. If instead, we exhaust the test cases and still have more than one program in the pool, then we return one of them randomly.

Why might one expect this selection method to be useful? One reason is that it allows programs to be selected if they perform particularly well on individual test cases, or on collections of test cases, even if they perform poorly on many others. This allows specialists to produce offspring that may build on their specialties, perhaps in conjunction with other specialties that they may have inherited from other ancestors. The full reasons for lexicase selection's utility, however, are more complex, and still under investigation [3, 15, 4, 5].

A variety of time optimizations of lexicase selection are possible. For example, we can include in the initial pool just a single random representative of any group that shares the same errors for all test cases. Doing so will decrease the number of programs in the pool that will have to be filtered, and it will also allow fewer test cases to be considered for some parent selection events.

3 Epsilon lexicase selection

In prior work, it was noted that lexicase selection would sometimes perform poorly on symbolic regression problems involving floating-point errors [7]. It was thought that this was due to the fact that in these contexts it would often be the case that most or all programs in the population would have unique error values when any particular test case is considered. In such situations, the strictness of lexicase selection would reduce the candidate pool to a single program as soon as the first test case is considered, and parenthood decisions would often be made on the basis of single test cases. These considerations led to the development of epsilon lexicase selection, which has indeed proven to be useful for problems involving floating-point errors [12, 11].

In epsilon lexicase selection we relax the elitism of the filtering steps. Rather than retaining only the programs with exactly the lowest error on the current test case, we retain all programs that are "close enough"—that is, those with errors within some small *epsilon* of the lowest error of any program in the pool on the current test case.

While the reasons for epsilon lexicase selection's good performance are still under investigation, an intuitive case for its success can be based on the considerations sketched above. In a population in which no two programs have the same error for any test case, which is not terribly hard to imagine for problems with floating-point errors, ordinary lexicase selection would select every parent based on a single test case. Specialists would still be selected, but not programs that perform well on multiple test cases. By allowing programs with errors that are "close enough" to the minimum error on a test case to pass through the filter, the algorithm will once again be able to select programs based on performance on larger subsets of the test cases. Support for this theory has been demonstrated by observing that epsilon lexicase selection uses more cases per selection event than ordinary lexicase selection does on regression problems [12, 11].

How should the epsilon in epsilon lexicase selection be determined? Several approaches to this question have been explored, with the most consistently good performance having been obtained so far with a method dubbed "MAD" epsilon lexicase selection, for "Median Absolute Deviation from the median." Here epsilon is computed one per generation, for each test case, on the basis of all the errors for the test case across the population. Specifically, epsilon for a particular test case is computed as the median of the differences between errors on the case and the median error for the case. When we use the name "epsilon lexicase selection" below, without further qualification, we are referring to this method.

4 Random Threshold Lexicase Selection

Epsilon lexicase selection sets a threshold for each challenge that a program must meet in order to survive a filtering step: if the program has an error for the case that differs from the best error by the threshold or less, then it survives. The threshold is set on the basis of the distribution of errors for the test cases in the population; for example, it is the median absolute deviation from the median error when the MAD version of epsilon lexicase selection is used.

The idea behind random threshold lexicase selection is to randomize the setting of the threshold as well. One motivation for doing this is the observation that the threshold in epsilon lexicase selection can be quite sensitive to changes in the distribution of error values across the population. If the distribution is not sufficiently well behaved, for example because of unusual features of the problem that we are trying to solve, or because high error penalties are imposed on programs that violate specified constraints, or because the genetic operators being used often produce large changes in error between parent and child, then one might expect the thresholds used by epsilon selection to be unhelpful.

For this reason, random threshold lexicase selection was developed to choose thresholds that are derived from the errors present in the population, but less sensitive to their specific distributions. Specifically, at each step of filtering, we choose an error randomly from those present in the current pool for the current case. We then retain only those programs that have the chosen error or better for the current case. If the randomly selected error happens to be the best error in the pool, then the filtering at this step will be equivalent to that used by ordinary lexicase selection. If it happens to be the worst error in the pool, then no filtering at all will take place for the current test case in the current selection event.

One can think of random threshold lexicase selection as randomly sampling combinations of relative tightness of selection on different test cases, all within lexicase selection's random ordering of test cases. So there is a sense in which all orderings of test cases and also all combinations of strictness vs. laxness for each test case may be considered. As with ordinary lexicase selection, however, we do not consider all of these combinations of challenges explicitly. Rather, we sample both the orderings and the strictnesses of the challenges that programs must confront.

At one extreme, when an elite error is picked at each step, this will act like ordinary lexicase selection. However, this will be rare. Consequentially, random threshold lexicase selection is a significantly relaxed form of lexicase selection, insofar as it will generally make it easier for programs to meet the challenges to survive filtering. For problems in which all errors are binary (pass/fail), it will act like lexicase selection on a random subset of the cases.

One would expect that this technique would often end up producing effects similar to those of ordinary lexicase selection, but with some test cases more-or-less skipped during some selection events, while the cases with "tight" bounds on errors will be the ones that do the major culling. How much of an effect this will have can be expected to depend on how many intermediate values there are between the elite values and the worst values; if there are many, then we might expect its effects to be quite different from those of ordinary lexicase selection.

We can think of both epsilon lexicase selection and random threshold lexicase selection as loosening lexicase selection's elitist filtering condition, and thereby weakening the challenge presented by each test case. Such weakening will generally lessen the selection pressure exerted by individual test cases while increasing the selection pressure exerted by groups of test cases that are adjacent in random shuffles.

We would generally expect random threshold lexicase selection to weaken test case challenges more than MAD epsilon lexicase selection does, since we would expect the bound provided by epsilon to be relatively tight, so that randomly chosen errors would not usually fall within it. It is possible that this will mean that random threshold lexicase selection will not provide enough selection pressure for good performance on individual test cases. Whether or not this will actually be the case is an empirical question, best answered by experiments.

5 MADCAP Epsilon Lexicase Selection

Random threshold lexicase selection relaxes test case challenges in a randomized way, but it may also be useful to consider methods that do something similar while nonetheless obeying the limit of relaxation used in epsilon lexicase selection. That is, it might be useful in some contexts to consider methods that vary in stringency between MAD epsilon lexicase selection and ordinary lexicase selection (which is strictly elitist), again using randomization to sample different strengths applied to different test cases. MADCAP epsilon lexicase selection is such a method.

At each filtering step of MADCAP epsilon lexicase selection, we sometimes retain just the best individuals on the case, and sometimes retain any individuals within epsilon of best, choosing randomly between these options for each test case. The application of epsilon lexicase's "cap" (threshold) is probabilistic.[1] Specifically, we provide a parameter for the probability of applying the MAD cap versus just retaining individuals with the best error. In the experiments described below, this parameter is set to 0.5. At each filtering step, we use this probability to determine whether to retain only those programs with exactly the best error in the pool on the current case, or whether to retain all programs with errors within the MAD epsilon of the best error. Thus the selectivity of MADCAP epsilon lexicase selection will be between that of ordinary lexicase selection and MAD epsilon lexicase selection.

The motivation for this formulation is that for some problems, it is required that solutions have errors that are actually zero, or at least equal to the lowest possible error, rather than just being low. Especially for these problems, but possibly for others as well, we would like selection to sometimes (probabilistically) distinguish between programs that have the minimum (possibly zero) error on a test case and programs that merely have low errors.

Intuitively, one might expect MADCAP epsilon lexicase selection to allow the genetic programming search process to hone in on minimal-error solutions. Whether this happens in practice will probably depend on several factors including the distribution of errors in the population, which will depend in turn on factors such as the genetic operators and rates that are being used. Epsilon lexicase selection is always sensitive to the distribution of errors across the population, while MADCAP epsilon lexicase selection will always provide some selection in favor of elites, regardless of the error distribution.

As with the other methods considered here, MADCAP epsilon lexicase selection uses sampling to consider, in the limit but not explicitly, all combinations of favoring vs. not favoring elites for each case and each combination of cases.

Again, whether or not this will actually be useful in practice is an empirical question, best answered by experiments.

[1] So "MADCAP" = Median Absolute Deviation from the median, Cap Applied Probabilistically.

6 Truncated Lexicase Selection

Truncated lexicase selection is a form of lexicase selection in which we limit the number of cases that are considered in each parent selection event. The number (or percentage) of the total cases that will be considered is a parameter of the method. For example, suppose that we use truncated lexicase selection on a problem with 100 test cases and that we specify that 25% of cases will be used. Then for each parent selection event, we will proceed initially as we do in ordinary lexicase selection, but if we have filtered the pool using 25 test cases and it still contains multiple programs, then we will immediately choose a random remaining member of the pool and return it as the selected parent.

Epsilon lexicase selection, random threshold lexicase selection, and MADCAP epsilon lexicase selection are all relaxed forms of lexicase selection in which the constraints on selection are reduced with respect to the allowed error values for individual cases. In ordinary lexicase selection, a program can only be selected to serve as a parent if it is globally elite on at least one case and elite with respect to the survivors in the selection pool as each subsequent case is considered. In epsilon lexicase selection, random threshold lexicase selection, and MADCAP epsilon lexicase selection, this requirement of eliteness is relaxed, to a greater or lesser (and sometimes random) extent.

By contrast, in truncated lexicase selection we still require eliteness *on the test cases that are considered*, but we place no constraints at all on the error values of the cases that are not considered. Whether or not this form of relaxation has beneficial impacts on the ability of the genetic programming system to solve problems, it also has the potential to improve system runtimes by reducing the amount of computation that must be dedicated to filtering the lexicase selection candidate pools.

7 Experimental results

We include here the results from two sets of experiments on the relaxed variants of lexicase selection presented above.

First, we present the results of comparisons of several selection methods, including ordinary lexicase selection, epsilon lexicase selection, random threshold lexicase selection, and MADCAP lexicase selection, on a collection of eight symbolic regression problems. For completeness, we also include comparisons to purely random selection, tournament selection with a tournament size of 2, lasso selection [21], age-fitness Pareto optimization [19], and deterministic crowding [16].

Table 1 describes the problems used for these experiments, each of which comes from the UCI repository [14]. Table 2 describes the genetic programming system parameters that were used, and also provides abbreviations for the parent selection methods that were studied, which are used in the graph of results in Figure 1. The

Table 1 Regression problems used for method comparisons.

Problem	Dimension	Samples
Airfoil	5	1503
Concrete	8	1030
ENC	8	768
ENH	8	768
Housing	14	506
Tower	25	3135
UBall5D	5	6024
Yacht	6	309

Table 2 Genetic programming system settings for symbolic regression problems.

Setting	Value
GP tool	ellyn
Population size	1000
Crossover / mutation	60/40%
Program length limits	[3, 50]
ERC range	[-1,1]
Generation limit	1000
Trials	50
Terminal Set	$\{\mathbf{x}$, ERC, $+, -, *, /$, sin, cos, exp, log$\}$
Elitism	keep best
Fitness (non-lexicase methods)	MSE
Method	**Abbreviation**
Lasso [21]	lasso
Random selection	rand
Tournament selection (size 2)	tourn
Lexicase selection	lex
Age-fitness Pareto optimization [19]	afp
Deterministic crowding [16]	dc
epsilon-lexicase selection	ep-lex
Random threshold lexicase selection	ep-lex-rand
MADCAP epsilon-lexicase selection	ep-lex-madcap

experiments were run using the ellyn,[2] a linear GP system described in [10] (in this experiment, no epigenetic markers were used).

As Figure 1 makes clear, epsilon lexicase selection achieves the best results, and achieves the most consistently good results, across this set of problems. Ordinary lexicase selection sometimes performs reasonably well, occasionally beating competitors, but as has been noted elsewhere and motivated the development of epsilon lexicase selection in the first place, ordinary lexicase selection often performs relatively poorly in the context of floating-point errors [12].

Random threshold lexicase selection performs much better than ordinary lexicase selection but worse than epsilon lexicase selection. Its average ranking over these regression problems is about the same as deterministic crowding. MADCAP lexicase selection has broadly similar performance, sometimes a bit better and sometimes a bit worse. It is possible, however, that each of these relaxed forms of epsilon lexicase selection will prove to be more advantageous for problems with particular character-

[2] https://epistasislab.github.io/ellyn/

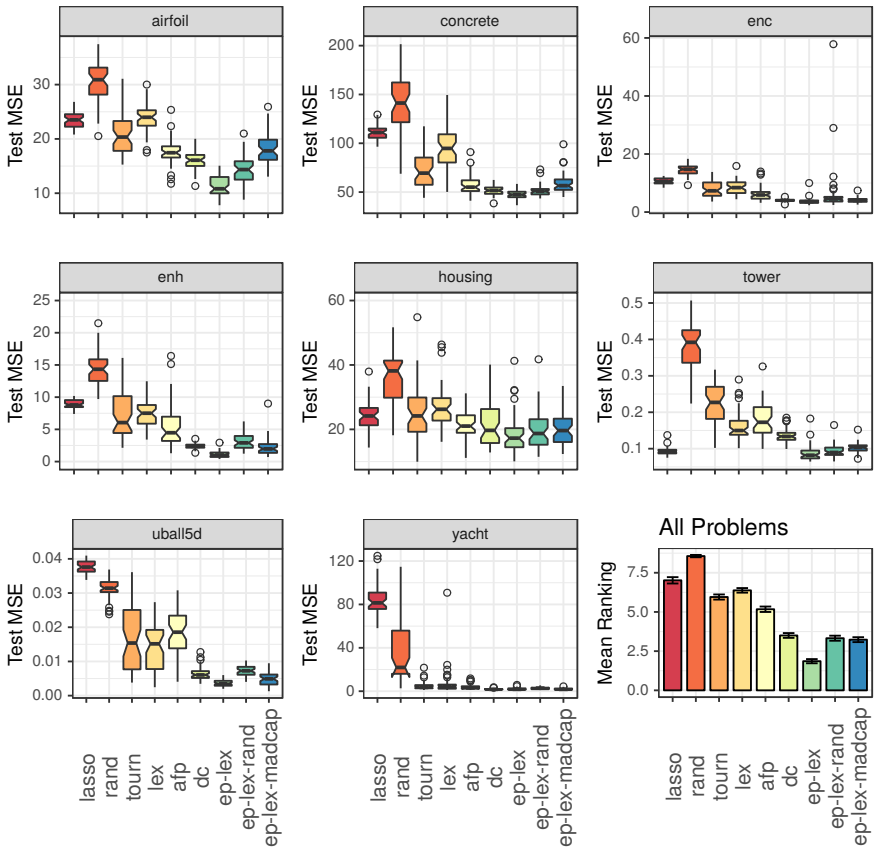

Fig. 1 Comparison of multiple parent selection methods on multiple symbolic regression problems. The boxplots span the upper and lower quartiles of test set mean squared error (MSE) for each problem, with a central line indicating the median. On the lower right, the mean ranking according to median MSE of each method across all problems is shown, with the error bar indicating the standard error.

istics that might be discovered by broadening the experiments to cover more types of problems.

In a second set of experiments, we compared ordinary lexicase selection to truncated lexicase selection on software synthesis problems from our general program synthesis benchmark suite [6]. Specifically, we conducted runs on the "Median," "Negative to Zero," "Number IO," "Replace Space with Newline,", "Smallest", and "Vector Average" problems. These problems require a range of data types and programming constructs to solve. For these problems, we tested truncating lexicase selection after considering just a single test case, and after considering 25% of the test cases. We note that when only a single test cases are considered we are creating, intentionally, exactly the situation which was thought to be problematic when using

ordinary lexicase selection on problems with floating-point errors, which motivated the development of epsilon lexicase selection.

Table 3 Genetic programming system settings for software synthesis problems. All problems are explained in detail in [6]. The full set of settings, including the set of instructions allowed in programs, can be found in the Clojush GitHub repository.

Setting	Smallest	Median	Negative To Zero	Number IO	RSWN	Vector Average
GP Tool	Clojush	Clojush	Clojush	Clojush	Clojush	Clojush
Population Size	1000	1000	1000	1000	1000	1000
Generation Limit	200	200	300	200	300	300
Alternation/Mutation/Both	20/30/50%	20/30/50%	20/30/50%	30/20/50%	20/30/50%	20/30/50%
Max Initial Genome Length	100	100	250	100	400	200
Test Cases	100	100	200	1000	100	250

The parameters for the runs comparing truncated lexicase selection with ordinary lexicase selection, using the Clojush implementation of the PushGP genetic programming system (https://github.com/lspector/Clojush), are shown in Table 3. We conducted 20 runs in each setting. Figure 2 shows the success rates that were observed. We note that the results appear to indicate that the relaxation provided by truncation damages problem-solving performance, with greater relaxation generally producing greater damage. We have too little data to draw any firm conclusions about the extent of this effect, or about the ways in which this effect may vary across problems, but we present it as a baseline for future study of truncated lexicase selection.

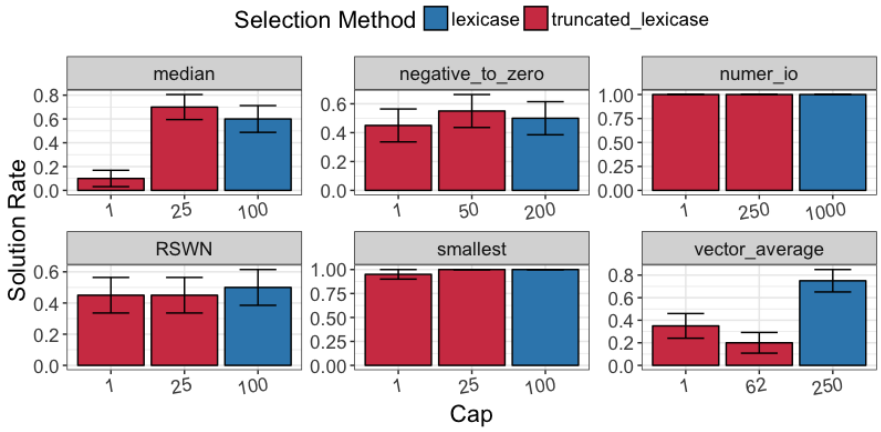

Fig. 2 Comparison of ordinary lexicase selection and truncated lexicase selection on software synthesis benchmark problems. The x-axis labels denote the values for the cap parameter of truncated lexicase selection in number of test cases. The values are a single test case, 25% of the total number of test cases, and 100% of the total number of test cases (which is ordinary lexicase selection). Error bars indicate the standard error.

8 Relation to many-objective optimization

All of the lexicase selection relaxation algorithms discussed above assess programs with respect to multiple test cases, each of which might be considered, in some sense, to be a separate objective of the genetic programming search process. Because there is existing research on so-called "many-objective optimization algorithms," which attempt to optimize four or more objectives, it is worth considering how such research relates to the parent selection algorithms we have discussed here.

We recently provided a multi-objective analysis of lexicase selection to show that, if we treat each fitness case as an objective, parents selected by lexicase selection are Pareto-optimal (i.e., they are non-dominated in the population) and located at the boundaries of the Pareto front [11]; we borrow from that analysis in the remainder of this section. The utility of Pareto dominance as a search driver is reduced as the number of fitness cases/objectives grows since the number of non-dominated solutions grow exponentially with the number of objectives. However, it is noteworthy that lexicase selection corresponds to selections of the boundaries of the Pareto front since boundary solutions influence typical measures of quality in many-objective optimization.

In many-objective optimization, the performance of algorithms is typically assessed in terms of convergence, uniformity, and spread [13], with the last of these dealing directly with the extent of boundary solutions. Some indicator-based methods, for example IBEA and SMS-EMOA, use a measure of the hypervolume in objective space to evaluate algorithm performance [23]. There appears to be some disagreement regarding the importance of boundary solutions in this context. Although Deb et. al [2] argued empirically that boundary solutions have an outsized effect on hypervolume measures, according to Auger et. al., "optimizing the unweighted hypervolume indicator stresses the so-called knee-points—parts of the Pareto front decision-makers believe to be interesting regions... Extreme points are not generally preferred as claimed in [2], since the density of points does not depend on the position on the front but only on the gradient at the respective point" [1].

In general, many-objective optimization methods have not highlighted random sampling as a useful method for exploring high-dimensional objective spaces. MOEA/D, R-NSGA-II, and NSGA-III opt for the use of reference points in objective space to preserve the spread of solutions. Unlike lexicase selection, there does not seem to be explicit motivation to keep boundary solutions in this literature. Results like those demonstrated in the present chapter, and in other work on lexicase selection, suggests that it may be worthwhile for many-objective optimization researchers to give greater consideration to methods based on random sampling, and to methods in which samples are considered with randomized priorities.

9 Discussion

In ordinary lexicase selection, test case order is randomized, allowing the algorithm to implicitly select on the basis of performance on all subsets of test cases, even though the exponentially many subsets are not considered explicitly. Randomization samples the space of test case sequence prefixes, thereby sampling the space of test case combinations that "matter" for a particular parent selection event, and thereby implicitly considering, over time, all subsets of test cases.

This appears to be a powerful technique for selecting useful parents in genetic programming, as it sometimes allows solutions to be found in significantly more runs and/or fewer generations than they can be found with previous, test-case-aggregating parent selection algorithms. However, the ordinary lexicase selection algorithm appears to be too strict in some circumstances, preventing the selection of parents that have much to offer to future generations in the evolutionary search process.

Epsilon lexicase selection appears to resolve this issue for many problems with floating-point errors by relaxing the requirement for eliteness in each step of the filtering of candidate parents. Two closely related methods were also explored here. The first, random threshold lexicase selection, is often (but not necessarily always) a more (and randomly more) relaxed version of epsilon lexicase selection, which also depends less directly on the distribution of errors in the population than does epsilon lexicase selection. The second, MADCAP epsilon lexicase selection, is a less (and randomly less) relaxed version of epsilon lexicase selection. Neither of these alternatives performed better than epsilon lexicase selection on the problems studied here, but they both performed better than several of the other methods considered, and one can imagine situations in which each would be more useful. For example, one might expect random threshold lexicase selection to be useful in contexts requiring particularly broad exploration, and one might expect MADCAP epsilon lexicase selection to be useful in contexts requiring convergence to solutions with zero or truly minimal errors. One area for future research is the application of these methods to problems of other types, in order to support or to falsify such expectations.

Truncated lexicase selection provides a different form of relaxation, limiting the number of test cases that can be considered in any parent selection event. While we were initially motivated to perform experiments on truncated lexicase selection by an expectation that the truncation might provide runtime performance benefits, we also wanted to study how this affects solution rates. Our experiments here, while preliminary, suggest that truncation often hampers the ability of the system to find solutions, but that it does so to a different extent on different problems. These results suggest that more experiments should be conducted, on additional problems of various types, to help us to understand when this form of relaxation might be beneficial (regarding success rates and/or runtime) and when it is detrimental. They also suggest that experiments should be conducted with additional truncation levels. Levels of 50% or higher may be particularly interesting, as they would provide only modest relaxation relative to ordinary lexicase selection. Levels that allow the consideration of very few test cases, but more than just a single one, may also be

revealing because they would be quite relaxed while still allowing for selection on the basis of multiple test cases.

Another obvious avenue for future research is to combine truncation with relaxation of the eliteness constraints for individual test cases. That is, it would be straightforward and possibly useful or at least instructive to conduct runs with truncated versions of epsilon lexicase selection, random threshold lexicase selection, and MADCAP epsilon lexicase selection.

Other forms of relaxation are also possible and may be useful in some circumstances. Two that we have implemented but not yet studied systematically are methods in which we perform lexicase selection with a certain probability and purely random selection otherwise, and methods in which we perform truncated lexicase selection but with different, randomly chosen numbers of test cases used in each selection event. Of course, these could also be combined with one another, and with many of the other methods described above. The choice of which of these to explore first might best be guided by theoretical consideration of the ways in which they sample the search space, along the lines of the initial discussion that we offered above on the relation between work on lexicase selection and work on many-objective optimization.

We note that many of the ideas discussed here could be applied to survival selection as well as parent selection.

While all of the methods here in some way relax the requirements made by ordinary lexicase selection, it is possible that strengthening the requirements in some way could potentially have benefits for some problems. It is unclear at this point what such strengthening would look like, but variants of this sort would certainly be interesting to examine.

The specific methods described here appear to have varying utility, at least from the experiments conducted to date. Regardless of the utility of the specific methods, however, we hope that the discussion here may help to stimulate additional work developing selection algorithms that can guide evolution more effectively.

Acknowledgements This material is based upon work supported by the National Science Foundation under Grants No. 1617087, 1129139 and 1331283. 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

1. Anne Auger, Johannes Bader, Dimo Brockhoff, and Eckart Zitzler. Theory of the hypervolume indicator: optimal -distributions and the choice of the reference point. In *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, pages 87–102. ACM, 2009.
2. Kalyanmoy Deb, Manikanth Mohan, and Shikhar Mishra. Evaluating the -Domination Based Multi-Objective Evolutionary Algorithm for a Quick Computation of Pareto-Optimal Solutions. *Evolutionary Computation*, 13(4):501–525, December 2005.

3. Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. Lexicase selection for program synthesis: A diversity analysis. In Rick Riolo, William P. Worzel, M. Kotanchek, and A. Kordon, editors, *Genetic Programming Theory and Practice XIII*, Genetic and Evolutionary Computation, Ann Arbor, USA, May 2015. Springer.
4. Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. Effects of lexicase and tournament selection on diversity recovery and maintenance. In Tobias Friedrich, Frank Neumann, Andrew M. Sutton, Martin Middendorf, Xiaodong Li, Emma Hart, Mengjie Zhang, Youhei Akimoto, Peter A. N. Bosman, Terry Soule, Risto Miikkulainen, Daniele Loiacono, Julian Togelius, Manuel Lopez-Ibanez, Holger Hoos, Julia Handl, Faustino Gomez, Carlos M. Fonseca, Heike Trautmann, Alberto Moraglio, William F. Punch, Krzysztof Krawiec, Zdenek Vasicek, Thomas Jansen, Jim Smith, Simone Ludwig, JJ Merelo, Boris Naujoks, Enrique Alba, Gabriela Ochoa, Simon Poulding, Dirk Sudholt, and Timo Koetzing, editors, *GECCO '16 Companion: Proceedings of the Companion Publication of the 2016 Annual Conference on Genetic and Evolutionary Computation*, pages 983–990, Denver, Colorado, USA, 20-24 July 2016. ACM.
5. Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. The impact of hyperselection on lexicase selection. In Tobias Friedrich, editor, *GECCO '16: Proceedings of the 2016 Annual Conference on Genetic and Evolutionary Computation*, pages 717–724, Denver, USA, 20-24 July 2016. ACM. Nominated for best paper.
6. Thomas Helmuth and Lee Spector. General program synthesis benchmark suite. In Sara Silva, Anna I Esparcia-Alcazar, Manuel Lopez-Ibanez, Sanaz Mostaghim, Jon Timmis, Christine Zarges, Luis Correia, Terence Soule, Mario Giacobini, Ryan Urbanowicz, Youhei Akimoto, Tobias Glasmachers, Francisco Fernandez de Vega, Amy Hoover, Pedro Larranaga, Marta Soto, Carlos Cotta, Francisco B. Pereira, Julia Handl, Jan Koutnik, Antonio Gaspar-Cunha, Heike Trautmann, Jean-Baptiste Mouret, Sebastian Risi, Ernesto Costa, Oliver Schuetze, Krzysztof Krawiec, Alberto Moraglio, Julian F. Miller, Pawel Widera, Stefano Cagnoni, JJ Merelo, Emma Hart, Leonardo Trujillo, Marouane Kessentini, Gabriela Ochoa, Francisco Chicano, and Carola Doerr, editors, *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1039–1046, Madrid, Spain, 11-15 July 2015. ACM.
7. Thomas Helmuth, Lee Spector, and James Matheson. Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643, October 2015.
8. Krzysztof Krawiec and Una-May O'Reilly. Behavioral programming: A broader and more detailed take on semantic gp. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 935–942, New York, NY, USA, 2014. ACM.
9. Krzysztof Krawiec, Jerry Swan, and Una-May O'Reilly. Behavioral program synthesis: Insights and prospects. In *Genetic Programming Theory and Practice XIII*, Genetic and Evolutionary Computation. Springer, 2015.
10. William La Cava, Kourosh Danai, and Lee Spector. Inference of compact nonlinear dynamic models by epigenetic local search. *Engineering Applications of Artificial Intelligence*, 55:292–306, October 2016.
11. William La Cava, Thomas Helmuth, Lee Spector, and Jason H. Moore. ε-Lexicase selection: a probabilistic and multi-objective analysis of lexicase selection in continuous domains. *arXiv:1709.05394 [cs]*, September 2017. arXiv: 1709.05394.
12. William La Cava, Lee Spector, and Kourosh Danai. Epsilon-lexicase selection for regression. In Tobias Friedrich, editor, *GECCO '16: Proceedings of the 2016 Annual Conference on Genetic and Evolutionary Computation*, pages 741–748, Denver, USA, 20-24 July 2016. ACM.
13. Miqing Li and Jinhua Zheng. Spread assessment for evolutionary multi-objective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 216–230. Springer, 2009.
14. M. Lichman. UCI machine learning repository, 2013.
15. Pawel Liskowski, Krzysztof Krawiec, Thomas Helmuth, and Lee Spector. Comparison of semantic-aware selection methods in genetic programming. In Colin Johnson, Krzysztof

Krawiec, Alberto Moraglio, and Michael O'Neill, editors, *GECCO 2015 Semantic Methods in Genetic Programming (SMGP'15) Workshop*, pages 1301–1307, Madrid, Spain, 11-15 July 2015. ACM.

16. Samir W Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, 1995.
17. Yuliana Martnez, Enrique Naredo, Leonardo Trujillo, Pierrick Legrand, and Uriel Lpez. A comparison of fitness-case sampling methods for genetic programming. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(6):1203–1224, 2017.
18. Nicholas Freitag McPhee, Brian Ohs, and Tyler Hutchison. Semantic building blocks in genetic programming. In *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 134–145, Naples, 26-28 March 2008. Springer.
19. Michael Schmidt and Hod Lipson. Age-fitness pareto optimization. In *Genetic Programming Theory and Practice VIII*, pages 129–146. Springer, 2011.
20. Lee Spector. Assessment of problem modality by differential performance of lexicase selection in genetic programming: A preliminary report. In Kent McClymont and Ed Keedwell, editors, *1st workshop on Understanding Problems (GECCO-UP)*, pages 401–408, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM.
21. Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.
22. Sarah Anne Troise and Thomas Helmuth. Lexicase selection with weighted shuffle. In *Genetic Programming Theory and Practice XV*, Genetic and Evolutionary Computation, Ann Arbor, USA, May 2017. Springer.
23. Tobias Wagner, Nicola Beume, and Boris Naujoks. Pareto-, Aggregation-, and Indicator-Based Methods in Many-Objective Optimization. In *Evolutionary Multi-Criterion Optimization*, pages 742–756. Springer, Berlin, Heidelberg, March 2007. DOI: 10.1007/978-3-540-70928-2_56.