

DAlex: Lexicase-like Selection via Diverse Aggregation^{*}

Andrew Ni¹[0000–0003–3480–4536], Li Ding²[0000–0002–1315–1196], and Lee Spector^{3,4}[10000–0001–5299–4797]

¹ Amherst College, Amherst MA 01002, USA ani24@amherst.edu

² University of Massachusetts Amherst, Amherst, MA 01003, USA
lding@umass.edu

³ Amherst College, Amherst, MA 01002, USA

⁴ University of Massachusetts Amherst, Amherst, MA 01003, USA
lspector@amherst.edu

Abstract. Lexicase selection has been shown to provide advantages over other selection algorithms in several areas of evolutionary computation and machine learning. In its standard form, lexicase selection filters a population or other collection based on randomly ordered training cases that are considered one at a time. This iterated filtering process can be time-consuming, particularly in settings with large numbers of training cases, including many symbolic regression and deep learning applications. In this paper, we propose a new method that is nearly equivalent to lexicase selection in terms of the individuals that it selects, but which does so in significantly less time. The new method, called DAlEx (for Diversely Aggregated Lexicase selection), selects the best individual with respect to a randomly weighted sum of training case errors. This allows us to formulate the core computation required for selection as matrix multiplication instead of recursive loops of comparisons, which in turn allows us to take advantage of optimized and parallel algorithms designed for matrix multiplication for speedup. Furthermore, we show that we can interpolate between the behavior of lexicase selection and its “relaxed” variants, such as epsilon and batch lexicase selection, by adjusting a single hyperparameter, named “particularity pressure,” which represents the importance granted to each individual training case. Results on program synthesis, deep learning, symbolic regression, and learning classifier systems demonstrate that DAlEx achieves significant speedups over lexicase selection and its relaxed variants while maintaining almost identical problem-solving performance. Under a fixed computational budget, these savings free up resources that can be directed towards increasing population size or the number of generations, enabling the potential for solving more difficult problems.

Keywords: Lexicase Selection · Learning Classifier Systems · Genetic Programming · Symbolic Regression · Deep Learning

^{*} Supported by Amherst College and members of the PUSH lab.

1 Introduction

Genetic algorithms [26] is a subfield of evolutionary computation that uses concepts from biological evolution—random variation and fitness-based survival—to solve a wide array of difficult problems. Genetic programming (GP) is a subfield of genetic algorithms that evolves programs or functions that take some inputs and produce some outputs. Common subfields of GP include software synthesis, in which a population of programs evolves to satisfy user-defined training cases, and symbolic regression (SR), in which a population of mathematical expressions evolves to fit a regression dataset. Central to these genetic algorithms is the concept of parent selection, in which members of the current population with desirable characteristics are chosen as the starting point from which to create the next generation of individuals. Lexicase selection [24] and epsilon lexicase selection [32] are state-of-the-art selection algorithms developed for the discrete-error domain and the continuous-error domain, respectively. The key idea behind lexicase selection is that it can be helpful to disaggregate the fitness function, selecting parents by considering training cases one at a time in a random order, all the while filtering out individuals which are not elite relative to the other remaining individuals on the current training case. This selection method has been shown to maintain beneficial diversity in evolved populations [36,17], especially in terms of “specialists”: individuals which may have high total error but have very low errors on a subset of the training cases [19,20]. However, due to its iterative nature, lexicase selection can often take a long time, and there is not an obvious way to take advantage of parallelism or single-instruction-multiple-data (SIMD) architectures to speed up its runtime.

In this work, we reexamine the lexicase intuition behind disaggregating the fitness function and we develop an efficient selection method based on randomly aggregating the fitness function at each individual selection event. Specifically, we quantify the idea that training cases occurring earlier in a lexicase ordering exert greater selection pressure by taking a randomly weighted average of each individual’s error vector. We formulate this selection mechanism as a matrix multiplication, which allows us to take advantage of modern advancements in matrix multiplication algorithms and SIMD architectures to achieve significantly faster runtime compared to lexicase selection.

This paper is organized as follows: In Section 2, we give a brief overview of lexicase selection and its variants. In Section 3, we describe the Diversely Aggregated Lexicase Selection (DALex) algorithm and give a brief theoretical treatment of its properties. In Section 4, we describe our experiments and results in three popular domains in which lexicase selection has seen demonstrated success. Empirical results show that DALex replicates the problem-solving success of lexicase selection and its most successful variants while offering significantly reduced runtime.

2 Background and Related Work

Lexicase selection is a parent selection method that assesses individuals based on each training case in turn, instead of constructing a single scalar fitness value for each individual [24]. During each individual selection event, the training cases are randomly shuffled. For each training case in the order determined by the random shuffle, candidate individuals in the population that are not “elite” with respect to that training case, i.e., have an error greater than the minimum error on that training case among the remaining candidates, are filtered out. If at any point only a single individual remains, then that individual is selected. If all of the training cases are exhausted, then a random individual is selected from those individuals still remaining. In other words, lexicase selection chooses an individual based on the first training case, using the rest of the cases in order to break ties. This mechanism has been shown to better maintain population diversity than methods based on aggregated fitness measures [17], which has been shown to improve problem-solving performance [15]. Lexicase selection and its variants have been successfully applied in many diverse problem domains beyond software synthesis, such as learning classifier systems [1], symbolic regression [32], SAT solvers [35], deep learning [9], and evolutionary robotics [44].

In continuous-valued domains like SR, it is often unlikely for more than one individual to share the minimum error on a training case. Therefore, the selection pressure exerted by each training case will be too large, and lexicase selection will not proceed beyond one training case. To combat this, the epsilon-lexicase selection method was developed as a relaxation of lexicase selection [32]. Instead of requiring individuals to have the minimum error out of the current candidates on a training case, epsilon-lexicase selection filters out individuals with errors exceeding an epsilon threshold above the minimum error. This epsilon threshold is adaptively determined by the population dynamics, specifically as the median of the absolute deviations from the median error on each training case.

Batch-lexicase selection [1,34,40] is another lexicase relaxation with improved performance on noisy datasets. Instead of considering each training case in turn, batch-lexicase selection considers groups of training cases, filtering out individuals whose average accuracy on that group is lower than some threshold. Batch-lexicase selection has been successfully applied to the field of Learning Classifier Systems (LCS) in Aenugu et al. [1], showing improved performance on noisy datasets compared to lexicase selection.

There have been many attempts to speed up lexicase selection. Dolson et al. [10] showed that the exact calculation of lexicase selection probabilities is NP-hard. However, Ding et al. [8] are able to calculate an approximate probability distribution from which to sample individuals. They show that their selection method achieves significantly faster runtime while achieving similar selection frequencies compared to lexicase selection. Ding et al. [7] also propose to use a weighted shuffle of training cases so that more difficult cases are considered first. They show that this technique can significantly reduce the number of training cases considered while suffering minimal degradation of problem-solving power. Batch tournament selection (BTS) [34] orders the training cases by difficulty,

then groups them into batches. BTS then performs tournament selection once per batch of cases, using the average error on that batch as the fitness function. They show that BTS achieves similar problem-solving performance to epsilon lexicase selection on SR datasets with significantly faster runtime.

Furthermore, many selection methods based on aggregated fitness measures have been proposed for multiobjective genetic algorithms. Fitness sharing is a selection method that attempts to maintain diversity by penalizing individuals for being too close to other individuals in the population with respect to a user-defined distance metric [6]. On the other hand, implicit fitness sharing (IFS) does not require a distance metric, and instead scales the reward for each training case by the rest of the population’s performance on that training case before computing the aggregated fitness function [33]. Historically assessed hardness (HAH) is another form of fitness sharing in which errors on each training case are scaled by the population’s historical success rate on that problem [28]. In addition to using a distance metric, NSGA-II sorts individuals into pareto fronts and conducts tournament selection using nondomination rank and local crowding factor as fitness values [5].

This work lies at the intersection of the lexicase variants mentioned above but takes a different direction compared to the speedup methods developed so far. For each individual selection event, we sample a random weighting of training cases. Then, for each individual, we compute a weighted average of the training case errors and select the individual with the lowest error with respect to the weighted average. In contrast to IFS or HAH, we use weights that are randomly chosen, span many more orders of magnitude, and are resampled at each selection event instead of at each generation. In contrast to NSGA-II, this method only selects non-dominated individuals because all of the training case weights are positive after the softmax operation. However, like epsilon and batch lexicase selection, and in contrast to lexicase selection, this method may select individuals outside of the “Pareto boundary” as defined in La Cava et al. [30]. We formulate this selection method as a vectorized matrix multiplication, selecting all individuals in a single, batched selection event. This allows us to take advantage of modern SIMD architecture and algorithmic advances in matrix multiplication to achieve significantly faster runtime compared to lexicase selection.

3 Diversely Aggregated Lexicase Selection

3.1 Description

Diversely Aggregated Lexicase Selection (DALex) operates by selecting the individual with the lowest average error with respect to a randomly sampled set of weights. It is parameterized by the shape and scale of the distribution from which these weights are sampled. In most of our experiments, we fix the shape to that of the normal distribution and vary the scale by changing the standard deviation of the distribution. For an ablation experiment exploring the effect of different distributions on the performance of DALex, see appendix C. We start with a population of n individuals, where each individual is evaluated on m

training cases. We sample an importance score for each training case from the distribution $\mathcal{N}(0, \text{std})$ where the standard deviation `std` is a tunable hyperparameter which we call the *particularity pressure*. Generally speaking, a training case’s importance score quantifies how many times more important it is compared to the rest of the training cases. To obtain the training case weights, we softmax the importance scores. This allows us to turn differences between the values of importance scores into differences between the magnitudes of the training case weights. For each individual, we compute its average weighted error as the dot product of the training case weight vector and the individual’s error vector. Finally, we select the individual with the lowest average weighted error.

In practice, we conduct all individual selection events in one batched selection event, combining multiple weight vectors into a weight matrix and using matrix multiplication in place of the dot product. We report selection runtimes in terms of this batched selection event, i.e. the time taken to go from a population of n individuals to the indices of the n selected parents for the next generation. We also perform an initial “pre-selection” step that differs slightly from other lexicase selection implementations due to the batched nature of our method: we group individuals into equivalence classes based on their error vectors, select equivalence classes using DALex, and then choose a random individual from each selected class [16].

For a population of n individuals evaluated on m training cases, there will be $k \leq n$ distinct equivalence classes, so the algorithm multiplies a $k \times m$ error matrix with a transposed $n \times m$ weight matrix, giving an asymptotic runtime of $O(m^2n)$, which is the same as that of lexicase selection. However, due to advances in the theory of matrix multiplication, the runtime of DALex can be reduced to $O(n^{\omega(\log_n m)})$ using methods such as the Coppersmith-Winograd algorithm [4].

3.2 Intuition

For a better understanding of DALex, we provide two intuitions linking it to lexicase selection.

First, we argue that DALex prompts us to reconsider a central tenet of lexicase selection. The success of lexicase selection is commonly attributed to its disaggregation of the fitness function. As the reasoning goes, aggregated selection methods like tournament or fitness-proportional selection are unable to maintain effective problem solving diversity the same way lexicase selection does. To rationalize our results on DALex, we instead propose the slightly different view that the discrepancy in problem-solving performance is due to the single aggregation event preceding these selection methods, which causes a loss of information. Because lexicase selection utilizes the entire high-dimensional error vector instead of simply the average or sum, it is able to more accurately identify promising individuals. Since DALex uses many diverse aggregation events, it is also able to fully utilize the information contained in the individuals’ error vectors, and is therefore able to replicate the success of lexicase selection. Even though each individual selection event operates on a single-dimensional total error, diversity

arises from different random weights promoting different individuals in different selection events.

Second, we draw a correspondence between the random orderings in lexicase selection and the random weightings in DALex. Given a random ordering of training cases in lexicase selection, the cases occurring earlier in the ordering exert more control over which individual is chosen at that selection event, i.e. have higher selection pressure. The first training case is paramount, and only if multiple individuals are equally good at the first case do we consider the second case, and so on. In DALex, the importance scores or training case weights assign explicit values to this notion of selection pressure. Given training case weights of $[1.0, 0.01, 0.1]$ for example, the first case would be ten times as important as the third case, which in turn is ten times as important as the second case. In the limit of infinite particularity pressure, the differences in importance scores tend towards infinity, so the difference in magnitudes of training case weights tends to infinity, and we recover lexicase selection. As the number of training cases increases or the range of magnitudes spanned by the errors on each case increases, the particularity pressure needed to replicate lexicase selection increases. Empirically, we are able to replicate lexicase selection using modest particularity pressures such as 20.

This interpretation also explains why we call the standard deviation hyperparameter the particularity pressure. When importance weights are sampled with a high standard deviation, the most important training case will be much more important than the second most, and so on, just like in lexicase selection. As the standard deviation decreases, the importance of a single training case decreases. In other words, higher particularity pressures correspond to increasingly lexicase-like selection dynamics[42]. We find that we can achieve similar successes to relaxed forms of lexicase selection such as batch-lexicase or epsilon-lexicase selection simply by choosing an appropriate particularity pressure. In this sense, DALex unifies the diverse selection methods lexicase, epsilon-lexicase, and batch-lexicase selection into a single selection method. It is important to note that DALex’s only theoretical guarantee is that of asymptotically lexicase-like selection with infinite particularity pressure. As decreasing the particularity pressure is a different way of relaxation compared to those proposed for epsilon-lexicase and batch-lexicase selection, we do not expect to be able to replicate the selection dynamics of these lexicase variants to an arbitrary degree of accuracy. However, empirical results show that varying the particularity pressure gives enough flexibility to replicate the successes of these relaxed lexicase variants on the problem domains for which they were designed.

3.3 Modifications

For domains amenable to a relaxed lexicase variant such as epsilon-lexicase or batch-lexicase selection, we first standardize the population errors on each training case to have zero mean and unit variance. While not necessary, we find that this normalization step helps DALex perform well across many problems with a single hyperparameter setting.

Of the problem domains we study, Learning Classifier Systems (LCS) [46] has a significantly different structure and therefore requires further modification. Since an LCS individual represents a rule that matches a subset of the training cases, each individual will only have errors defined on a subset of the training cases. For these problems, we let the individual have error 0 on cases for which it is undefined. We also use the individual’s support vector, which has a 1 for each training case on which the individual is defined, and a 0 otherwise. To compute the individual’s average error, we take the dot product of the error vector with the weight vector and then normalize by dividing this value by the dot product of the support vector with the weight vector.

Algorithm 1: Diversely Aggregated Lexicase Selection, batched selection event

Data:

- $E = \{e_{i,j}\}$ the error value of individual i on training case j , or 0 if individual i is not defined on training case j
- $N = \{s_{i,j}\}$ the support of individual i on training case j , which equals 1 if individual i is defined on case j , otherwise 0
- n the number of selection events, m the number of training cases
- **particularity_pressure**, the standard deviation of the sampled weights
- **Relaxed**, whether to simulate a relaxed version of lexicase selection.

Result:

- **idx**, the n indices of the selected individuals to be the parents of the next generation.

if Relaxed then

| $E \leftarrow \frac{E - \text{mean}(E, \text{axis}=0)}{\text{std}(E, \text{axis}=0)}$ The standardized error matrix

end

$I \leftarrow \mathcal{N}(0, \text{particularity_pressure})$ The importance scores, an i.i.d gaussian matrix of size $[n, m]$

$W \leftarrow \text{softmax}(I, \text{axis} = 1)$ The training case weights

$F \leftarrow \frac{EW^T}{SW^T}$ The normalized, weighted, average error for each individual

idx $\leftarrow \text{argmin}(F, \text{axis} = 0)$ lowest error individuals w.r.t the weights W

return idx

In problems where individuals are defined on all training cases, the support is 1 everywhere, so the normalization constant SW^T is simply the sum of the training case weights, which is 1 due to the softmax operation. Under this condition, our algorithm degenerates to the simpler case considered above. The complete algorithm pseudocode, with the augmentations described above to accommodate for LCS and SR problems, is described in Algorithm 1. In our pseudocode, we define arithmetic operations as the vectorized element-wise operations used in libraries like **numpy**, and batch together the n importance score vectors for the n individual selection events into an $n \times m$ matrix.

4 Experiments and Results

4.1 CBGP

We first compare DALex to lexicase selection on program synthesis problems, the class of problem for which lexicase selection was first developed. While we use a particularity pressure of 20 for these experiments to illustrate the robustness of DALex, we recommend setting the particularity pressure as large as possible within the range of floating point precision, or around 200, for problems suitable for lexicase selection. As DALex can exactly simulate lexicase selection in the limit of infinite particularity pressure, we also assess the ability of DALex to replicate the lexicase selection probability distribution with a moderate particularity pressure. To situate our results with respect to other lexicase approximations we also compare to plexicase selection¹[8]. For plexicase selection, we use the hyperparameter setting $\alpha = 1$. We use the Code-Building Genetic Programming (CBGP) system developed by Helmuth et al. [37] and test on a suite of problems from the PSB1 Benchmark [21] on which lexicase selection has shown good performance. For an overview on CBGP, see appendix A. We follow the default settings in CBGP, evolving 1000 individuals for 300 generations and using UMAD [18] with a rate of 0.09 as the sole variation operator. We present results in both the full data and downsampled paradigms. In the full data paradigm, all individuals are evaluated on all training cases. In the downsampled paradigm, a subset of the training cases is sampled at each generation and used to evaluate individuals in that generation. Downsampling has been shown to benefit the performance of Genetic Programming as we can increase the number of generations and/or individuals in the population for a given total computational budget [23,22,25]. Additional downsampling methods utilizing information from the population to choose the downsampled cases have been proposed [2], but we only study the randomly downsampled paradigm. In our experiments, we use a downsampling rate of 25%. The specific problems we study are **Compare String Lengths** (CSL), **Median**, **Number IO**, **Replace Space with Newline** (RSWN), **Smallest**, **Vector Average**, and **Negative to Zero** (NTZ).

Following the recommendation of the PSB1 benchmark, we report problem-solving performance as number of successful runs out of 100. We define a successful run as one which produces an individual with zero error on both the training cases and the unseen testing cases.²

Table 1 compares the success rates of DALex, plexicase, and lexicase selection on six benchmark problems under the full and downsampled paradigms. No results were significantly different between the selection methods, and DALex achieves very similar success rates to lexicase selection. For our ablation experiments on the distribution of the sampled importance scores, see appendix C. Additionally, we find that DALex runs faster than lexicase selection but slower than plexicase selection. Detailed results can be found in appendix B.

¹ Fixed a bug in the downsampling implementation in the released version of [8]

² Due to specific quirks of the code-building system, it is very difficult for CBGP to generalize successfully on **Compare String Lengths**.

Table 1: Success rates of GP runs on six benchmark problems in the full data and downsampled paradigms. The success rates across the board are very similar, and no results were significantly different between the selection methods. We perform chi-squared tests following [8] and underline results that were significantly worse than lexicase selection with ($p < 0.05$). No results were significantly better than lexicase selection.

Problem	Success Rate			Downsampled Success Rate		
	<i>Lexicase</i>	<i>DALex</i>	<i>Plexicase</i> ($\alpha = 1$)	<i>Lexicase</i>	<i>DALex</i>	<i>Plexicase</i> ($\alpha = 1$)
CSL ²	0	0	0	0	0	0
Median	91	91	<u>83</u>	100	100	60
Number IO	99	99	100	100	100	100
RSWN	12	15	6	66	68	<u>50</u>
Smallest	100	100	100	100	100	100
Vector Average	100	100	99	100	100	100
NTZ	78	83	80	99	100	100

To compare the selection dynamics of the three methods, we solve the same PSB1 problems using lexicase selection, and at every generation we sample 50,000 individuals using each selection method. From this we build up the empirical probability distributions $\{p_{it}\}$ the probability of selecting individual i at generation t via lexicase selection, $\{p'_{it}\}$ the probability of selecting individual i at generation t via DALex, and $\{p''_{it}\}$ the probability of selecting individual i at generation t via plexicase selection. To quantify the differences in probability distributions between lexicase selection and its approximations, we use the Jensen-Shannon divergence metric

$$D_t = \frac{1}{2} \left[\sum_i p_{it} \log \left(\frac{2p_{it}}{p_{it} + q_{it}} \right) + \sum_i q_{it} \log \left(\frac{2q_{it}}{p_{it} + q_{it}} \right) \right]$$

Where q_{it} is p'_{it} or p''_{it} for DALex and plexicase selection, respectively. The lower the JS-divergence of the DALex/plexicase selection probability distribution from the lexicase selection probability distribution, the better the approximation to lexicase selection. Furthermore, for each run in which lexicase selection produces a generalizing individual, we track that individual's lineage to find its ancestor a_t in each generation t , of which there is only one per generation since we only use mutation operators. For each of these individuals we compute the probability ratio $r_t = \frac{q_{a_t t}}{p_{a_t t}}$ quantifying how much more likely DALex and plexicase selection are to select the successful lineage than lexicase selection. We report the average of these two metrics over all generations and all runs.

Figure 1 shows the results for these two metrics on the problems studied. In almost all problems, the probability of selecting the successful lineage is almost the same between DALex and lexicase selection. In contrast, the probability ratio of plexicase and lexicase selection often deviates much more from 1, indicating a less faithful approximation. Furthermore, the Jensen-Shannon divergences between the DALex and lexicase selection probability distributions are often very

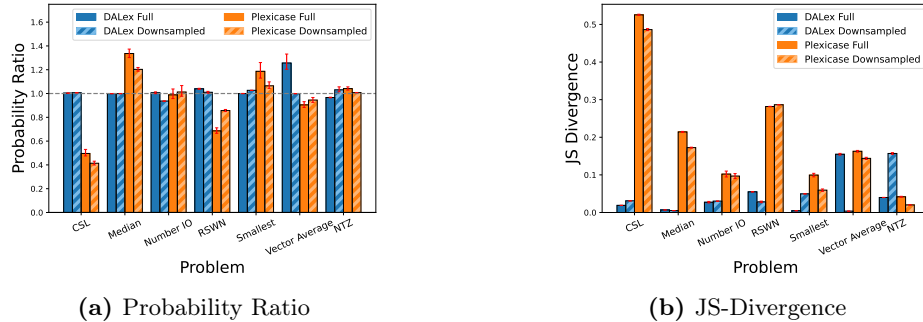


Fig. 1: Fidelity of the DALex and plexicase approximations to lexicase selection on CBGP problems. Error bars show the bootstrapped 95% confidence intervals. The ratio of selecting the successful lineage via DALex versus the probability using lexicase is very close to 1 in 6 out of 7 problems. The Jensen-Shannon divergence of the DALex selection probability distribution from the lexicase distribution is close to 0 in 5 out of 7 problems.

close to zero and much smaller than the corresponding divergences of plexicase selection from lexicase selection, indicating that DALex gives a close approximation of lexicase selection dynamics. As mentioned before, we expect increasing the value of the standard deviation to improve the fidelity of the DALex approximation, bringing the DALex JS divergences even lower and the DALex probability ratios even closer to 1.

4.2 Image Classification

To test the ability of DALex to mimic lexicase selection even with many training cases, we compare DALex against lexicase selection as the selection method in gradient lexicase selection. We use the popular VGG16 [39] and ResNet18 [14] neural networks on the CIFAR10 [29] dataset. This dataset is comprised of 32x32 bit RGB images distributed evenly across 10 classes. There are 50,000 training instances and 10,000 test instances. As shown in Ding et al. [7], lexicase selection often uses up to the entire training set to perform selection, especially towards the end of the genetic algorithm.

Gradient lexicase selection[9] is a hybrid of lexicase selection with deep learning, in which a population of deep neural networks (DNN) is evolved to fit an image classification dataset. In gradient lexicase selection, stochastic gradient descent on a subset of the training cases is used as the mutation operator. Lexicase selection proceeds by evaluating the population of DNNs on one training case at a time and keeping only the DNNs that correctly predict the training case’s label. In contrast, DALex evaluates each individual on the entire training set, assigns to each training case an error value of 0 if the case was correctly predicted or 1 otherwise, and computes a weighted sum of the error values. As this domain is intended to test the limits of our lexicase approximation with its

large number of training cases, we use the previously recommended particularity pressure of 200.

Since DALex is appropriate for both discrete-valued and continuous-valued errors, we also investigate the possibility of using cross-entropy loss instead of accuracy as the basis of selection.

We use a population size of $p = 4$ and run evolution for a total of $200(p + 1)$ epochs. For a more detailed description of other hyperparameters such as the SGD learning rate schedule, see Ding et al. [9].

Each algorithm is run 10 times, and both the final test accuracy and algorithm runtime are recorded. All runs were done on a single Nvidia A100 gpu. We report the mean and standard deviation of these two metrics in Table 2 the form $\text{mean} \pm \text{std}$.

Table 2: Performance of three selection methods and two architectures on the CIFAR10 dataset. Vanilla SGD is also included as a baseline. DALex achieves similar test accuracy and lower runtime compared to lexicase selection.

Selector	VGG16		ResNet18	
	<i>Test Accuracy</i>	<i>Runtime (h)</i>	<i>Test Accuracy</i>	<i>Runtime (h)</i>
Vanilla SGD	92.8 ± 0.2	0.962 ± 0.003	93.8 ± 0.1	1.003 ± 0.006
Lexicase	93.7 ± 0.1	11.1 ± 0.3	95.1 ± 0.2	29 ± 1
DALex accuracy (std=200)	93.7 ± 0.2	8.7 ± 0.1	95.3 ± 0.1	9.1 ± 0.1
DALex losses (std=200)	93.7 ± 0.2	8.66 ± 0.02	95.2 ± 0.2	9.2 ± 0.1

We find that DALex performs just as well as lexicase selection whether selecting on cross-entropy losses or accuracy, and additionally runs much faster. This is likely due to the fact that lexicase selection evaluates individuals on each training case with a batch size of 1. The small batch size causes too much communication overhead between the cpu and gpu, resulting in a slower runtime. Furthermore, as training progresses and the models approach 100% accuracy on the training set, lexicase selection begins to exhibit its worst-case runtime. As the ResNet18 model reaches a higher training set accuracy than the VGG16 model, lexicase selection takes more than twice as long to run. With DALex, however, both models have consistent and much lower runtimes.

4.3 SRBench

To examine the ability of DALex to replicate epsilon lexicase-like behavior, we use the SRBench [31] benchmark suite for symbolic regression. Specifically, we compare DALex, plexicase, and epsilon lexicase selection as the selection method for the gplearn [45] framework and train on 20 subsampled datasets from the Penn Machine Learning Benchmark (PMLB) [38] suite with sizes ranging from 50 to 40,000 instances. As recommended by SRBench, we repeat each experiment 10 times with different random seeds, and take the median of each statistic

over these 10 experiments. For these experiments, we use the pre-tuned settings of 1000 individuals per generation, 500 generations, and functions drawn from the set $\{+, -, *, /, \log, \text{sqrt}, \sin, \cos\}$. We also use the default parsimony coefficient of 0.001, which adds a model size-based penalty to the error of each test case. We choose the DALex particularity pressure to be 3, as determined from a preliminary hyperparameter search.

The statistics we examine in this paper, following the SRBench defaults, are test R^2 , training time, and model complexity. Unlike in program synthesis, we find that the runtime of epsilon lexicase dominates the algorithm runtime, so we expect the choice of selection method to significantly impact the overall runtime. We report the median of these statistics over all 20 problems, using bootstrapping to obtain a 95% confidence interval.

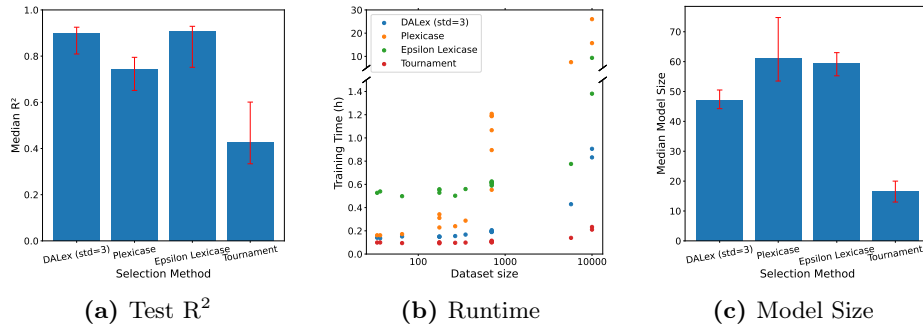


Fig. 2: Performance of three selection methods on 20 downsampled problems from the PMLB repository. DALex has a similar test R^2 to epsilon lexicase while having a lower runtime and generating more concise models.

Figure 2 shows the results for the 20 selected problems. We find that DALex gives a very similar test R^2 statistic to epsilon lexicase selection while generating significantly more concise models and taking much less time to train. Of the problems we studied, epsilon lexicase selection has a much higher runtime on the `344_mv` problem than on all other problems. We hypothesize that this is due to epsilon lexicase selection approaching its worst-case runtime, where it has to consider all test cases over all individuals. Helmuth et al. [20] find that lexicase selection often selects a single individual after considering only a small fraction of the total training cases, resulting in empirical runtimes that are much faster than the worst-case runtimes. On problems like `344_mv`, however, where the selection dynamics approach the worst-case situation, lexicase selection will have much higher runtimes than we’d expect from empirical results. In contrast, DALex has a consistent and low runtime that scales predictably with the length of individuals’ error vectors. The high runtimes and degraded approximation performance for plexicase on large datasets are new findings that call for additional investigation.

4.4 Learning Classifier Systems

Finally, to show that DALex can reproduce the success of batch-lexicase, we use the learning classifier system (LCS) from Aenugu et al. [1]. We choose the led24 dataset [3] for the stark contrast between the performances of lexicase selection and batch-lexicase selection. The led24 problem [3], taken from the UCI ML Repository, tests the ability of selection methods to cope with noise. This dataset, consisting of 3200 instances, contains 7 boolean attributes corresponding to whether each of 7 LED lights in a display is lit up. It has 10 categories, the digits 0 through 9, corresponding to which digit is shown on the LED display. In this dataset, each boolean attribute has a 0.1 probability having its value inverted. The evolved rule distribution for this problem is sparse, with most rules representing fewer than 5 data instances [7]. This property makes the led24 problem unsuitable for lexicase selection and motivates the development of batch-lexicase selection. For this domain, we use a particularity pressure of 20 as determined by a preliminary hyperparameter search.

As in Aenugu et al. [1], we conduct experiments in both the full data and partial data paradigms. In the full data paradigm, we use the entire led24 dataset, leaving out 30% for testing. In the partial data paradigm, we randomly downsample the dataset, removing 40% of the instances. We repeat this downsample for each run, meaning the instances available for each run will be different. We then split the remaining 60% of the instances into a 40% training set and a 20% testing set.

Figure 3 displays the results of our experiments. In both domains, we find that DALex has both the highest test accuracy of the four selection methods and generates the least number of distinct rules, which suggests that it may have better generalization ability. Finally, DALex has a significantly lower runtime in both domains than lexicase and batch-lexicase selection.

5 Conclusion and Future Work

In this work, we presented the novel parent selection method DALex, which efficiently approximates the lexicase selection mechanism by using random aggregations of error vectors. The proposed method, parametrized by the particularity pressure, is flexible enough to replicate the successes of lexicase selection and its relaxed variants on multiple problem domains, and is robust to very large datasets. Furthermore, it has a consistent and low runtime, which frees up resources that can be used for the other aspects of evolution such as number of generations. We make the code used in these experiments available at <https://github.com/andrewni420/DALex>.

While we have used reasonable presets for the standard deviation hyperparameter in DALex, we do not anticipate that a single hyperparameter setting will work for all cases or even for all generations during the course of a GP run. Therefore, future work could determine the optimal standard deviation using hyperparameter search or even a more radical paradigm such as autoconstruction

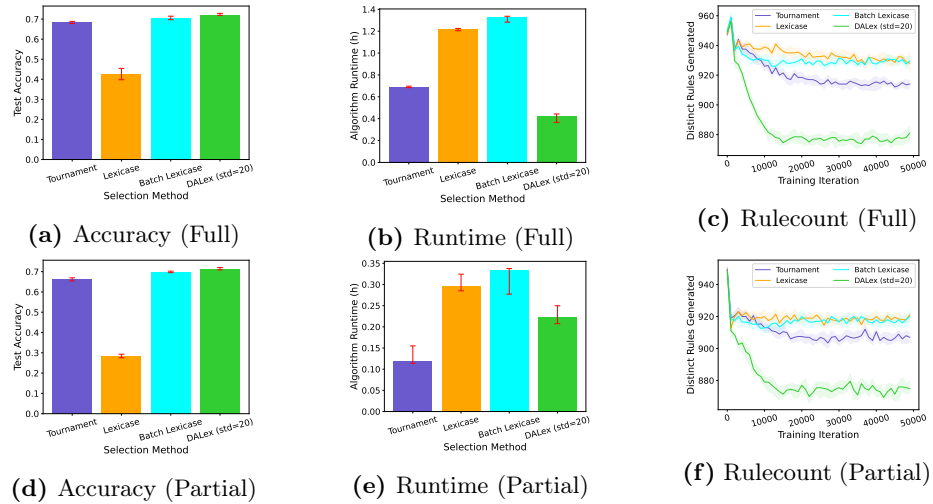


Fig. 3: Performance of four selection methods on the led24 problem in the full data and partial data scenarios. Bootstrapped 95% confidence intervals are displayed as error bars for the accuracy and runtime plots and as a shaded region for the rulecount plot. DALex has approximately equal test accuracy to batch lexicase while having much lower runtime and generating the fewest rules.

[41,43], Additionally, since DALex computes a randomly weighted fitness function, it can be combined with selection methods other than pure elitist selection, such as tournament or fitness-proportional selection [11] to increase diversity. In fact, the weighted aggregation formulation of DALex is not confined to parent selection algorithms. The concept of randomly weighted aggregation could also be used to transform the losses for each training case in deep learning. By introducing more stochasticity into training, this could enable models to be trained with large batch sizes without suffering performance degradation [27]. Alternatively, DALex could be used to randomly weight each node in the policy head of a reinforcement learner when sampling trajectories, improving the exploration of off-policy learning algorithms such as Double Q-learning [13] or SAC [12].

6 Acknowledgements

This work was performed in part using high-performance computing equipment at Amherst College obtained under National Science Foundation Grant No. 2117377. 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. The authors would like to thank Ryan Boldi, Bill Tozier, Tom Helmuth, Edward Pantridge and other members of the PUSH lab for their insightful comments and suggestions.

References

1. Aenugu, S., Spector, L.: Lexicase selection in learning classifier systems. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. p. 356–364. GECCO’19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3321707.3321828>
2. Boldi, R., Briesch, M., Sobania, D., Lalejini, A., Helmuth, T., Rothlauf, F., Ofria, C., Spector, L.: Informed down-sampled lexicase selection: Identifying productive training cases for efficient problem solving. *Evolutionary Computation* (2024), <https://doi.org/10.48550/arXiv.2301.01488>, to appear
3. Breiman, L., Friedman, J., Olshen, R., Stone, C.: LED Display Domain. UCI Machine Learning Repository (1988), <https://doi.org/10.24432/C5FG61>
4. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. p. 1–6. STOC’87, Association for Computing Machinery, New York, NY, USA (1987). <https://doi.org/10.1145/28395.28396>
5. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In: Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J.J., Schwefel, H.P. (eds.) *Parallel Problem Solving from Nature PPSN VI*. pp. 849–858. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
6. Deb, K., Goldberg, D.E.: An investigation of niche and species formation in genetic function optimization. In: *Proceedings of the Third International Conference on Genetic Algorithms*. p. 42–50. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)
7. Ding, L., Boldi, R., Helmuth, T., Spector, L.: Lexicase selection at scale. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. p. 2054–2062. GECCO’22, Association for Computing Machinery, New York, NY, USA (2022). <https://doi.org/10.1145/3520304.3534026>
8. Ding, L., Pantridge, E., Spector, L.: Probabilistic lexicase selection. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. p. 1073–1081. GECCO’23, Association for Computing Machinery, New York, NY, USA (2023). <https://doi.org/10.1145/3583131.3590375>
9. Ding, L., Spector, L.: Optimizing neural networks with gradient lexicase selection. In: *International Conference on Learning Representations* (2022), <https://doi.org/10.48550/arXiv.2312.12606>
10. Dolson, E.: Calculating lexicase selection probabilities is NP-hard. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO’23, ACM (Jul 2023). <https://doi.org/10.1145/3583131.3590356>
11. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, vol. 1, pp. 69–93. Elsevier (1991). <https://doi.org/https://doi.org/10.1016/B978-0-08-050684-5.50008-2>
12. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Dy, J., Krause, A. (eds.) *Proceedings of the 35th International Conference on Machine Learning*. Proceedings of Machine Learning Research, vol. 80, pp. 1861–1870. PMLR (10–15 Jul 2018), <https://proceedings.mlr.press/v80/haarnoja18b.html>
13. Hasselt, H.: Double Q-learning. In: *Advances in neural information processing systems*. vol. 23, pp. 2613–2621 (2010)

14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016)
15. Helmuth, T., Abdelhady, A.: Benchmarking parent selection for program synthesis by genetic programming. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. p. 237–238. GECCO’20, Association for Computing Machinery, New York, NY, USA (2020). <https://doi.org/10.1145/3377929.3389987>
16. Helmuth, T., Lengler, J., La Cava, W.: Population diversity leads to short running times of lexicase selection. In: Rudolph, G., Kononova, A.V., Aguirre, H., Kerschke, P., Ochoa, G., Tušar, T. (eds.) *Parallel Problem Solving from Nature – PPSN XVII*. pp. 485–498. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-14721-0_34
17. Helmuth, T., McPhee, N.F., Spector, L.: Lexicase selection for program synthesis: A diversity analysis. In: *Genetic Programming Theory and Practice XIII*. pp. 151–167. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-34223-8_9
18. Helmuth, T., McPhee, N.F., Spector, L.: Program synthesis using uniform mutation by addition and deletion. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. p. 1127–1134. GECCO’18, Association for Computing Machinery, New York, NY, USA (2018). <https://doi.org/10.1145/3205455.3205603>
19. Helmuth, T., Pantridge, E., Spector, L.: Lexicase selection of specialists. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. p. 1030–1038. GECCO’19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3321707.3321875>
20. Helmuth, T., Pantridge, E., Spector, L.: On the importance of specialists for lexicase selection. *Genetic Programming and Evolvable Machines* **21**(3), 349–373 (Sep 2020). <https://doi.org/10.1007/s10710-020-09377-2>
21. Helmuth, T., Spector, L.: General program synthesis benchmark suite. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. p. 1039–1046. GECCO’15, Association for Computing Machinery, New York, NY, USA (2015). <https://doi.org/10.1145/2739480.2754769>
22. Helmuth, T., Spector, L.: Explaining and Exploiting the Advantages of Down-sampled Lexicase Selection. vol. *Proceedings of ALIFE 2020: The 2020 Conference on Artificial Life*, pp. 341–349 (2020). https://doi.org/10.1162/isal_a_00334
23. Helmuth, T., Spector, L.: Problem-Solving Benefits of Down-Sampled Lexicase Selection. *Artificial Life* **27**(3–4), 183–203 (2022). https://doi.org/10.1162/artl_a_00341
24. Helmuth, T., Spector, L., Matheson, J.: Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation* **19**(5), 630–643 (2015). <https://doi.org/10.1109/TEVC.2014.2362729>
25. Hernandez, J.G., Lalejini, A., Dolson, E., Ofria, C.: Random subsampling improves performance in lexicase selection. In: *Proceedings of the 2019 Genetic and Evolutionary Computation Conference Companion*. p. 2028–2031. GECCO’19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3319619.3326900>
26. Holland, J.H.: *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press (1992)
27. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR* (2016), <http://arxiv.org/abs/1609.04836>

28. Klein, J., Spector, L.: Genetic programming with historically assessed hardness. In: *Genetic Programming Theory and Practice VI*, pp. 1–14. Springer (2008)
29. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Technical Report. University of Toronto (2009)
30. La Cava, W., Helmuth, T., Spector, L., Moore, J.H.: A Probabilistic and Multi-Objective Analysis of Lexicase Selection and ϵ -Lexicase Selection. *Evolutionary Computation* **27**(3), 377–402 (09 2019). https://doi.org/10.1162/evco_a_00224
31. La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F., Virgolin, M., Jin, Y., Kommenda, M., Moore, J.: Contemporary symbolic regression methods and their relative performance. In: Vanschoren, J., Yeung, S. (eds.) *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. vol. 1. Curran (2021)
32. La Cava, W., Spector, L., Danai, K.: Epsilon-lexicase selection for regression. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. p. 741–748. GECCO’16, Association for Computing Machinery, New York, NY, USA (2016). <https://doi.org/10.1145/2908812.2908898>
33. McKay, R.I.B.: Fitness sharing in genetic programming. In: *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*. p. 435–442. GECCO’00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
34. de Melo, V.V., Vargas, D.V., Banzhaf, W.: Batch tournament selection for genetic programming: The quality of lexicase, the speed of tournament. In: *Proceedings of the 2019 Genetic and Evolutionary Computation Conference*. p. 994–1002. GECCO’19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3321707.3321793>
35. Metevier, B., Saini, A.K., Spector, L.: Lexicase selection beyond genetic programming. In: *Genetic Programming Theory and Practice XVI*. pp. 123–136. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-04735-1_7
36. Moore, J.M., Stanton, A.: Tiebreaks and Diversity: Isolating Effects in Lexicase Selection. In: *Proceedings of ALIFE 2018: The 2018 Conference on Artificial Life*. pp. 590–597 (2018). https://doi.org/10.1162/isal_a_00109
37. Pantridge, E., Spector, L.: Code building genetic programming. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. p. 994–1002. GECCO’20, Association for Computing Machinery, New York, NY, USA (2020). <https://doi.org/10.1145/3377930.3390239>
38. Romano, J.D., Le, T.T., Cava, W.L., Gregg, J.T., Goldberg, D.J., Ray, N.L., Chakraborty, P., Himmelstein, D., Fu, W., Moore, J.H.: Pmlb v1.0: An open source dataset collection for benchmarking machine learning methods (2021). <https://doi.org/10.48550/arXiv.2012.00058>
39. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: *Proceedings of the International Conference on Learning Representations* (May 2015). <https://doi.org/10.48550/arXiv.1409.1556>
40. Sobania, D., Rothlauf, F.: Program synthesis with genetic programming: The influence of batch sizes. In: Medvet, E., Pappa, G., Xue, B. (eds.) *EuroGP 2022: Genetic Programming*. Lecture Notes in Computer Science, vol. 13223, pp. 118–129. Springer International Publishing, Cham (2022)
41. Spector, L.: Autoconstructive evolution: Push, PushGP, and Pushpop. In: *Proceedings of the 2001 Genetic and Evolutionary Computation Conference, GECCO-2001*. pp. 137–146. Morgan Kaufmann Publishers, San Francisco, CA (2001)
42. Spector, L., Ding, L., Boldi, R.: Particularity. In: *Genetic Programming Theory and Practice XX*. Springer (2023), to appear.

43. Spector, L., Robinson, A.: Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines* **3**(1), 7–40 (Mar 2002). <https://doi.org/10.1023/A:1014538503543>
44. Stanton, A., Moore, J.M.: Lexicase Selection for Multi-Task Evolutionary Robotics. *Artificial Life* **28**(4), 479–498 (11 2022). https://doi.org/10.1162/artl_a_00374
45. Stephens, T.: gplearn. <https://github.com/trevorstephens/gplearn> (2023)
46. Urbanowicz, R.J., Moore, J.H.: Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications* **2009**, 1–25 (2009). <https://doi.org/10.1155/2009/736398>

A Code-Building GP for Program Synthesis

Code Building Genetic Programming (CBGP)[37] is a typed genetic programming system that aims to use constructs like those found in human programming to solve program synthesis problems. Incorporating concepts like data types, data structures, and control flow into a GP system not only promises to create more complex hierarchical systems but also produces more interpretable programs which are analogous in structure to human created programs.

CBGP encodes programs as directed acyclic graphs (DAGs) where each leaf node is an input or a constant, and each inner node a function. Together, these nodes are called “expressions” and form a computational graph. In these graphs, the return type of child nodes are required to be sub-types of the corresponding argument in the parent node for type safety. The CBGP implementation used in this paper supports the expression types *Constants*, *Inputs*, *Functions*, *Methods*, *Constructors*, and *Higher Order Functions*.

While CBGP programs are represented by a graph structure, it still uses the linear plushy genome representation found in other GP systems like PushGP. This representation consists of a sequence of expression tokens representing program syntax and structure tokens representing the hierarchical structure of the program. These plushy genomes are compiled into Push programs which are then translated into CBGP DAGs for execution. As in PushGP, CBGP uses the Uniform Mutation by Addition and Deletion (UMAD) mutation operator[18] as the sole variation operator.

B Program Synthesis Runtime Comparisons

During our program synthesis experiments, we record the time taken to select the parents of the next generation. We then take the average of this metric over all generations and all runs and report them in Figure 4. The results show that DALex is up to 5x faster than lexicase selection. Despite having a similar or slower runtime compared to plexicase selection, DALex is a more faithful approximation to lexicase selection and has shown better performance as presented in Table 1, where plexicase selection sometimes produces worse results than lexicase selection.

C Ablation of DALex probability distributions

As an ablation experiment, we examine the effect of the distribution from which importance scores are chosen on the problem-solving ability of DALex. We compare our default selection from a normal distribution to selection from a uniform distribution and shuffling a range of evenly spaced numbers. The range case is interesting in that it is guaranteed to produce lexicase selection when the spacing between weights is larger than the magnitude difference between the largest

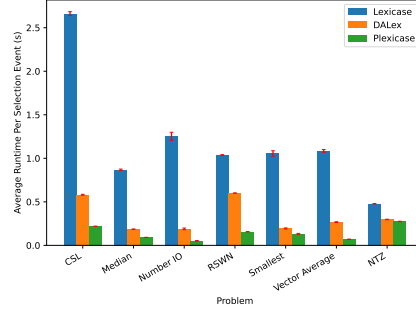


Fig. 4: Runtime comparison of three selection methods on program synthesis problems. 95% confidence intervals obtained by bootstrapping.

Table 3: Success rates of GP runs using different distributions for DAllex. All distributions were scaled to a standard deviation of 20 unless otherwise noted. We perform chi-squared tests following [8] and underline results that were significantly worse than the normal distribution with ($p < 0.05$). No results were significantly better than the normal distribution.

Problem	Success Rate		
	<i>Normal</i>	<i>Uniform</i>	<i>Range</i>
CSL	0	0	0
Median	91	88	87
Number IO	99	99	100
RSWN	15	15	18
Smallest	100	100	100
Vector Average	100	100	100
NTZ	83	<u>65</u>	<u>66</u>
NTZ (std=100)	79	85	72

possible error and the smallest possible nonzero error. In each case the importance scores are scaled so that they have a standard deviation of 20 prior to softmaxing.

The results are shown in Table 3. The different distribution choices perform almost identically on the problems considered except for the **Negative To Zero** problem. We hypothesize that this problem is more taxing for our lexicase approximations, and highlights the differences between the distributions used for DALex. Of the three distributions, the normal distribution has the largest kurtosis. Therefore, the largest importance scores sampled from the normal distribution will be farther apart than for the other two distributions. Since lexicase selection often selects an individual after considering only a fraction of the training cases [20], it is these largest importance scores which determine the success of DALex. The uniform distribution and random shuffle, on the other hand, lower kurtosis, and perform worse compared to the normal distribution. However, increasing the standard deviation recovers the success rate of the normal distribution.