The Impact of Hyperselection on Lexicase Selection

Thomas Helmuth
Computer Science
Washington and Lee U.
Lexington, Virginia, USA
helmutht@wlu.edu

Nicholas Freitag McPhee
Div. of Science & Mathematics
U. of Minnesota, Morris
Morris, Minnesota, USA
mcphee@morris.umn.edu

Lee Spector
Cognitive Science
Hampshire College
Amherst, Massacusetts, USA
lspector@hampshire.edu

ABSTRACT

Lexicase selection is a parent selection method that has been shown to improve the problem solving power of genetic programming over a range of problems. Previous work has shown that it can also produce *hyperselection* events, in which a single individual is selected many more times than other individuals. Here we investigate the role that hyperselection plays in the problem-solving performance of lexicase selection. We run genetic programming on a set of program synthesis benchmark problems using lexicase and tournament selection, confirming that hyperselection occurs significantly more often and more drastically with lexicase selection, which also performs significantly better. We then show results from an experiment indicating that hyperselection is not integral to the problem-solving performance or diversity maintenance observed when using lexicase selection. We conclude that the power of lexicase selection stems from the collection of individuals that it selects, not from the unusual frequencies with which it sometimes selects them.

CCS Concepts

•**Computing methodologies** → **Genetic programming;** *Heuristic function construction;*

Keywords

lexicase selection; tournament selection; hyperselection; program synthesis

1. INTRODUCTION

Evolutionary computation systems direct search through the selection of individuals, with the goal of refining and recombining promising programs to produce better ones. Selection can be applied at various stages of the evolutionary algorithm, but in genetic programming it is generally applied only when choosing parents. That is, a *parent selection* method chooses the individuals to vary to generate offspring, typically through recombination and mutation. Different parent selection methods will select parents with different frequencies, directing the search in different ways.

GECCO '16, July 20 - 24, 2016, Denver, CO, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4206-3/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2908812.2908851

For problems involving multiple test cases, most parent selection methods select on the basis of aggregate performance across all test cases. For example, when using tournament selection, one first calculates the fitness of each individual as the total or average error across all test cases. Then, one conducts tournaments in which the program with the best fitness wins and is selected as a parent.

Lexicase selection chooses parents based not on aggregate performance across all test cases, but instead on performance on individual test cases, considered one at a time in random order. Lexicase selection shares some characteristics with other behavior-based selection mechanisms that consider test cases individually instead of in aggregate [11, 12]. It differs from these other methods in the way in which it emphasizes different combinations of test cases in each selection. The lexicase selection algorithm is described in detail in Section 2.

In previous work, lexicase selection has been shown to significantly enhance the problem-solving power of genetic programming compared to standard tournament selection, and to tournament selection with implicit fitness sharing [13], where the aggregate measure is weighted by case difficulty. For example, it has improved performance when finding terms in finite algebras, designing digital multipliers, producing programs that replicate the `wc` command, and performing symbolic regression of the factorial function [8]. In other work it was shown to significantly enhance the ability of genetic programming to solve the problems in a general program synthesis benchmark suite [7, 5].

Subsequent investigations have focussed on why lexicase selection often enhances problem-solving performance, with an eye toward refinement of the method or the development of more powerful methods. For example, studies of population diversity under lexicase selection have shown that it tends to produce and maintain significantly more diverse populations than those produced by tournament selection with or without implicit fitness sharing [6]. Another study showed that it tends to produce and propagate specialist individuals that do well on some cases but poorly on others [5].

The study described in the present paper stems from an observation that when lexicase selection is used, single individuals are sometimes exploited aggressively in the production of the next generation, being chosen in an exceptionally large number of parent selection events. This was discovered in part through studies in which graph databases were used to visualize the ancestries of evolved solution programs [14].

We use the term *hyperselection* to describe events in which a single individual is selected in a large percent of the parent selections in a generation. Anecdotally, we have observed cases in which a single individual is selected in over 90% of the parent selection events in a single generation, and have often observed a parent making up 5% or 10% of the selections. In some cases, individuals that are hyperselected have relatively poor total error, but they nonetheless give rise to progeny that evolve to solve the target problem. These observations strike us as remarkable, in part because the level of hyperselection that we see with lexicase selection is not even possible with many of the more commonly used selection methods, such as tournament selection.

Our observations of hyperselection events when using lexicase selection led us to hypothesize that hyperselection may be partly responsible for the problem-solving power of lexicase selection, instead of simply being a side effect. To test this hypothesis, we introduce a new *sampled lexicase-tournament selection* method, which takes individuals from a pool produced by lexicase selection but with frequencies similar to those of tournament selection. If hyperselection is responsible for the problem-solving performance of lexicase selection, then sampled lexicase-tournament selection should not perform as well as lexicase selection since it has similar hyperselection characteristics to tournament selection. However, our results show that sampled lexicase-tournament selection does indeed perform comparably to lexicase selection, leading us to conclude that hyperselection does not play a large role in lexicase selection's success.

In the following section, we describe the lexicase selection algorithm. In Section 3, we formally define hyperselection and discuss the selection frequency profile of standard tournament selection, in which hyperselection should be rare. Our results in Section 4 confirm that high levels of hyperselection are only present when using lexicase selection, and not with tournament selection. We then investigate the role that hyperselection plays in the problem-solving performance of lexicase selection. Section 5 presents our experiments with sampled lexicase-tournament selection. We conclude that the power of lexicase selection stems from the collection of individuals that it tends to select, not from the unusual frequencies with which it sometimes selects them.

2. LEXICASE SELECTION

Lexicase selection is a parent selection method designed for population-based stochastic search algorithms such as genetic programming [8, 15]. It can be used any time that potential parents are assessed with respect to multiple test cases. Previous work has shown that lexicase selection can effectively increase performance while also increasing behavioral diversity on a variety of genetic programming problems [7, 8, 12, 6].

We outline the lexicase selection algorithm in Figure 1. The key idea is that the test cases are randomly shuffled for *each* selection event, and then considered in that order. For each test case, the only individuals that are kept are those whose error is minimal *among those still under consideration.* This filtering is then repeated for each test case until there is only a single individual left or until all the test cases have been considered, in which case an individual is randomly chosen from the remaining pool.

```
To select one parent program for use in a genetic operation:

  1. Initialize:
       (a) Set candidates to be the entire population.
       (b) Set cases to be a list of all of the test cases in the
           training set in random order.

  2. Loop:
       (a) Set best to be the best performance of any individual
           currently in candidates for the first case in cases.
       (b) Set candidates to be the subset of the current
           candidates that have exactly best performance on the
           first case in cases.
       (c) If candidates contains just a single individual then
           return it.
       (d) If cases contains just a single test case then return a
           randomly selected individual from candidates.
       (e) Otherwise remove the first case from cases and go to
           Loop.
```

Figure 1: The lexicase selection algorithm.

Because the ordering of the test cases is randomized for each selection, the priority of test cases is different for each selection. Assuming the population size is considerably larger than the number of test cases, this ensures that each test case is most important (first in the order) for several of the selections, and very important (in the early part of the order) for many more selections. Conversely, a test case that comes near the end of the ordering will only impact a selection if a subset of the population performs equally well on every test case before it. Since lexicase selection often ignores test cases near the end of the ordering, it sometimes selects *specialist* individuals that perform poorly on some test cases but perform better than many individuals on one or more other cases.

Since lexicase emphasizes different test cases in each selection event, it does not base selection on a single measure of performance. Methods like tournament selection that use a single fitness value tend to select *generalist* individuals that have good average performance across all test cases, but may not perform particularly well on any. This difference allows lexicase selection to maintain higher population diversity by emphasizing a different part of the problem for each selection, where tournament selection loses diversity by focusing on individuals with good average performance. This difference in diversity maintenance has been shown empirically on a number of program synthesis benchmark problems, where lexicase selection substantially outperforms standard tournament selection with or without implicit fitness sharing [7, 5] and typically maintains higher levels of diversity [6].

Other parent selection techniques have been invented to achieve similar goals to lexicase selection [3, 9, 10, 4]. Because the aim of this paper is only to investigate the relationship between hyperselection and lexicase selection, and not to compare lexicase selection to other selection methods, further discussion of alternate selection methods is outside the scope of this paper.

3. HYPERSELECTION

Observations of runs using lexicase selection have revealed non-trivial levels of *hyperselection*, where individuals are selected to produce large portions of the children in the next generation.[1] We say an individual is *hyperselected at the X% level* if that individual receives at least X% of the parent selections in a single generation. For example, an individual that receives at least 170 selections out of 1700 total in a generation is considered hyperselected at the 10% level (as well as any level below 10%). Examining hyperselection events can help us characterize how often single individuals receive a large percent of the selections in their generations; here, we will look at hyperselection events at the 1%, 5%, and 10% levels. The 1% level represents individuals with higher than typical impact on the next generation, where the 5% and 10% levels identify individuals that significantly influence the parent pool in their generations.

With tournament selection, the number of times an individual can be selected is limited by the number of tournaments in which it participates, a number which only depends on the population and tournament sizes, not on the properties of the individual or other individuals in the population. For example, if the best member of the population participates in 1.27% of the tournaments for a given generation, it will be selected 1.27% of the time that generation, but no more. Since the expected number of tournaments in which each individual participates is constant for a particular population size P and tournament size t, the probability of an individual being selected by tournament selection is entirely determined by its rank in the population. In particular, the probability of selecting an individual with rank $i \in [1, P]$, with $i = 1$ being the best rank, is

$$p(i) = \frac{(P - i + 1)^t - (P - i)^t}{P^t} \tag{1}$$

assuming no two individuals have the same fitness [1, 2]; we plot this probability mass function in Figure 2. Equation 1 does not hold exactly when there are ties in the rankings, but is approximately correct unless there are many tied individuals.

From Equation 1, we see that when we use population size 1000 and tournament size 7, the best few individuals will be selected approximately 0.7% of the time each. This also follows from the fact that every individual will participate in approximately 0.7% of the tournaments, and the best individual will win each tournament in which it participates. With tournament selection it would therefore be unlikely to hyperselect many individuals at the 1% level in a generation, and extremely unlikely for any individuals to be hyperselected at the 5% or 10% level. Empirical simulations show that with population size 1000 and no ties, an average of just over 2 individuals are hyperselected at the 1% level each generation, and no individuals are hyperselected at the 5% or 10% level. These limited number of hyperselections will be even lower if there are ties, especially amongst the best individuals, since those selections will then be shared across the tied individuals.

On the other hand, we have observed numerous examples of lexicase selection rewarding an interesting individual with over 90% of the selections in a generation—over 100 times as often as the best individual is expected to be selected under tournament selection. In general, if an individual's error vector Pareto dominates a substantial percentage of the other error vectors in the population, then we would expect lexicase selection to select that individual quite often. In an extreme case, if a population has an individual that Pareto dominates every other member of the population, then that individual would receive *all* of the selections in that generation. We therefore expect lexicase selection to produce non-zero numbers of hyperselections at the 5% and 10% levels, though without empirical data it is unclear how common these are or what impact they have on evolution.

[1]Much of the following text is adapted from the lead author's Ph.D. dissertation [5], but not previously published elsewhere.

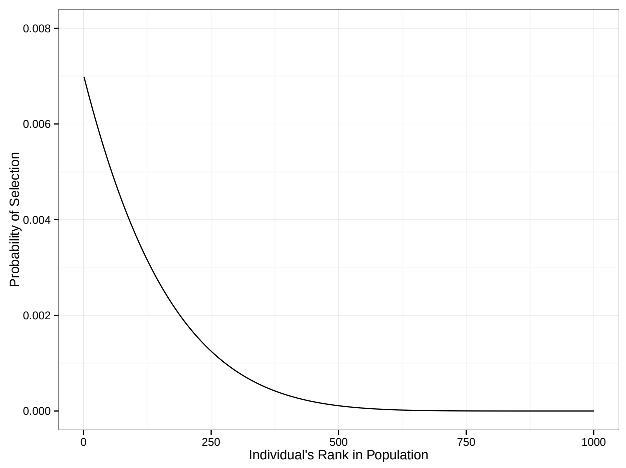

Figure 2: Probability mass function of selecting individual with rank i out of a population of 1000 individuals using tournament selection with tournament size 7, assuming no two individuals have the same rank. This plots Equation 1 with $P = 1000$ and $t = 7$.

4. MEASURING HYPERSELECTION

To empirically measure hyperselection, we gathered data from genetic programming runs using lexicase selection and tournament selection with size 7 tournaments. We tested each selection method on nine problems, exhibiting a range of problem requirements and difficulties, from a recent general program synthesis benchmark suite [7]; see Table 1 for details. These problems, taken from introductory computer science homework sets, mimic the types of programming expected of humans writing software. On each problem, we use a training data set during evolution and an unseen test set to test generalization, following the methods described in the benchmark suite [7]. A program must pass both the training and test sets to be considered a solution.

For our experiments we used PushGP [18, 17], a stack-based genetic programming system.[2] PushGP supports a variety of control structures and multiple data types, making it a good choice for program synthesis tasks. The PushGP parameters used in these experiments are the same as those reported in [5]; Table 2 presents the most relevant parameters. Note that these runs use linear genomes that are translated into Push programs before execution, as described in

[2]Lexicase selection has also been shown to be effective in tree-based genetic programming [8, 12].

Table 1: Problems used in our experiments. The instruction set used for each problem includes all instructions available in PushGP that make use of the data types listed in the third column. Any problem with "print" listed as a data type requires the program to print the result to standard output. All problems also used instructions making use of Push's exec stack, which allow for a variety of control flow structures such as conditional execution, recursion, and iteration. See the benchmark suite specifications for more details [7].

Problem	Description	Data Types
Count Odds	Given a vector of integers, return the number of integers that are odd, without use of specific `even` or `odd` instructions.	integer, boolean, vector of integers
Double Letters	Given a string, print the string, doubling every letter character, and tripling every exclamation point. All other non-alphabetic and non-exclamation characters should be printed a single time each.	integer, boolean, char, string, print
Mirror Image	Given two vectors of integers, return `true` if one vector is the reverse of the other, and `false` otherwise.	integer, boolean, vector of integers
Negative To Zero	Given a vector of integers, return the vector where all negative integers have been replaced by 0.	integer, boolean, vector of integers
Replace Space with Newline	Given a string input, print the string, replacing spaces with newlines. Also, return the integer count of the non-whitespace characters.	integer, boolean, char, string, print
String Lengths Backwards	Given a vector of strings, print the length of each string in the vector starting with the last and ending with the first, each on a separate line.	integer, boolean, char, string, vector of strings, print
Syllables	Given a string containing symbols, spaces, digits, and lowercase letters, count the number of occurrences of vowels (a, e, i, o, u, y) in the string and print that number as X in `The number of syllables is X`.	integer, boolean, char, string, print
Vector Average	Given a vector of floats, return the average of those floats. Results are rounded to 4 decimal places.	integer, float, vector of floats
X-Word Lines	Given an integer X and a string that can contain spaces and newlines, print the string with exactly X words per line. The last line may have fewer than X words.	integer, boolean, char, string, print

Table 2: PushGP parameters

Parameter	Value
runs per problem/parameter combination	100
population size	1000
maximum generations	300
Genetic Operator	**Prob**
alternation	0.2
uniform mutation	0.2
uniform close mutation	0.1
alternation followed by uniform mutation	0.5

[5]. We use a variety of genetic operators on these linear genomes, including alternation, a linear crossover operator modeled after ULTRA [16]; uniform mutation, which replaces each instruction with some probability; uniform close mutation, which probabilistically adds or removes parentheses; and alternation followed by uniform mutation.

To track hyperselection, we calculated the average number of hyperselected individuals at the 1%, 5%, and 10% levels per generation, which we present in Table 3 (ignore SLT for now). On all problems except one, lexicase selection hyperselects 5 or more individuals per generation on average at the 1% level; tournament selection averages less than one per generation on all problems, though always greater than 0.2. Unsurprisingly, tournament selection never hyperselected a single individual at the 5% or 10% levels. On 7 of the 9 problems, lexicase selection hyperselected one individual at the 5% level at least once every every 10 generations on average, with some higher. On those same problems, lexicase selection hyperselected one individual at the 10% level at least once every 25 generations on average.

The Vector Average and Count Odds problems showed much lower levels of hyperselection with lexicase selection, especially at the 5% and 10% levels, indicating that selecting a single individual to parent many of the children in a single generation was rarer on those problems. These two problems were also among the least-solved problems in this subset of the benchmark problems (see Table 5); one hypothesis is that most of the genetic programming runs had trouble getting any traction on these problems, leading to more homogeneous populations and fewer hyperselection events than on other problems.

Considering that tournament selection conforms to the probability of selection given in Equation 1 regardless of problem, it is at first surprising that its hyperselections at the 1% level vary as much as they do across problems. This difference is likely explained by how often tied individuals appear near the top of the rankings for different problems, since Equation 1 does not strictly hold in the presence of ties. Intuitively, if many individuals tie for the best rank, they will each win fewer tournaments than a single best individual would, since ties in tournaments are broken randomly. Therefore, lower hyperselection for tournament selection on a problem likely indicates that ties happened more often on those problems, which we have observed anecdotally.

The results in Table 3 clearly show that lexicase selection gives more of its parent selections to single individuals than tournament selection, both at low levels (1%) and higher levels (5% and 10%) of hyperselection. Thus lexicase selection more often concentrates its selection pressure on single individuals or small groups of individuals than tournament selection, increasing its exploitation of the individuals it selects most often. This data does not, however, indicate whether

Table 3: Average number of hyperselected individuals per generation at the 1%, 5%, and 10% levels for lexicase selection, tournament selection and SLT selection.

	Lexicase			Tournament			SLT		
Problem	**1%**	**5%**	**10%**	**1%**	**5%**	**10%**	**1%**	**5%**	**10%**
Count Odds	0.49	0.02	0.00	0.70	0.00	0.00	1.54	0.00	0.00
Double Letters	12.28	0.29	0.09	0.36	0.00	0.00	1.54	0.00	0.00
Mirror Image	9.72	0.23	0.05	0.22	0.00	0.00	1.55	0.00	0.00
Negative To Zero	8.21	0.39	0.16	0.43	0.00	0.00	1.55	0.00	0.00
Replace Space with Newline	13.39	0.38	0.11	0.28	0.00	0.00	1.56	0.00	0.00
String Lengths Backwards	6.21	0.54	0.25	0.42	0.00	0.00	1.53	0.00	0.00
Syllables	5.74	0.13	0.05	0.38	0.00	0.00	1.55	0.00	0.00
Vector Average	6.99	0.02	0.01	0.91	0.00	0.00	1.56	0.00	0.00
X-Word Lines	5.31	0.13	0.04	0.36	0.00	0.00	1.55	0.00	0.00

Table 4: Average number of hyperselected individuals per generation at the 1%, 5%, and 10% levels for lexicase selection on five of the nine benchmark problems, broken out into successful and unsuccessful runs.

	Successful			Unsuccessful		
Problem	**1%**	**5%**	**10%**	**1%**	**5%**	**10%**
Count Odds	0.84	0.07	0.02	0.47	0.01	0.00
Double Letters	11.88	0.72	0.28	12.29	0.28	0.08
Mirror Image	9.32	0.27	0.06	10.49	0.15	0.04
Vector Average	8.35	0.05	0.02	6.78	0.02	0.00
X-Word Lines	5.43	0.47	0.22	5.29	0.09	0.02

lexicase selection is hyperselecting the same individuals that tournament selection ranks highest (those with best total error), or if it actually selects individuals that would receive few or no selections with tournament selection.

Previous work and the results in Table 5 show that genetic programming with lexicase selection significantly outperforms genetic programming with tournament selection across many benchmark problems, including most of those in Table 3. The fact that lexicase selection often concentrates selection pressure onto small numbers of individuals through hyperselection raises the question of whether there is a relationship between the rates of hyperselection in the lexicase runs and whether those runs are successful or not.

Table 4 shows the average number of hyperselected individuals for the lexicase runs of five of the nine benchmark problems, separated into the successful and unsuccessful runs.[3] While there are not major differences in the numbers at the 1% level, the average number of hyperselected individuals at the 5% and 10% levels are often several times higher in the successful runs than in the unsuccessful runs. In the X-Word Lines problem, for example, the average number of hyperselections at the 5% level for the successful runs was 0.47, over five times the rate for the unsuccessful runs (0.09), and the 10% rate (0.22) was over 10 times the rate for the unsuccessful runs (0.02).

Can we attribute lexicase selection's success to its ability to concentrate selection in hyperselection events? Or, is it more important that lexicase selects different individuals than tournament selection, in some sense the "right" individuals to drive evolution toward a solution? We investigate these questions in the following section.

[3]We do not have success/failure split data for the other 4 problems, since they were conducted before we considered splitting the data. Since this result is not a major contribution of this work, we did not rerun this experiment.

5. HYPERSELECTION AND LEXICASE PERFORMANCE

We have shown that lexicase selection often ends up selecting the same individual many times in one generation, and does so much more often than is mathematically possible with tournament selection. This raises the question of whether the hyperselections observed in lexicase selection runs are important in driving evolution toward solutions. The alternative is that the individuals that lexicase selects the most often are simply different from those selected most often by tournament selection, in particular specialists with high total error but low error on a subset of cases. In this section we test the hypothesis that the hyperselection events we observed in runs using lexicase selection are integral to its success, and that without these extreme exploitative events, lexicase selection would perform significantly worse than it does with them.

To test this hypothesis, we designed a new parent selection algorithm that focuses its selections on the same individuals that lexicase does, but has hyperselection characteristics much closer to those of tournament selection. The new algorithm, *sampled lexicase-tournament selection* (SLT), starts by sampling the population, which only happens once per generation before selecting any parents. We sample k individuals from the population by running the lexicase selection algorithm and tracking how often each individual is selected. In this work we set $k = 2P$, where P is the population size (1000 in our runs), guaranteeing at least as many samples as the number of parent selections in that generation.[4] We

[4]We expect about 1700 parent selections per generation.

then use the number of samples each individual received to rank the population from best (most samples) to worst (least samples). Next, every time we need to select a parent, we conduct a tournament, where the winner of the tournament is based on the lexicase-sampled ranking instead of total error. In this experiment we used size 7 tournaments, just like we did with tournament selection in our experiments.

SLT can be seen as a variation of tournament selection in which fitness is based on lexicase sampling instead of total error. SLT gives the highest probabilities of selection to those individuals that lexicase would select the most often in the population. Since it uses tournaments for selection, however, its probability of selecting the individual ranked i in the lexicase-sampled ranking will be same as in tournament selection, as given in Equation 1. We would therefore expect the hyperselection characteristics of SLT to mirror those of tournament selection, and differ only when the two behave differently with respect to tied individuals in the rankings, especially ties amongst the best individuals. To be clear, we developed SLT not because we believe it could be a useful parent selection method, but instead because it will allow us to test the hypothesis that hyperselection events are critical to lexicase selection's improved performance compared to tournament selection.

We conducted 100 runs of PushGP using SLT on the same 9 benchmark problems; the hyperselection results for SLT are also included in Table 3. The first thing to note is that SLT has higher hyperselection at the 1% level than tournament selection. Theoretically, we would expect SLT to behave similarly to tournament selection if neither had ties in rank within the population. We believe the differences we see here are a product of ties in total error when using tournament selection, especially near the top of the rankings. The relative consistency of SLT's 1%-level hyperselection likely comes from the fact that it rarely had large numbers of tied individuals near the top of the rankings.

In these runs, SLT usually had substantially lower hyperselection at the 1% level than lexicase selection, and always lower at the 5% and 10% levels, on which it never had a non-zero result. Since SLT was designed to have hyperselection characteristics similar to tournament selection, it is unsurprising that it received no hyperselections at the upper levels. This means that SLT succeeds in our goal of creating a lexicase-based selection mechanism that never puts as much as 5% of the parent selections in a generation on a single individual. This contrasts with lexicase selection, which often selects single individuals to make large numbers of children for the next generation.

Since we have shown that SLT has similar hyperselection characteristics to tournament selection, let us now examine its performance results in these runs, which we present in Table 5. Across these 9 problems, SLT shows very similar performance to lexicase selection, and better performance than tournament selection on every problem. Both SLT and lexicase found at least one solution on each of the 9 problems, where tournament selection only found solutions to 6 of the problems. Comparing these methods using a chi-square test with the Holm correction, SLT never has a significantly different success rate compared to lexicase selection. SLT is significantly better than tournament selection on 5 of the 7 problems on which lexicase is, and is additionally significantly better on one other problem. Though the difference is never significant, SLT seems to slightly outperform lexicase selection on the easier problems where both find more solutions, and lexicase selection slightly outperforms SLT on the more difficult problems where both find fewer solutions.

Table 5: Number of successful runs out of 100 for each parent selection method on each problem. For each problem, underline indicates significant improvement over tournament selection using a pairwise chi-squared test with Holm correction and 0.05 significance level. SLT and lexicase selection never have significantly different success rates.

Problem	Lex	Tourn	SLT
Count Odds	8	0	5
Double Letters	6	0	4
Mirror Image	78	46	84
Negative To Zero	45	10	53
Replace Space with Newline	51	8	61
String Lengths Backwards	66	7	79
Syllables	18	1	13
Vector Average	16	14	30
X-Word Lines	8	0	4

Previous work has shown lexicase selection's ability to maintain higher levels of population diversity than other methods [6], so we were interested to see if SLT also shares this characteristic, or if hyperselection when using lexicase raises (or even lowers) diversity. Figures 3, 4, and 5 show the error diversity—the number of unique error vectors in the population—over time for three representative problems. Each of these figures plots the median and quartiles for error diversity across 100 runs. Interestingly, the diversity plots for SLT are very similar to those of lexicase selection. Thus, even though the techniques produce significantly different hyperselection rates, their populations still maintain similar abilities to search widely.

It is perhaps somewhat surprising that SLT and lexicase selections lead to such similar error diversities, as we might expect lexicase selection's hyperselection events to cause major drops in population diversity. Figure 6 shows the error diversity over time for three specific runs of the Replace Space With Newline problem using lexicase selection, each of which has at least one substantial drop in diversity associated with a hyperselection event. In each of these specific cases lexicase selection is able to increase error diversity within a matter of generations, bringing it to approximately the median diversity levels seen in Figure 4. The consistently high levels of error diversity when using lexicase (e.g., Figures 3–5) suggest that if hyperselection events lead to drops in diversity, lexicase selection is generally able to promptly restore diversity. This sort of diversity recovery is likely important to lexicase selection's success; otherwise, diversity cliffs caused by hyperselection events could lead to populations with little diversity, incapable of widely exploring the search space.

These results show that even though SLT has far fewer hyperselection events than lexicase selection, never selecting a single individual to parent more than 5% of the children in a generation, it nevertheless exhibits the problem-solving performance and diversity levels shown by lexicase selection. These results give strong evidence against the hypothesis that lexicase selection's increased exploitation of hyperse-

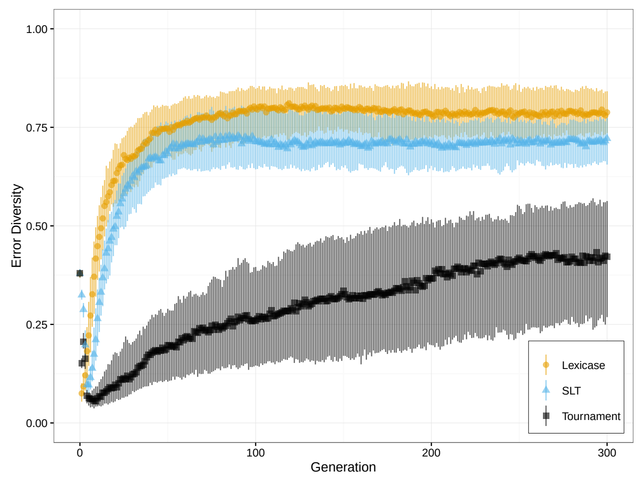

Figure 3: Error diversity for the Double Letters problem.

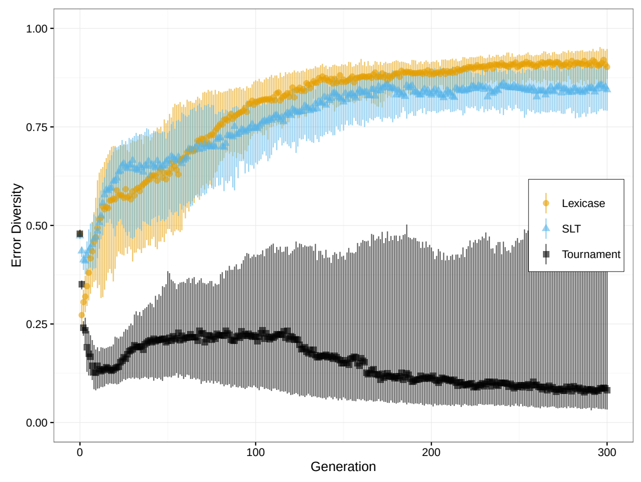

Figure 4: Error diversity for the Replace Space With Newline problem.

lected individuals is necessary for its ability to outperform tournament selection with and without implicit fitness sharing. Instead, this suggests that it is more important *which* individuals lexicase selection selects most often, which it has in common with SLT but not tournament selection.

While SLT achieved similar performance to lexicase selection in this experiment, we do not see indications that it would make a better parent selection mechanism. Notably, it will be slower than lexicase selection in practice, since it performs both lexicase sampling and then tournaments for selection. Even so, it may merit further examination on other types of problems to see if it behaves differently in other settings.

6. CONCLUSIONS

Because lexicase selection can significantly improve the ability of genetic programming to solve hard problems, it would be useful to better understand the reasons that it sometimes works so well. Previous observations of the unusual phenomenon of hyperselection under lexicase selection, in which a single individual is selected as a parent in a very large number of selection events, suggested that there might be an interesting connection between hyperselection and the problem-solving power of lexicase selection.

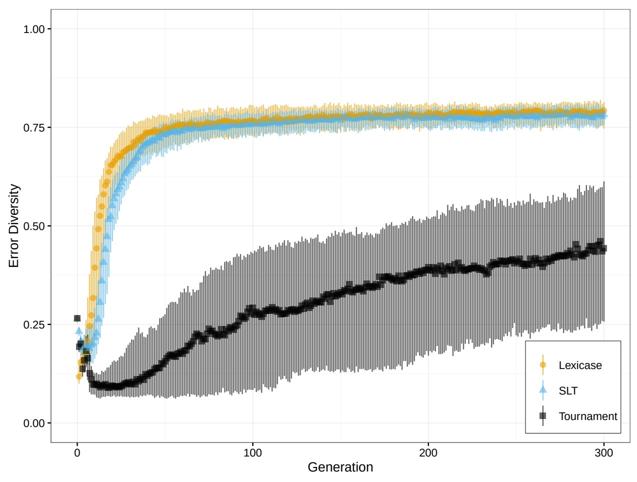

Figure 5: Error diversity for the Syllables problem.

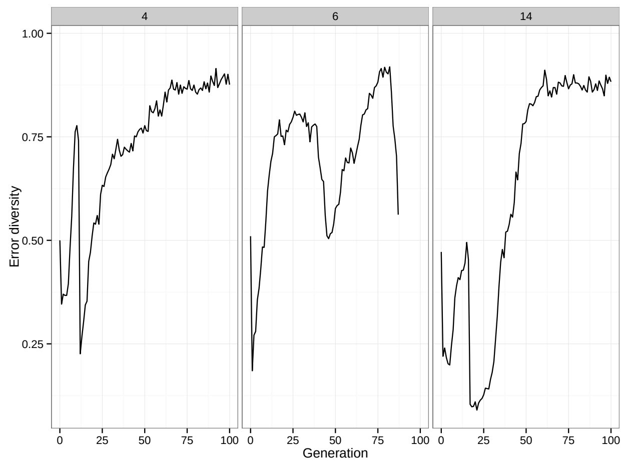

Figure 6: Error diversity for three runs of Replace Space With Newline using lexicase selection, showing the rapid return of diversity after hyperselection events that rapidly decrease diversity.

The experiments presented here show that the problem-solving power of lexicase selection is maintained even if it is altered to prevent hyperselection, using selection frequencies like those of tournament selection, as exhibited by the new *sampled lexicase-tournament selection* algorithm. This result gives strong evidence that the power of lexicase selection must stem from the collection of individuals that it tends to select, not from the unusual frequencies with which it sometimes selects them. Previous work had already shown that lexicase selection often selects specialist individuals that do well on some cases but poorly on others [6, 5], and our results here suggest that it may be the exploitation of these specialists that is primarily responsible for the unusual problem-solving power of lexicase selection.

One avenue for future research is to investigate the notion of a "specialist" more thoroughly, to determine if, perhaps, some kinds of specialists may be more valuable than others.

Our anecdotal exploration of quick diversity increases following major drops in diversity when using lexicase selection suggests another area for future work. A more thorough exploration of diversity recovery could lend understanding to the causes of lexicase selection's ability to maintain high levels of population diversity.

While this work focuses on the effects of hyperselection on lexicase selection, hyperselection may have a role in other selection methods not explored here. While other tournament-based methods (such as implicit fitness sharing [13]) should roughly mirror the hyperselection characteristics of tournament selection, other methods such as fitness-proportionate selection or Pareto-based techniques may have different hyperselection characteristics that could be worth exploring.

7. ACKNOWLEDGMENTS

Thanks to the members of the Hampshire College Computational Intelligence Lab for discussions that helped shape this work and to Josiah Erikson for systems support. This material is based upon work supported by the National Science Foundation under Grants No. 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.

8. REFERENCES

[1] T. Bäck. Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 57–62 vol.1, Jun 1994.

[2] T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 9–16, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[3] J. E. Fieldsend and A. Moraglio. Strength through diversity: Disaggregation and multi-objectivisation approaches for genetic programming. In *GECCO '15: Proceedings of the 2015 conference on Genetic and evolutionary computation*. ACM, 2015.

[4] E. Galván-López, B. Cody-Kenny, L. Trujillo, and A. Kattan. Using semantics in the selection mechanism in genetic programming: A simple method for promoting semantic diversity. In L. G. de la Fraga, editor, *2013 IEEE Conference on Evolutionary Computation*, volume 1, pages 2972–2979, Cancun, Mexico, June 20-23 2013.

[5] T. Helmuth. *General Program Synthesis from Examples Using Genetic Programming with Parent Selection Based on Random Lexicographic Orderings of Test Cases*. Ph.D. dissertation, 2015.

[6] T. Helmuth, N. F. McPhee, and L. Spector. Lexicase selection for program synthesis: a diversity analysis. In *Genetic Programming Theory and Practice XIII*, Genetic and Evolutionary Computation. Springer.

[7] T. Helmuth and L. Spector. General program synthesis benchmark suite. In *GECCO '15: Proceedings of the 2015 Conference on Genetic and Evolutionary Computation*, July 2015.

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

[9] K. Krawiec and P. Lichocki. Using co-solvability to model and exploit synergetic effects in evolution. In *PPSN 2010 11th International Conference on Parallel Problem Solving From Nature*, volume 6239 of *Lecture Notes in Computer Science*, pages 492–501, Krakow, Poland, 11-15 Sept. 2010. Springer.

[10] K. Krawiec and P. Liskowski. Automatic derivation of search objectives for test-based genetic programming. In *18th European Conference on Genetic Programming*, volume 9025 of *LNCS*, pages 53–65, Copenhagen, 8-10 Apr. 2015. Springer.

[11] K. Krawiec, J. Swan, and U.-M. O'Reilly. Behavioral program synthesis: Insights and prospects. In *Genetic Programming Theory and Practice XIII*, Genetic and Evolutionary Computation. Springer, 2015.

[12] P. Liskowski, K. Krawiec, T. Helmuth, and L. Spector. Comparison of semantic-aware selection methods in genetic programming. In *GECCO 2015 workshop on Semantic Methods in Genetic Programming*. ACM, 2015.

[13] R. I. McKay. Fitness sharing in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 435–442, Las Vegas, Nevada, USA, 10-12 July 2000. Morgan Kaufmann.

[14] N. F. McPhee, D. Donatucci, and T. Helmuth. Using graph databases to explore the dynamics of genetic programming runs. In *Genetic Programming Theory and Practice XIII*, Genetic and Evolutionary Computation. Springer.

[15] L. Spector. Assessment of problem modality by differential performance of lexicase selection in genetic programming: A preliminary report. In *1st workshop on Understanding Problems (GECCO-UP)*, pages 401–408, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM.

[16] L. Spector and T. Helmuth. Uniform linear transformation with repair and alternation in genetic programming. In *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation, chapter 8, pages 137–153. Springer, Ann Arbor, USA, 9-11 May 2013.

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

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