Theoretical Analysis of Lexicase Selection in Multi-objective Optimization

Thomas Jansen and Christine Zarges(✉)

Department of Computer Science, Aberystwyth University,
Aberystwyth SY23 3DB, UK
{t.jansen,c.zarges}@aber.ac.uk

Abstract. Lexicase selection is a parent selection mechanism originally introduced for genetic programming that has also been considered in the context of multi-objective optimization. This is the first theoretical runtime analysis of lexicase selection showing results for the bi-objective leading ones trailing zeroes benchmark problem. The lexicase selection operator is embedded into a simple hillclimbing algorithm and compared with different selection operators from the literature that are based on the classical dominance relationship. Strengths and weaknesses of the operators are demonstrated providing insights into their working principles. Results of experiments accompany the theoretical findings and point towards interesting questions for future research.

Keywords: Runtime analysis · Multi-objective optimisation
Selection operators

1 Introduction

The choice of appropriate selection operators is a crucial step in the design of evolutionary algorithms. Selection occurs twice in the typical evolutionary cycle—as selection for reproduction when selecting search points as parents and as selection for survival when deciding which search points will form the next population. In principle, the same mechanisms can be used for both scenarios and thus, it makes sense to consider common operators for both settings. Classical examples for selection operators include uniform, fitness-proportional, tournament and truncation selection (see [13] for an overview). A more recent proposal in the context of genetic programming is lexicase selection [22], where fitness evaluation is based on a number of test cases. However, it has been noted that lexicase selection lends itself naturally to multi-objective optimisation [15].

We consider lexicase selection in the context of a pseudo-Boolean bi-objective optimisation problem and compare it with three other multi-objective selection mechanisms based on the classical dominance relationship. Lexicase selection has a performance that is comparable to the best of three simple dominance-based selection mechanisms and clearly outperforms two of them.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 153–164, 2018.
https://doi.org/10.1007/978-3-319-99259-4_13

We discuss motivation and background of this study in more detail in the next section. We introduce the concrete algorithms and problems we study in a more formal way in Sect. 3. Section 4 is devoted to the theoretical analysis and contains our main results. We present results of empirical analysis in Sect. 5 that confirm the theoretical results and lead to open questions by considering a modified version of the algorithm. We summarise our findings and discuss directions for possible future research in Sect. 6.

2 Background

Our main goal is to perform a theoretical analysis of lexicase selection in the context of discrete multi-objective optimization. Lexicase selection is a parent selection mechanism that was introduced by Spector [22] to improve the performance of genetic programming in situations where fitness evaluation is based on (potentially a large number of) test cases. It is designed in a way that allows it to be used in place of tournament selection (or any other parent selection mechanism) without any other changes to the overall genetic programming algorithm. It has since been used in applications [9,11] and analysed in some detail. This includes comparing its performance to other selection mechanisms [7,10,12,18], studying its impact on population diversity [6,8], and considering its effects and performance in detail for a specific problem [19] (see also [5]). Since it can be less effective for problems in the continuous domain a variant called ε-lexicase selection for such continuous-valued problems has been introduced [15] and also analysed [14]. To the best of our knowledge the work by La Cava et al. [14] is the first theoretical analysis of a lexicase selection variant and it applies to the special version for continuous domains. We present the first theoretical analysis of the original lexicase selection method for discrete-valued problem and its impact on the expected optimization time.

Despite its origin in genetic programming lexicase selection is a general parent selection mechanism that can be employed in any kind of search algorithm that performs a step that is essentially parent selection. This includes for example evolutionary algorithms [1] and artificial immune systems [23]. We consider a simple hillclimber and compare lexicase selection with other selection mechanisms that have been used in this context. We will discuss details of all algorithms and selection operators we consider in the next section.

We consider a well known benchmark problem for multi-objective optimisation, the leading ones trailing zeros problem (formally defined as LOTZ in the following). It was introduced by Laumanns et al. [16] and was the first problem to be used for a theoretical runtime analysis. This motivates its use in this study which is a much more specific first in theoretical runtime analysis. LOTZ has been analysed for a number of simple evolutionary algorithms for multi-objective optimization (e. g., see [3,17]) and has also motivated the introduction of other, similar benchmark problems (e. g., see [4,17]). We will formally introduce the problem in the next section.

3 Algorithms, Problems, and Definitions

Our analysis focuses on lexicase selection, a parent selection method that was introduced by Spector [22]. It aims at genetic programming where the fitness of an individual is defined by its performance on a number of test cases. We consider it here as a general parent selection method for multi-objective optimization and formulate it in this context. We consider a multi-objective function f with c components so that for all x we have $f(x) = (f_1(x), f_2(x), \ldots, f_c(x))$. Without loss of generality we assume that the optimisation goal is to maximise each of the c different components. For such a multi-objective function f we describe lexicase selection as a method that selects one out of a pool P of parents. It goes through the c different objectives in random order (selecting the order randomly for each new selection operation) and eliminates for the current objective all search points that are inferior to other search points in the parent pool P. When this reduces the number of remaining potential parents to one the remaining parent is returned as selected. If at the end there is more than one potential parent left one is selected uniformly at random. We give a formal description as Algorithm 1.

Algorithm 1. Lexicase Selection

1 **for** $i \in \{1, 2, \ldots, c\}$ *in random order* **do**
2 | Remove from P all x with $f_i(x) < \max_{y \in P} \{f_i(y)\}$.
3 | **if** $|P| = 1$ **then**
4 | | return remaining $x \in P$ as selected
5 Select $x \in P$ uniformly at random and return x.

Evolutionary algorithms make use of selection in two different places in the evolutionary cycle: they use parent selection and selection for survival (e.g., see [13] for a general description). In some sense selection for survival is the opposite to parent selection since normally we select a worst individual to be removed from the population. We can do the same with lexicase selection simply by replacing the maximum by the minimum in line 2 of Algorithm 1. This follows in spirit the use of negative tournament selection in a well-known standard GP algorithm, called TinyGP, in the field guide to GP [21]. We refer to this version as negative lexicase selection and would use it whenever we need to decide which member of a population should be discarded. In the context of this work, we will not be needing this.

In multi-objective optimisation one usually considers Pareto optimality in some form as optimisation goal. For a multi-objective function $f\colon \{0,1\}^n \to \mathbb{R}^c$ with $f = (f_1, f_2, \ldots, f_c)$ we say that x weakly dominates y ($x \succeq y$) if $f_i(x) \geq f_i(y)$ for all $i \in \{1, 2, \ldots, c\}$. Note that we do not distinguish between dominance between x and y and $f(x)$ and $f(y)$ here. Clearly, the same terminology can be applied to both. We say that x dominates y ($x \succ y$) if $x \succeq y$ and there exists

some $i \in \{1, 2, \ldots, c\}$ with $f_i(x) > f_i(y)$. If neither $x \succeq y$ nor $y \succeq x$ we say that x and y are non-comparable. We say that $x \in \{0,1\}^n$ is non-dominated if there is no $y \in \{0,1\}^n$ such that $y \succ x$. The Pareto set is the set of all non-dominated search points in $\{0,1\}^n$. Its image under f is called the Pareto front.

When using an optimization heuristic for a multi-objective function one is usually interested in not only finding some solution that belongs to the Pareto set but a set of solutions such that the image of this set equals (or approximates) the Pareto front.

We analyse two different points of time. Let $x_1, x_2, \ldots$ be the sequence of search points that a multi-objective heuristic optimization algorithm generates. First, we consider the first point of time when a search point that belongs to the Pareto set is discovered (i.e., the smallest T_1 such that x_{T_1} belongs to the Pareto set and all x_i with $i < T_1$ do not belong to the Pareto set). Second, we consider the first point of time when the whole Pareto front is discovered (i.e., the smallest T_2 such that $\{f(x_1), f(x_2), \ldots, f(x_{T_2})\}$ is the Pareto front and for all $i < T_2$ we have that $\{f(x_1), f(x_2), \ldots, f(x_i)\}$ is not the Pareto front. Clearly, T_1 and T_2 are random variables if the optimization algorithm makes random choices. We will analyze their expectation in the following. Note that the definitions of T_1 and T_2 do not impose any requirements on the algorithm. It is not necessary that the algorithm knows that T_1 or T_2 have been reached. For T_2, it is not necessary that the algorithm keeps a representative for each point of the Pareto front somewhere. This allows us not to discuss the use of populations and archives and still analyse the performance of any multi-objective optimisation algorithm. If in an application it is desirable to output the Pareto front once it is found an archive that stores a representative for each non-dominated solution could easily be used.

Since this is a first theoretical run time analysis of lexicase selection it makes sense to start with an algorithm that is as simple as possible. In the context of evolutionary algorithms this is often the so-called $(1+1)$ EA. It is similar to local search because it has a population of size only 1 and creates one offspring in each round, selecting the better of parent and offspring for survival. Different from local search it uses standard bit mutations, a global mutation operator that flips each bit independently with probability $1/n$ (where n is the number of bits). While in expectation only 1 bit flips (the same number as for local search) any number of bits can flip so that a single mutation can reach any point in the search space with positive probability. While local search and the $(1+1)$ EA are often similar it is known that the global mutation can be the cause of very different behaviour [2]. In particular in the context of multi-objective optimization it is known that global mutations make the analysis considerably more difficult. This is the reason why usually results for SEMO, a simple evolutionary optimizer employing the same local steps as local search, are much easier to obtain than for GEMO, the same optimizer but with the same global mutations as used in the $(1+1)$ EA [17]. This is the reason why we consider random local search here. For the sake of completeness we give a formal description as Algorithm 2.

Algorithm 2. Random Local Search (RLS)

```
1 Select x ∈ {0,1} uniformly at random.
2 while termination criterion not met do
3 |   y := x
4 |   Select one bit in y uniformly at random and flip this bit.
5 |   Use a selection mechanism to determine which of {x, y} replaces x.
```

When considering RLS and analysing T_1 and T_2 it is important to note that we analyse the sequence of search points x that RLS generates. We do not take into account any search points y that do not replace x as new current search point.

Clearly, lexicase selection is not the only possibility to turn an evolutionary algorithm (or other heuristic optimiser like RLS) into an algorithm that can be used for multi-objective optimisation. Giel and Lehre [4] consider four different operators, three very simple and straightforward ones and one slightly more complicated on a different benchmark function. We adopt the three simple selection mechanisms to have a baseline for comparison here.

Definition 1. *The strong selection operator prefers a new search point y over x if y dominates x. The weak selection operator prefers a new search point y over x if y weakly dominates x. The weakest selection operator prefers a new search point y over x if y weakly dominates x or x and y are not comparable.*

We see that all three selection operators are different from lexicase selection and we will see how this influences the performance of RLS in the next section. We will refer to the four different algorithms as RLS with lexicase selection, RLS with strong selection, RLS with weak selection, and RLS with weakest selection in the following.

As mentioned earlier we use LOTZ as our benchmark function. The function LOTZ was introduced by Laumanns et al. [16] to facilitate theoretical analysis and it was important in the development of a theory-grounded understanding of evolutionary multi-objective optimization. This is the reason why we use the same function here. Our results will be specific to LOTZ, of course. We hope they will serve as a useful first step.

LOTZ is a bi-objective function and similar in structure to the well-known leading ones problem (e. g., see [13] for some background on this example problem). It asks to maximize the number of leading 1-bits and trailing 0-bits at the same time.

Definition 2. *The function* LOTZ$\colon \{0,1\}^n \to \mathbb{N}^2$ *is defined as* $\left(\sum_{i=1}^{n}\prod_{j=1}^{i} x[j], \sum_{i=1}^{n}\prod_{j=i}^{n}(1-x[j])\right)$.

It is well known and not difficult to see that the Pareto front of LOTZ equals $\{(0,n),(1,n-1),(2,n-2),\ldots,(n-1,1),(n,0)\}$ and that the Pareto set equals

$\{000\cdots000, 100\cdots000, 110\cdots000, \ldots, 111\cdots110, 111\cdots111\}$. The function is extreme in the sense that Pareto set and Pareto front have equal size. Each member of the Pareto front has only one representative in the search space.

4 Analysis

We analyse the performance of RLS with all four different selection operators (lexicase selection, strong selection, weak selection, weakest selection) on LOTZ. We consider both points of time that we defined in the previous section, the first point of time when a point on the Pareto front is found and also the first point of time when all points on the Pareto front have been found at least once.

Lemma 1. *The expected time until RLS with any of the four different selection operators (lexicase selection, strong selection, weak selection, weakest selection) finds a point on the Pareto front of* LOTZ *is* $O(n^2)$.

Proof. We consider the current search point x. Let $l \in \{0, 1, \ldots, n-2\}$ denote the number of leading 1-bits in x (i.e., $x[1] = x[2] = \cdots = x[l] = 1$ and $x[l+1] = 0$), let $r \in \{0, 1, \ldots, n-2\}$ denote the number of trailing 0-bits in x (i.e., $x[n] = x[n-1] = \cdots = x[n-r+1] = 0$ and $x[n-r] = 1$). Note that $l > n-2$ or $r > n-2$ imply that x is already at the Pareto front (where $r + l = n$ holds).

Let l' and r' denote the corresponding numbers in the new search point y. The probability to increase either l or r by 1 in a mutation equals $2/n$ because it suffices to flip either $x[l+1]$ or $x[n-r]$. In this case we have either $l' = l$ and $r' = r + 1$ or $l' = l + 1$ and $r' = r$.

In both cases each of the four selection mechanisms will prefer y over x because y is not inferior in any criterion and strictly better in one.

After at most n such steps we have $r + l = n$ and the Pareto front is reached. The expected waiting for each event equals $n/2$, thus the total expected waiting time is bounded by $n^2/2 = O(n^2)$. □

It is not hard to make Lemma 1 more precise and prove that the expected time is actually $\Theta(n^2)$, i.e., the bound $O(n^2)$ is asymptotically tight. However, since we are more interested in the time it takes to explore the Pareto front there is no point in making this more precise. The next results will prove that the time to find the first point of the Pareto front is insignificant in comparison to the time it takes to explore the whole Pareto front.

Theorem 1. *RLS with strong selection and RLS with weak selection never find more than a single point of the Pareto front of* LOTZ.

Proof. We consider the situation after the first point of the Pareto front of LOTZ has been found. By definition of the Pareto front, no other search point can dominate this search point. This implies that RLS with strong selection is stuck at this search point and will never go to a second search point on the Pareto front. By definition of LOTZ, no other search point can weakly dominate this search point because each point of the Pareto front of LOTZ has only

one representative point in the search point. This implies that RLS with weak selection is stuck at this search point and will never go to a second search point on the Pareto front. □

We see that RLS with strong and weak selection fail to optimise LOTZ in the sense that both are not able to find the whole Pareto front, even when given infinite time. The following result shows that RLS with lexicase selection is efficient in finding the complete Pareto front on LOTZ.

Theorem 2. *The expected time until RLS with lexicase selection finds each point of the Pareto front of* LOTZ *is* $\Theta(n^3)$.

Proof. We first prove an upper bound of $O(n^3)$ for the time to find the whole Pareto front of LOTZ. We start at a point of time when the current search point x is at the Pareto front. In the same way as in the proof of Lemma 1 let l denote the number of leading 1-bits in x and r denote the number of trailing 0-bits in x. Note that $l + r = n$ since x is at the Pareto front. Let l' and r' denote the corresponding numbers in the new search point y that is created as a mutant of x.

There are at least one and at most two positions such that either $l' = l - 1$ and $r' = r + 1$ or $l' = l + 1$ and $r' = r - 1$. For all other positions we have either $l' < l$ and $r' = r$ or $l' = l$ and $r' < r$. In this latter case the lexicase selection will prefer x over y.

We now consider the other case, the one that happens with probability at least $1/n$ and at most $2/n$. In both cases it depends on the order of selection criteria if x or y is preferred. In both cases there is one order where x is preferred over y and one order where y is preferred over x. Thus, we see that y replaces x with probability $1/2$ in this case and in total with probability at least $1/(2n)$ and at most $1/n$. Since the situation is completely symmetric x is replaced with a y that has a smaller or greater number of 1-bits with equal probability (except for the fringe cases $l = n$ and $r = n$). This implies that we can identify the changes of l (or, equivalently r) as an unbiased random work on $\{0, 1, \ldots, n\}$ where each step is taken with equal probability (except for the fringe cases $l = n$ and $r = n$) and the expected waiting time between two steps is between n and $2n$. It is well known (e.g., see [20, Chap. 6.5]) that the time to have visited each state (also know as cover time) is at most $2n \cdot 2n^2 = 4n^3$. This implies the upper bound of $O(n^3)$.

For the lower bound it suffices to notice that for the chain graph that corresponds to the unbiased random work on $\{0, 1, \ldots, n\}$ the cover time is $\Theta(n^2)$ and all the probabilities we considered were tight. Thus, the expected time is $\Theta(n^3)$. □

While strong and weak selection are both unsuccessful on LOTZ lexicase is not the only selection mechanism that is successful. The weakest selection mechanism has the same asymptotic run time as the next theorem shows.

Theorem 3. *The expected time until RLS with weakest selection finds each point of the Pareto front of* LOTZ *is* $\Theta(n^3)$.

Proof. If we consider two different points on the Pareto front they are always not comparable. Thus, in the situation where the current search point x is on the Pareto front and the new search point y is also on the Pareto front employing the weakest selection mechanism implies that y will replace x because it will prefer the new search point (see Definition 1). If the new search point is not the Pareto front it will be dominated by the old search point. Thus, RLS with the weakest selection performs the same kind of random walk on the Pareto front as RLS with lexicase selection and the same runtime bounds apply. □

It is worth noting that with lexicase selection a new search point at the Pareto front will only be accepted with probability 1/2 while with the weakest selection it will certainly be accepted. We therefore expect RLS with weakest selection to be twice as fast as RLS with lexicase selection. We remark that the same asymptotic runtime for LOTZ has been proven for the simple evolutionary multi-objective optimizer (SEMO) [17].

5 Experimental Supplements

We present the results of experiments to accompany our theoretical findings. We perform 100 runs for RLS (Algorithm 2) with lexicase selection (Algorithm 1) and with weakest selection (Definition 1) for $n = 10, 20, \ldots, 300$. We visualise the results using boxplots[1] in Figs. 1 and 2. For both algorithms we consider the time to reach the Pareto front (T_1) and the time until all points on the Pareto front have been sampled at least once (cover time T_2). Recall that both algorithms reach the Pareto front in time $O(n^2)$ (Lemma 1). Once the Pareto front is reached, only points on the Pareto front will be accepted and the random walk on the Pareto front has a cover time of $\Theta(n^3)$ for both algorithms (Theorems 2 and 3). The observed runtimes in our experiments nicely match these theoretical bounds as can be seen in Fig. 3 where we plot the mean observed number of function evaluations against a fitted polynomial based on our theoretical bounds. As expected, RLS with weakest selection is about twice as fast as RLS with lexicase selection when considering cover time. It should be noted that both algorithms exhibit a large variance when considering the cover time for the Pareto front. Thus, we observe a considerable number of outliers in our experiments. All statistical operations have been performed with R[2].

Our theoretical results only consider mutations that flip exactly one bit. As discussed previously, moving to global mutations such as standard bit mutations in the $(1+1)$ EA can have dramatic consequences. From our experiments, we can indeed see that using standard bit mutations instead of one bit mutations has a significant influence on the performance of the algorithm. We visualise results for $n = 10, 20, \ldots, 70$ using boxplots in Fig. 4. While the increase in time needed

[1] Boxplots depict the minimum, maximum, median and first and third quartiles of the observed runtimes. Circles indicate outliers, which are observations outside 1.5 times the interquartile range above the upper quartile and below the lower quartile.

[2] http://www.r-project.org

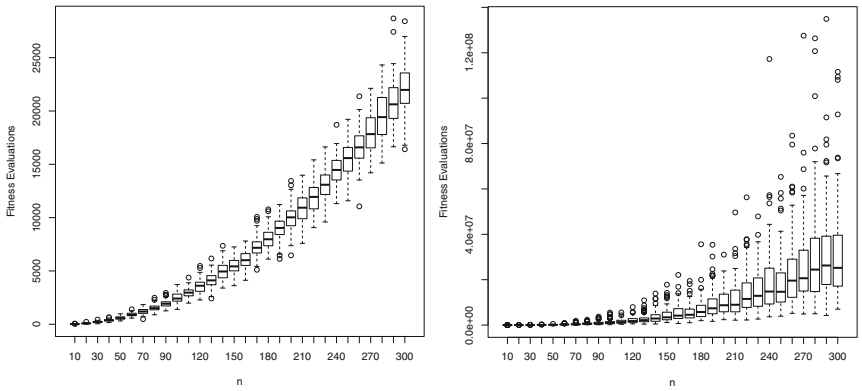


Fig. 1. Experimental results for RLS with lexicase selection over 100 runs. Boxplots show the number of fitness function evaluations until the Pareto front is reached (left) and until the whole Pareto front has been sampled (right).

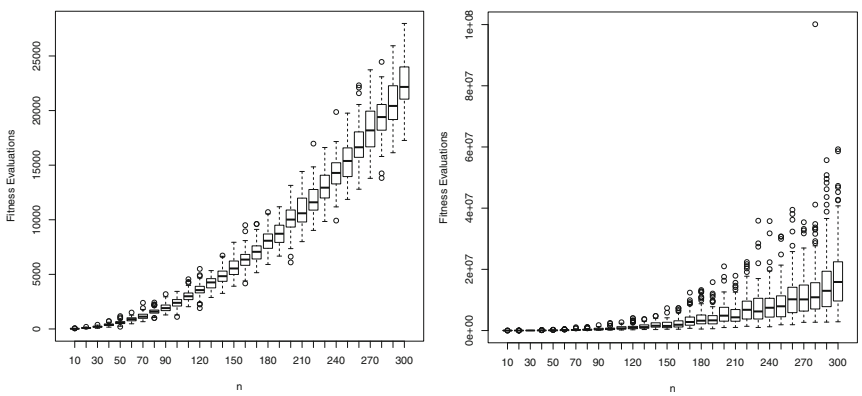
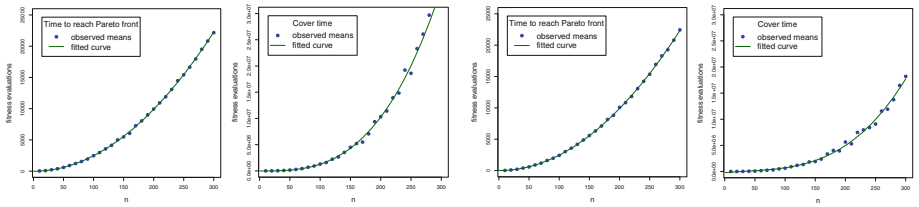


Fig. 2. Experimental results for RLS with weakest selection over 100 runs. Boxplots show the number of fitness function evaluations until the Pareto front is reached (left) and until the whole Pareto front has been sampled (right).



(a) RLS with lexicase selection: (b) RLS with weakest selection

Fig. 3. The average number fitness function evaluations observed in our experiments together with the fitted polynomials matching our theoretical results. For RLS with lexicase selection we obtain $0.2476x^2 - 11.41$ (T_1) and $1.285x^3 - 21\,190$ (T_2). For RLS with weakest selection we obtain $0.2481x^2 - 20.3$ (T_1) and $0.6479x^3 - 50\,520$ (T_2).

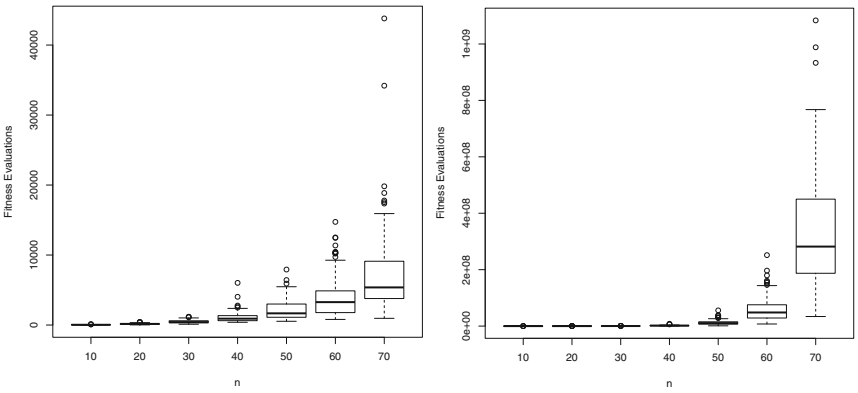


Fig. 4. Experimental results for the $(1+1)$ EA with lexicase selection over 100 runs. Boxplots show the number of fitness function evaluations until the Pareto front is reached (left) and until the whole Pareto front has been sampled (right).

to find the Pareto front seems not too different (but with larger outliers), we see that already for $n = 70$ the average number of fitness function evaluations needed to sample the entire Pareto front is significant larger than the largest outlier for the two algorithms using local mutations. We will examine this case further in the conclusions when discussing questions for future work.

6 Conclusions

We have presented a first theoretical analysis of lexicase selection in the context of discrete multi-objective optimisation. Considering a simple hillclimbing algorithm we show that lexicase selection can be used to efficiently sample the entire Pareto front of the well-known bi-objective benchmark function leading ones trailing zeros (LOTZ). We compare lexicase selection with three multi-objective selection mechanisms from the literature. We obtain asymptotically the same optimisation times when using the mechanism with the weakest selection pressure while classical selection mechanisms based on the (weak) dominance relationship get stuck on the first search point sampled on the Pareto front.

Our study hints towards several interesting questions for future research. We give some insights into these questions in the following, but leave the formal analyses for future research. A comparison with indicator-based selection [24] or considering problems with more than two objectives would also be interesting.

As discussed in the previous section, experiments indicate that moving from local mutations to global mutations significantly increases the runtime of the algorithm. We conjecture that the $(1+1)$ EA with lexicase selection is not able to find the entire Pareto front efficiently. One reason for this is that standard bit mutations may hinder the algorithm to perform a random walk on the Pareto front by preferring a non Pareto optimal search point over a Pareto optimal one.

Consider for example, the search points $x = 1^{50}\,0^{50}$ (50 1-bits followed by 50 0-bits) with fitness $(50, 50)$ and $y = 1^{30}\,0\,1^{17}\,1\,0^{51}$ with fitness $(30, 51)$ for $n = 100$, where y is the result of a 2-bit mutation of x (flipping the bits at positions 31 and 49). With probability $1/2$, lexicase selection chooses the second objective over the first objective and thus, will accept y over x. After that any mutation only effecting the 1-bits at positions 32–48 will be accepted. Note that the $(1+1)$ EA with weakest selection would also accept the new search point as x and y are incomparable.

Since lexicase selection is a parent selection mechanism it would make sense to consider populations, e.g., a $(\mu+1)$ RLS or $(\mu+1)$ EA. We conjecture that simply using lexicase selection for parent selection and negative lexicase selection for replacement will not be efficient. Again, the main reason is that search points that have made some progress on the Pareto front can be lost and replaced by search points that are not Pareto optimal. Consider for example a population that contains $x = 1^{n-1}\,0$ and $y = 1\,0^{n-1}$ and assume that we have not found the two extreme points on the Pareto front, yet. Moreover, we assume that x and y are the two points with fewest trailing zeros and leading ones, respectively. That means that the remaining search points points can be arbitrary points with less than $n-1$ but more than 1 leading ones and trailing zeros. In this case, lexicase selection will select either x or y as parent (depending on the random order of objectives). Assume w. l. o. g. that x was selected and $z = 1^{n-3}\,0\,1\,0$ was created (flipping only the bit at position $n-2$). If negative lexicase selection uses the same objective for replacement, the offspring (that is not on the Pareto front) will replace y. The latter happens with probably $1/2$ and thus, we expect to frequently lose points on the (extreme ends of the) Pareto front.

References

1. De Jong, K.A.: Evolutionary Computation. A Unified Approach. MIT Press, Cambridge (2016)
2. Doerr, B., Jansen, T., Klein, C.: Comparing global and local mutations on bit strings. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008), pp. 929–936. ACM Press (2008)
3. Giel, O.: Expected runtimes of a simple multi-objective evolutionary algorithm. In: Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003), pp. 1918–1925 (2003)
4. Giel, O., Lehre, P.K.: On the effect of populations in evolutionary multi-objective optimisation. Evol. Comput. **18**(3), 335–356 (2010)
5. Helmuth, T.: General program synthesis from examples using genetic programming with parent selection based on random lexicographic orderings of test cases. Ph.D. thesis, University of Massachusetts Amherst (2015)
6. Helmuth, T., McPhee, N.F., Spector, L.: Effects of lexicase and tournament selection on diversity recovery and maintenance. In: Genetic and Evolutionary Computation Conference (GECCO 2016), Companion, pp. 983–990 (2016)
7. Helmuth, T., McPhee, N.F., Spector, L.: The impact of hyperselection on lexicase selection. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016), pp. 717–724 (2016)

8. Helmuth, T., McPhee, N.F., Spector, L.: Lexicase selection for program synthesis: a diversity analysis. In: Riolo, R., Worzel, B., Kotanchek, M., Kordon, A. (eds.) Genetic Programming Theory and Practice XIII. GEC, pp. 151–167. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34223-8_9
9. Helmuth, T., Spector, L.: Evolving a digital multiplier with the PushGP genetic programming system. In: Genetic and Evolutionary Computation Conference (GECCO 2013), Companion, pp. 1627–1634 (2013)
10. Helmuth, T., Spector, L.: Word count as a traditional programming benchmark problem for genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014), pp. 919–926 (2014)
11. Helmuth, T., Spector, L.: General program synthesis benchmark suite. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2015), pp. 1039–1046 (2015)
12. Helmuth, T., Spector, L., Matheson, J.: Solving uncompromising problems with lexicase selection. IEEE Trans. Evol. Comput. **19**(5), 630–643 (2015)
13. Jansen, T.: Analyzing Evolutionary Algorithms. The Computer Science Perspective. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-17339-4
14. La Cava, W., Helmuth, T., Spector, L., Moore, J.H.: A probabilistic and multi-objective analysis of lexicase selection and epsilon-lexicase selection. Evol. Comput. (2018, to appear). http://doi.org/10.1162/evco_a_00224
15. La Cava, W., Spector, L., Danai, K.: Epsilon-lexicase selection for regression. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016), pp. 741–748 (2016)
16. Laumanns, M., Thiele, L., Zitzler, E., Welzl, E., Deb, K.: Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem. In: Guervós, J.J.M., Adamidis, P., Beyer, H.-G., Schwefel, H.-P., Fernández-Villacañas, J.-L. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 44–53. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45712-7_5
17. Laumanns, M., Thiele, L., Zitzler, E.: Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. IEEE Trans. Evol. Comput. **8**(2), 170–182 (2004)
18. Liskowski, P., Krawiec, K., Helmuth, T., Spector, L.: Comparison of semantic-aware selection methods in genetic programming. In: Genetic and Evolutionary Computation Conference (GECCO 2015), Companion, pp. 1301–1307 (2015)
19. McPhee, N.F., Donatucci, D., Helmuth, T.: Using graph databases to explore the dynamics of genetic programming runs. In: Riolo, R., Worzel, B., Kotanchek, M., Kordon, A. (eds.) Genetic Programming Theory and Practice XIII. GEC, pp. 185–201. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34223-8_11
20. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
21. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming (2008). http://lulu.com, http://www.gp-field-guide.org.uk
22. Spector, L.: Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. In: Genetic and Evolutionary Computation Conference (GECCO 2012), Companion, pp. 401–408 (2012)
23. Timmis, J.: Artificial immune systems. In: Sammut, C., Webb, G.I. (eds.) Encyclopedia of Machine Learning and Data Mining, pp. 61–65. Springer, New York (2017)
24. Zitzler, E., Künzli, S.: Indicator-based selection in multiobjective search. In: Yao, X., et al. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 832–842. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30217-9_84