BEYOND BENCHMARKS SUITES: ENGINEERING DIAGNOSTIC TOOLS TO CHARACTERIZE SELECTION SCHEMES

By

Jose Guadalupe Hernandez

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science - Doctor of Philosophy
Ecology, Evolution, and Behavior - Dual Major

2023

ABSTRACT

Evolutionary algorithms (EAs) draw inspiration from biological evolution and replicate evolutionary processes into a computational framework that can often solve challenging optimization problems. These algorithms evolve a population of candidate solutions, where the population typically cycles through three phases: evaluation, selection, and reproduction. Specifically, the evaluation phase assesses the qualities of the candidate solutions, the selection phase identifies which regions will be searched further, and the reproduction phase identifies the next positions to search. Clearly, each phase plays a specific role in the evolutionary search that is implemented through one or more interacting components that fully specify the algorithm. Of course, interactions can make it difficult to isolate individual components in some complex EAs. As such, if we want to understand how each component affects the properties of the overall algorithm, we need a framework to formally define each component, and we need tools that characterize how each component contributes to overall problem-solving success.

When a new EA is proposed, it is typically evaluated against a benchmark suite or hand-picked test problems that clearly demonstrate its capabilities. Multiple benchmark suites exist to highlight which classes of problems an EA is most effective against. Such suites, however, are limited in their ability to diagnose *why* an EA performs the way it does. In particular, problems with complex fitness landscapes do not facilitate an intuitive understanding of how an algorithm traverses the search space. At a high level, components in an EA are well-classified for the role they are supposed to play in traversing a search space: evaluation components generate qualities for a candidate solution; selection components use these qualities to identify parents; reproduction components propagate parents and apply variation. However, it is often less clear which particular components would be most effective on a given problem or how different components will alter each other's behavior. Given the importance of component features and interactions, my aim is to disentangle the mechanistic effects of each choice on the search process so that we can better anticipate which combinations of components are most likely to produce an optimal solution to a given problem.

In this dissertation, I achieved three synergistic goals: (1) I developed a formal definition for selection scheme components that provides a framework for their study within generational EAs; (2) I crafted a set of diagnostic tools that allow me to isolate the effects of individual selection

scheme components within this framework; and (3) I used these diagnostics to characterize the search strategies employed by a set of common selection schemes.

In the chapters below, I first present a formal framework for dividing any selection scheme into three fundamental components: population structures, trait processing, and selectors (Chapter 2). Next, I use lexicase selection as the basis of two case studies where I demonstrate how subtle alterations of this selection scheme affect performance on program synthesis problems, sometimes producing dramatic improvements, but leaving many open questions as to when and why these improvements will occur (Chapters 3 and 4). Once this motivation is established, I improve our toolset for understanding selection schemes by developing a set of diagnostics that more precisely and intuitively measure the strengths and weaknesses of a set of schemes (Chapters 5 and 6). Finally, I apply these diagnostics to a new area, island structures, to demonstrate their versatility and expected general usefulness (Chapter 7). This work emphasizes the importance of properly configuring an EA for the problem at hand, and provides a precise and informative contribution to the set of available benchmark suites.

For my family, friends, and mentors that supported me on this journey to achieving my dream.
Thank you for all your time, encouragement, and patience – I can only hope to pay it forward

ACKNOWLEDGEMENTS

As I wrap up this dissertation, I cannot help but reflect on the 10-year collegiate journey it took to get here, all of which occurred at Michigan State University (MSU). This dissertation would not be possible without all the love, support, and guidance I received throughout my life, from preschool to graduate school. Additionally, this dissertation would not be possible without the sacrifices countless others have made before me to give someone like me the opportunity to be in the spot I am today. I will continue to carry the torch and I will advocate for diversity, equity, and inclusion within STEM and academia.

From an early point in my life, my elementary, middle, and high school teachers saw something in me that illustrated that I can become a college graduate. **Thank you to all my teachers** for pushing, motivating, and inspiring me. Thank you for presenting my mom with the idea of me becoming a college graduate, I would not be here without you all. Specifically, thank you **Peter Montoya** and **Da Xiong** for motivating me to join AVID to become college ready, thank you **Rebekah Esau** for being a second mom throughout my high school years, and thank you **Johnathan Holland** for coaching me and giving me my work ethic. Again, thank you to all my teachers for doing their absolute best with the limited resources that come from schools in low-income areas – this is our victory.

Navigating college as a first-generation scholar was a difficult challenge to overcome, but was made a little easier with the support I found at MSU. Thank you to all the **staff at the College Assistance Migrant Program (CAMP)** for all the guidance and support I received throughout my college career. Specifically, thank you **Luis Garcia**, **Elias Lopez**, and **Aleida Martinez**. The notion of me becoming a graduate student was also daunting, but became more realistic through the Summer Research Opportunities Program – thank you **Steven Thomas** for giving me my first research opportunity and motivation to continue to graduate school. Thank you **Dr. Eric Torng** for motivating and inspiring me to consider graduate school. Throughout my entire college career, I was fortunate to gain another mom who took me in like one of her own – thank you **Teresa Isela VanderSloot**, words cannot describe how much I appreciate you and am grateful for you.

Once a graduate student, I quickly realized that research was a team effort and I would not have excelled without my fellow teammates. Thank you to my colleagues, collaborators, and friends in the Devolab, ECODE Lab, and BEACON, I benefited greatly from your insight and friendship. Specifically, thank you **Dr. Acacia Ackles**, **Cliff Bohm**, **Dr. Emily Dolson**, **Austin Ferguson**, **Shakiba Shahbandegan**, **Dr. Josh Nahum**, **Dr. Anselmo Pontes**, and **Kate Skocelas**, and **Dr. Anya Vostinar**, **Sydney Leither**, **Max Foreback**, **Dr. Matthew Moreno**, and **Santiago Rodriguez Papa**.

This dissertation has blossomed with the advice of my committee members: **Dr. Charles Ofria**, **Dr. Alexander Lalejini**, **Dr. Wolfgang Banzhaf**, **Dr. Emily Dolson**, and **Dr. Bill Punch**. Thank you for the fruitful discussions and perspectives that guided this dissertation.

A special thank you must be given to both **Dr. Charles Ofria** and **Dr. Alexander Lalejini**, as they helped me grow into the scholar I am today. I would not be where I am today without your support, guidance, and friendship. Thank you **Alex** for taking me under your wing as a researcher during my Junior year of college, your patience and guidance is greatly appreciated. Thank you **Charles** for agreeing to be my advisor, working with you demonstrated to me what it means to be an advisor, researcher, scholar, and teacher. I can go on forever why I am so grateful for both of you, but thank you both

Para mi apá, amá, familia, y quieridos: gracias por todo su amor, apoyo, y paciencia. **Amá** y **Apá**, gracias por su apoyo y amor durante mi carrera universitaria, ya pueden decir que su hijo es un doctor. **Jesse** y **Ruby**, gracias por hacer mi ultimo año mas divertido, los quiero mucho. **Leslie Claro**, gracias por acompañarme durante este processo, tu has hecho este proceso mas mejor y agradable. Mi diploma va a tener mi nombre, pero en realidad es de todo nosotros y todos los que nos han apoyado.

TABLE OF CONTENTS

Chapter 1 Introduction 1

Part I Formalizing selection schemes and demonstrating that subtleties alter performance 24

Chapter 2 Selection Scheme Framework 25

Chapter 3 Random subsampling improves performance in lexicase selection 34

Chapter 4 Characterizing the Effects of Random Subsampling on Lexicase Selection . . . 42

Part II Characterizing search strategies for selection schemes 65

Chapter 5 An Exploration of Exploration: Measuring the ability of lexicase selection to find obscure pathways to optimality 66

Chapter 6 A suite of diagnostic metrics for characterizing selection schemes 91

Chapter 7 Diagnosing Island Structures Within Selection Schemes 119

Chapter 8 Conclusions 147

BIBLIOGRAPHY 156

Chapter 1
Introduction

"How one goes about selecting and applying [evolutionary algorithms] to particular problem domains is presently more of an art than a science" – Kenneth De Jong (1993)

Evolutionary algorithms (EAs) constitute a diverse family of optimization algorithms inspired by biological evolution. Most EAs excel at some problem types or instances, but struggle when used on others. The "no free lunch" theorem asserts that no one optimization technique can dominate across all possible problems (Wolpert and Macready, 1997), but we can do a better job identifying the properties of specific EA techniques and which EA should be applied to a given problem. Practitioners are tasked with configuring an EA such that it is tailored to the problem at hand, where the configuration ultimately determines the likelihood of success. This task is rather challenging, as EAs have grown in complexity to contend with more challenging problems.

My thesis focuses on enhancing our current understanding of generational EAs. In it, I accomplish three synergistic aims:

1. I develop a theoretical framework to formally define a selection scheme.
2. I demonstrate how subtleties to a selection scheme impact problem-solving success.
3. I engineer and use a set of diagnostic tools to characterize a selection scheme's abilities.

By formalizing the definition and composition of selection schemes, it becomes much easier to describe, identify, and understand the scheme within an EA. Furthermore, my framework allows us to more easily predict how individual changes to a selection scheme are likely to impact problem-solving success. While it is helpful to find changes that improve success rates, we must understand how the mechanisms behind these changes affect search space traversal to predict where else they might be beneficial and develop more general principles. My diagnostic tools help illuminate why differences in performance and search space traversal may occur due to modifications in a selection scheme. Ultimately, this thesis leaves practitioners better equipped to define their EAs and better able to make decisions on how to configure them.

Evolutionary algorithms are a perfect example of technology inspired by nature, as this family of optimization algorithms lies at the intersection of evolutionary biology and computer science. In nature, organisms within a population compete with one another for the opportunity to reproduce. Offspring typically have genetic variations from their parent or parents, which may affect their traits or behaviors and, ultimately, their ability to survive and continue their lineage. While most genetic changes are deleterious or neutral, those offspring with variations that improve their survival and competitiveness are more likely to produce offspring of their own. Through this cycle of genetic variation and natural selection, populations evolve to be better adapted to their environment (Darwin, 1859). Indeed, evolution is effective at producing organisms with complex traits and behaviors that enable them to thrive in environments where it appears difficult for life to exist, such as deserts, volcanoes, and the deep sea. This description is a simplification of biological evolution, but translating even these high-level evolutionary processes into a computational framework has proven useful for optimization (Barros et al., 2012; Coello Coello, 2002; Freitas, 2003; Hruschka et al., 2009; Karafotias et al., 2015; Slowik and Kwasnicka, 2020; Zhou et al., 2011).

Four main categories of EAs have historically included evolutionary strategies (Rechenberg, 1965), evolutionary programming (Fogel et al., 1966), genetic algorithms (Holland, 1967), and genetic programming (Forsyth, 1981; Koza, 1989). Each category possesses its own unique methods and motivations that distinguish it from the others, yet many similarities exist among them. Traditionally, EAs implement three key phases: evaluation, selection, and reproduction. While defining different EAs with the three phases helps with understanding how an algorithm operates, each individual phase can be implemented as one or more interacting components that execute the role of individual phases. **This dissertation focuses on the selection phase within a generational EA.** The framework developed in Chapter 2 formally defines the selection phase, breaking it into three components: population structure, trait processing, and selectors. Indeed, my framework allows for precise changes to be applied to selection schemes and helps practitioners to predict the impact of those changes on problem-solving success. Additionally, my framework allows practitioners to easily characterize the selection scheme within an EA, thus reducing the likelihood of constructing redundant selection schemes.

After developing the selection scheme framework in Chapter 2, I then demonstrate how small alterations to a scheme can lead to differences in performance for a given problem. In Chapters 3 and 4, I systematically alter lexicase selection (Helmuth et al., 2015) – a highly successful selection scheme for program synthesis problems with a large number of test cases. I engineer promising variations of lexicase selection (cohort lexicase and down-sampled lexicase) that often give better performances than standard lexicase selection and have since been shown to be state-of-the-art for program synthesis (Helmuth and Abdelhady, 2020). While the new lexicase variants proved to be effective, it was initially difficult to precisely and intuitively understand why the changes increased problem-solving success.

Traditionally, newly developed EAs and components are assessed on a benchmark suite (a targeted set of problems) to measure their problem-solving efficacy (Garden and Engelbrecht, 2014). While this may be the standard approach, two critical issues remain to be resolved: (1) no standard benchmark exists (Hussain et al., 2017; Jamil and Yang, 2013), and (2) there is no consensus on methodology for performing benchmark studies (Bartz-Beielstein et al., 2020). The first issue is difficult to resolve, as different benchmark suites focus on specific problem domains and characteristics, and multiple suites are available for practitioners to consider (*e.g.* Adorio and Dilman (2005); Andrei (2008); Averick et al. (1992); Floudas et al. (1999); Hansen et al. (2009); Helmuth and Kelly (2021); Helmuth and Spector (2015); Suganthan et al. (2005)). For Chapter 5, I developed an initial diagnostic tool that starts to help address the second issue. Specifically, I formalized a methodology for isolating and characterizing the exploration abilities of a selection scheme. This methodology differs from the traditional approach for assessing EAs, as I focus only on the selection scheme, which allows me to easily and intuitively diagnose its abilities. I was able to use this diagnostic to better understand the different dynamics of the lexicase variants from Chapters 3 and 4, and identified conditions where each technique was likely to be an appropriate choice. In Chapter 6, I expanded our set of diagnostic tools and applied them to a broader set of commonly used selection schemes, allowing me to quantitatively identify many of the benefits and drawbacks of each. Additionally, my diagnostic tools allow us to diagnose the impact of integrating new components and techniques into a selection scheme. In Chapter 7, I integrated island structures into a broad set of selection schemes and diagnosed their effects under each condition.

1.1 Evolutionary algorithms

Evolutionary algorithms replicate the processes from biological evolution into a computational framework that is applied to problem-solving, where these algorithms can be described as probabilistic search algorithms that use evolutionary processes to sample a subset of possible solutions (Bäck and Schwefel, 1993; Blickle and Thiele, 1995; Goldberg and Deb, 1991). Prior to running an EA, the underlying representation must be determined for the problem at hand, which dictates the space of all possible solutions that will constitute the population of candidate solutions under consideration. At the beginning of an evolutionary run, a population of random solutions is generated. Next, candidate solutions in the population are evaluated on the problem. The results from these evaluations are commonly summarized as a single fitness value, but for generality, we will assume that this evaluation phase can record arbitrary information about the qualities of these solutions, which we will call *traits*. Once one or more traits are recorded, these values can be used to choose the set of candidate solutions to act as parents for the construction of a new generation. Offspring produced from the set of parents will be subject to a source of variation, where variation typically comes from mutations applied to their underlying representation or recombination from the genetic material of other parents or previous candidate solutions. The same process is repeated with the newly formed population until a stopping criterion is met, such as a suitable solution being found or after a maximum number of generations have occurred.

1. **Initialize** the population of candidate solutions.
2. Repeat until *stopping criterion* is met:
   (a) **Evaluation**: Prospective solutions are analyzed and traits are recorded as measures of its quality and performance.
   (b) **Selection**: A number of parents are identified through some procedure that determines which candidate solutions will be chosen as parents for the next generation.
   (c) **Reproduction**: Offspring are produced and subjected to a source of variation.

Algorithm 1.1: Phases for a simple evolutionary algorithm.

The process typically repeated by a simple EA (Algorithm 1.1) can be summarized into three phases: evaluation, selection, and reproduction. While the EA description in Algorithm 1.1 has each phase only once, more complex EAs may have the same phase multiple times in any order. Each evolutionary algorithm phase has its own set of components, some of which interact with components in other phases. For example, consider a component that determines population structure, where a population structure defines how candidate solutions interact with one another. The population is shared across all three phases, where each phase alters and influences the construction of a new population. During the evaluation phase, a population structure might determine the conditions under which a candidate solution is evaluated (*e.g.* cohort lexicase in Chapter 3). During the selection phase, a population structure might influence which other individuals a candidate solution must compete against (*e.g.* age-layered population structures in Hornby (2006)). And during the reproduction phase, a population structure might determine where a candidate solution's offspring is placed (*e.g.* MAP-Elites in Mouret and Clune (2015)). Given the interconnectedness of an EA, we must develop a deeper understanding of how components in different phases interact to create more robust and efficient algorithms.

Before using an EA, practitioners must develop a blueprint for the EA, and determine the *nuts and bolts* of each component that are needed to construct the EA. Once the EA is fully configured, executed, and finished running, it may potentially find optimal solutions for the problem it is attempting to solve. If the EA produced interesting results, practitioners might want to publish their findings and share the algorithm with others. However, the multiple components and the software implementation of the EA may make it difficult to describe and interpret. Given that EA components interact, optimizations used within the software implementation make sense from a practical perspective, but make it difficult to isolate and identify individual components. Describing EAs through written descriptions is another commonly used approach, but the text may be ambiguous, uninformative, and convoluted with both justifications and descriptions of the EA. The confusion generated from the inability to understand an EA can lead to the construction of redundant, inaccurate, and ineffective algorithms, along with impeding the ability to reproduce results similar to existing research. Thus, the ability to formally describe evolutionary algorithms is crucial for developing continuous progress within the field of EAs (López-Ibáñez et al., 2021).

I believe pseudocode is the best approach to use and expand for describing an EA's components and configuration, but more refinement and improvements are needed. The favoring of this approach is not meant to halt the use of software and written descriptions to describe EAs, but rather to ease the understanding of them by using the pseudocode as a bridge between software and written descriptions. Both Bäck et al. (1997) and Rozenberg et al. (2012) successfully describe how specific kinds of evolutionary algorithms operate with the three simple EA phases found in Algorithm 1.1. Yet, this description is insufficient to fully describe an EA used for a specific problem, as the solution representation, selection scheme, and reproduction components are missing. While not complete, this approach is beneficial to understand how an EA is operating at a high level, but more details are needed that define the specific components used. Both Helmuth et al. (2015) and Chapter 5 do a good job of describing and simplifying the understanding of the selection scheme used in their work. In both cases, pseudocode could have simplified the understanding of how the EA operated in their respective works, along with including pseudocode for the reproduction components similar to the selection scheme. Given the benefit of using pseudocode for both EA and component descriptions, I believe pseudocode is the best path forward.

In order to contend with more challenging problems, practitioners developed new techniques, procedures, and components that increase the problem-solving potential of EAs (Vikhar, 2016). Indeed, the state-of-the-art EAs developed have grown in complexity (Hornby, 2006; Kriegman et al., 2020; Mouret and Clune, 2015; Skolicki and De Jong, 2004; Stanley and Miikkulainen, 2002). As the complexity of new EAs grows to contend with more challenging problems, the importance of implementation details and communication of the EA is even more critical. I argue that the three fundamental phases of a simple EA (evaluation, selection, and reproduction in Algorithm 1.1) are a unifying framework for *all* EAs.

Next, I use four classes of EAs as a case study to demonstrate two key ideas: (1) why implementation details are important and (2) how more formal descriptions with pseudocode can illuminate similarities and differences between EAs. Indeed, when all four classes are viewed through the lens of simple EA phases, they all fit into a common framework. The classes of EAs used in this case study are the four historical branches of EAs: evolutionary strategies (Rechenberg, 1965), evolutionary programming (Fogel et al., 1966), genetic algorithms (Holland, 1967), and genetic programming (Koza, 1989, 1990a). In the early stage of these branches, computational resources were scarce and limited by the technology available at the time, which impeded the ability to carry out more complex experiments and develop variants. In other words, the founders of each branch were not limited by their imagination, but by the tools available. However, the founders of these branches paved the way for the progress and success evolutionary algorithms see today. A more complete history of the field of evolutionary computation can be found in Bäck et al. (1997) and Rozenberg et al. (2012). Next, I give a more detailed description of the four main branches in the following sections.

1.1.1 Evolutionary strategies

The field of evolutionary strategies (ES) is credited to three students of the Technical University of Berlin, Peter Bienert, Ingo Rechenberg, and Hans-Paul Schwefel, who initially planned to develop an autonomous system that repeatedly executed experiments and used the results from the experiments to improve a real-world object for a given problem (Bäck et al., 1997; Luke, 2013; Rozenberg et al., 2012). The first evolutionary strategy, (1+1)-ES, used stochasticity in its search for optima to overcome the limitations of gradient-based optimization techniques at the time when tackling problems from engineering domains (Rechenberg, 1965). This evolutionary strategy maintained a single solution that asexually produced an offspring that received mutations, and only the better-performing solution between the parent and offspring continued to the next generation. Indeed, selection and mutation drove the evolutionary search of (1+1)-ES, where the mutation parameters were static and arbitrary. The (1+1)-ES would be further explored and extended, such as mutations becoming more systematic, self-adaptive, and better understood (Rechenberg, 1973; Schwefel, 1965, 1977). For example, in Rechenberg (1973) the $\frac{1}{5}$ rule is established, where if the proportion of successful mutations over a period of time deviates from $\frac{1}{5}$, then the magnitude of the following mutations were adjusted.

Two generic, multi-solution variations of the (1+1)-ES became more widespread (Bäck et al., 1997): (μ, λ)-ES and $(\mu + \lambda)$-ES. For both these new variations of evolutionary strategies, μ is the number of parents selected and λ is the number of offspring created. The (μ, λ)-ES algorithm starts with generating λ solutions randomly. Once each solution receives its evaluation on some problem, the μ top-performing individuals are identified as parents and produce $\frac{\lambda}{\mu}$ offspring. Each offspring undergoes mutation and proceeds to the next generation. The same process is repeated with the newly formed population until a stopping criterion is met, such as a suitable solution being found or after a maximum number of generations have occurred. The $(\mu + \lambda)$-ES algorithm deviates from this procedure, where both offspring and parents are placed in the population for the next generation. Note that the population size for the (μ, λ)-ES algorithm is λ and the population size for the $(\mu + \lambda)$-ES algorithm after the first generation is $\mu + \lambda$. These implementations of evolutionary strategies were gathered from Luke (2013).

According to Rozenberg et al. (2012) and Bäck et al. (1997), canonical evolutionary strate-

```
1. Initialize the population of candidate solutions.
2. Evaluate initial solutions on the given problem.
3. Repeat until stopping criterion is met:
    (a) Select a list of parents from the population.
    (b) Reproduce offspring with mutations applied.
    (c) Evaluate offspring on the given problem.
    (d) Select solutions from offspring and population for next generation.
```

Algorithm 1.2: Pseudocode for canonical evolutionary strategies.

gies are typically executed as follows: at the start, initialize a population of randomly generated solutions and evaluate all solutions on the given problem. Parents are then selected uniformly at random and each parent produces an offspring. Offspring receive mutations and then are evaluated on the same problem. To construct the following population, both offspring and parents are ranked according to their evaluation, where only the top-ranked solutions continue to the next generation. This cycle continues with the newly formed population until a stopping criterion is met, such as a suitable solution being found or after a maximum number of generations have occurred. Interestingly, early work involving evolutionary strategies emphasized mutations so it was thought that recombination was not a key component of evolutionary strategies, but is now part of contemporary evolutionary strategies (Rozenberg et al., 2012). Here, the mutation-based implementation of canonical evolutionary strategies is presented and the pseudocode for this algorithm can be found in Algorithm 1.2.

Indeed, evolutionary strategies have been successfully applied in multiple fields, such as parameter optimization (Hatanaka et al., 1996; Li et al., 2013a), image processing (Li et al., 2006; Louchet, 2000), task scheduling (Ahire et al., 2000; Belaqziz et al., 2014; Greenwood et al., 1994), path planning (Sauter et al., 2002; Watanabe et al., 1999), and vehicle design (Ostertag et al., 1995; Tayarani-N. et al., 2015).

1.1.2 Evolutionary programming

Lawrence Jerome Fogel founded evolutionary programming when simulating evolution to evolve finite state machines that sequentially processed symbols to predict the next symbol within a predetermined environment (Fogel et al., 1966), where environments consisted of a sequence of symbols generated from a finite alphabet. At the start of the simulation, a population of parent finite state machines was randomly generated and each machine was evaluated on the prediction task. Each finite state machine processed a symbol from the environment and predicted the following symbol in the sequence, where a payload function scored each prediction. Once all the predictions for a finite state machine were scored, they were summarized by a different payload function to quantify a machine's *fitness*. Next, all parent machines asexually generated an offspring, where offspring possibly received mutations (changes to output state symbol, initial state, and topology). Once all offspring were constructed, each offspring was evaluated on the prediction task. Only the machines with the highest fitness made it to the next generation between both parents and offspring solutions. The same cycle was repeated until a stopping criterion was met. This evolutionary process ultimately laid the framework for the field of evolutionary programming.

```
1. Initialize the population of candidate solutions.
2. Evaluate initial solutions on the given problem.
3. Repeat until stopping criterion is met:
    (a) Select a list of parents from the population.
    (b) Reproduce offspring with mutations applied.
    (c) Evaluate offspring on the given problem.
    (d) Select solutions from offspring and population for next generation.
```

Algorithm 1.3: Pseudocode for canonical evolutionary programming.

According to Fogel et al. (1991), Bäck et al. (1997), and Rozenberg et al. (2012), canonical evolutionary programming is typically implemented as follows: at the start, initialize a population of random solutions for the problem at hand, where all initial candidate solutions are assigned an evaluation from one or more payoff functions. To construct the following generation, all candidate solutions asexually produce one offspring, and mutations are applied to the offspring; mutations are implemented such that the offspring's behaviors do not deviate far from their parents. Once all offspring are constructed, they receive an evaluation from the payoff functions. The population for the next generation is filled with the best solutions from both the current population and offspring.

This cycle continues with the newly formed population until a stopping criterion is met, such as a suitable solution being found or after a maximum number of generations have occurred. The pseudocode for this algorithm can be found in Algorithm 1.3.

While the first experiment with evolutionary programming focused on evolving finite state machines for a prediction problem, the same evolutionary process would be extended (Fogel et al., 1991; Swain and Morris, 2000; Yao et al., 1999). Indeed, evolutionary programming would be successfully applied in various areas, such as the traveling salesman problem (Fogel, 1988, 1993), neural networks (Fogel et al., 1995a, 1997; Jian and Yugeng, 1997), and constrained optimization (Fong et al., 2006; Hoorfar, 2007; Kim and Myung, 1997; Shailti Swamp and Natarajan, 2005).

1.1.3 Genetic algorithms

The field of genetic algorithms is credited to John Holland (Holland, 1967), and the field was extended and explored by his students and colleagues (Bäck et al., 1997; Rozenberg et al., 2012). In Holland (1962), Holland sets out to develop a better understanding of adaptation, or how systems can generate solutions that thrive in a given environment. Ultimately, this goal of better understanding adaptation laid the groundwork for genetic algorithms, where Holland describes populations of programs that accumulate changes that increase the ability of a program to thrive in a given environment each following generation. Later, Holland developed the schema theorem that would make certain guarantees for genetic algorithms, which states that schema with low-order, small defining length, and above-average fitness have a higher chance of continuing to the next generation (Holland, 1975). There are three features that distinguish Holland's first proposed genetic algorithm: a genome represented by a bitstring, proportional selection, and variation through mutation and recombination. In fact, this early variation of genetic algorithms preferred recombination over mutation.

In Bäck et al. (1997) and Rozenberg et al. (2012), canonical genetic algorithms are typically executed as follows: at the start, initialize a population of randomly generated solutions and evaluate all solutions on the problem at hand. Next, select a set of parents for reproduction, where offspring are generated through crossover and mutations are applied. The parents selected for reproduction are identified probabilistically, where the probability of a solution being selected is determined by its performance. Once all offspring are constructed, they are evaluated on the given problem. The next generation is then constructed from both the current population and the set

```
1. **Initialize** the population of candidate solutions.
2. **Evaluate** initial solutions on the given problem.
3. Repeat until *stopping criterion* is met:
    (a) **Select** a list of parents from the population.
    (b) **Reproduce** offspring through recombination and mutations.
    (c) **Evaluate** offspring on the given problem.
    (d) **Select** solutions from offspring and population for next generation.
```

Algorithm 1.4: Pseudocode for canonical genetic algorithms.

of offspring, where solutions must be selected to continue to the next generation. Additionally, not all solutions may be needed to fill the next population. This cycle continues with the newly formed population until a stopping criterion is met, such as a suitable solution being found or after a maximum number of generations have occurred. Interestingly, genetic algorithms emphasize recombination over mutations. The pseudocode for this algorithm can be found in Algorithm 1.4.

Indeed, genetic algorithms have proven their abilities in numerous fields (Katoch et al., 2021), such as healthcare (Ghosh and Bhattacharya, 2020; Sharma and Kumar, 2022), scheduling (Hou et al., 1994; Pezzella et al., 2008), security (Devaraj and Yegnanarayana, 2005; Kaur and Kumar, 2018), image processing (Bhanu et al., 1995; Hashemi et al., 2010), and neural networks (Ding et al., 2011; Leung et al., 2003; Miller et al., 1989).

1.1.4 Genetic programming

Prior branches of evolutionary algorithms focused on solving individual instances of problems, whereas genetic programming (GP) shifted the emphasis to evolving *code* that could solve all instances of a given problem Koza and Poli (2005). Implementations of GP can be found as early as Forsyth (1981), but became popularized by John Koza (Koza, 1990b). Early implementations of GP used syntax trees to represent a program, which remains a popular technique. These trees, inspired by LISP programming, consist of three key features (Koza, 1989, 1990a): functional nodes, terminal nodes, and the edges connecting them. Functional (internal) nodes process inputs and generate output, such as arithmetic, Boolean, or conditional operations. Terminal (leaf) nodes typically consist of inputs to the program, variables, or constants. Edges connect the outputs from one node to the inputs of another. While syntax-tree representations are popular, other types of GP representations include stack-based GP (Perkis, 1994), grammatical evolution GP (O'Neill and Ryan, 2001), cartesian GP (Miller, 1999), linear GP (Brameier et al., 2007), and graph programming (Atkinson et al., 2018). This flexibility in representation allows practitioners to solve a wide range of problems with GP (De Jong, 1988). The theoretical foundations for GP had a slow start, but eventually, an exact and general schema theory was developed that helped explain its problem-solving success (Poli, 2001; Poli and McPhee, 2003a,b).

```
1. Initialize the population of candidate programs.
2. Evaluate programs in the population on each fitness case.
3. Repeat until stopping criterion is met:
   (a) While offspring are needed choose an operation randomly:
         i. Select a parent to produce a clonal offspring.
        ii. Select two parents to recombine to produce an offspring.
       iii. Select a parent to reproduce with mutations.
        iv. Select a parent to reproduce with architecture-altering operations.
   (b) Evaluate offspring on each fitness case.
   (c) Replace the current population with the population of offspring.
```

Algorithm 1.5: Pseudocode for canonical genetic programming.

Since GP focuses on evolving actual *computer programs*, these programs must be executed to measure how well they solve the given problem. Typically, multiple *fitness cases* are used to measure problem-solving success and guide the evolutionary search toward optima, where infinitely many cases may exist for a given problem (*e.g.* evolving a generalized sorting algorithm (Kinnear, 1993)). Indeed, since the full program must be run for each fitness case, a trade-off must be made for the number of cases used to evolve programs. An excessive number of cases will increase run time, and an insufficient number of cases will not adequately guide the evolutionary search. Ultimately, the collection of fitness cases must capture the complete set of capabilities required of a program in order to successfully solve a problem. Additionally, techniques have been developed to select fitness cases for problems with infinitely many possibilities or to limit cases when fitness evaluations are computationally expensive (Curry and Heywood, 2004; Gathercole and Ross, 1994; Giacobini et al., 2002; Hmida et al., 2017; Martínez et al., 2017; Ross, 2000). Indeed, GP has proven its abilities in numerous fields (Langdon et al., 2008), such as automated machine learning (Olson and Moore, 2019), healthcare (Le et al., 2019), classification (Espejo et al., 2010), scheduling (Nguyen et al., 2017), image processing (Khan et al., 2021), and civil engineering (Zhang et al., 2021).

```
1. Initialize the population of candidate programs.
2. Evaluate initial programs on each fitness case.
3. Repeat until stopping criterion is met:
    (a) Select a list of parents from the population.
    (b) Reproduce offspring by choosing a reproduction method and working
        down the list of selected parents.
    (c) Evaluate offspring on each fitness case.
    (d) Select programs from offspring and population.
```

Algorithm 1.6: Pseudocode for canonical genetic programming adjusted to match other evolutionary algorithm organization.

According to Koza and Poli (2005), canonical genetic programming implementations are typically executed as described in Algorithm 1.5. This cycle continues until a stopping criterion is met, typically because a suitable program was found or the maximum number of generations occurred. Interestingly, the selection scheme used to identify parents is similar between both genetic programming and genetic algorithms. This canonical pseudocode description may appear different from the previous EA branches, but we can adjust the description to follow the same format. An example of this new pseudocode description can be seen in Algorithm 1.6. I will use this more standard formulation for my subsequent analyses.

1.2 Differences and similarities for evolutionary algorithm branches

Clearly, each EA branch can be adjusted to operate similarly and use identical phases, yet the components of each phase are implemented and function differently. In this section, I focus on highlighting the key differences and similarities between the canonical algorithms of the four EA branches.

1.2.1 Differences in initial purpose, design philosophy, and problem domain

Here I describe the differences between evolutionary strategies, evolutionary programming, genetic algorithms, and genetic programming in relation to each branch's purpose, philosophy, and problem domain.

Differences in purpose between branches

The purpose of the canonical algorithm for each EA branch is the first clear difference among branches (Bäck et al., 1997; Rozenberg et al., 2012). Evolutionary programming attempted to address the goals of artificial intelligence at the time, which focused on developing heuristics and neural networks. Evolutionary strategies were intended to act as an autonomous system for modifying real-world objects by repeatedly conducting experiments on the object and applying changes to them based on the experiment results. Genetic algorithms were intended to study the principles of adaptive systems. Genetic programming focused on getting computers to automatically solve a problem.

Differences in design philosophy between branches

Each branch focuses on different evolutionary principles and mechanisms that guide the evolutionary search to solve a problem. Both genetic algorithms and genetic programming evolve solutions based on the assumption that the accumulation of building blocks within the genotype will guide the evolutionary search toward optima. Schema theory reinforced this motivation, as Holland (1975) and Poli (2001) demonstrated that solutions with small building blocks will be improved over time. Additionally, the use of recombination as a variation operator goes hand in hand with the building block motivation, as solutions may combine building blocks within their offspring. Because of this preference for building blocks, both genetic algorithms and genetic programming focused on evolving a genotype, where the former is typically an indirect encoding for a solution to a given problem and the latter is an executable computer program for a given problem.

Evolutionary programming and evolutionary strategies focus the evolutionary search on phenotypic behaviors and make no assumption on the genotypic representation (Bäck et al., 1997; Rozenberg et al., 2012). As a result, recombination did not directly benefit the improvement of phenotypic behaviors, which explains the emphasis on mutations for both branches. Indeed, different approaches for creating robust and effective mutation operators would be developed for both evolutionary strategies (Hansen and Ostermeier, 1996, 2001; Yao and Liu, 1997) and evolutionary programming (Fogel et al., 1995b; Lee and Yao, 2004; Zhao et al., 2007). Although, recombination would become part of these branches later on (Beyer and Schwefel, 2002; Fogel and Beyer, 1995).

Differences in target problem domain and genetic representation between branches

As the original purpose and philosophy varied between the EA branches, so did the problems each was initially designed to solve. Thus, the solution representation also differed: evolutionary programming evolved finite state machines for a simple prediction task (Fogel et al., 1966); evolutionary strategies evolved real-world objects for different engineering domains (Rechenberg, 1965); genetic algorithms evolved binary strings that represented encoded solutions for a variety of problems (Bäck et al., 1997; Rozenberg et al., 2012); and genetic programming evolved executable code (often in the form of syntax trees) for a variety of problems (Koza, 1989). While the initial representation and problem domains for each branch may be different, they have expanded into other domains, and even overlap on some problems. For example, evolutionary programming, evolutionary strategies, and genetic algorithms attempt to solve different instances of the traveling salesman problem through their own unique approach (Fogel, 1993; Karabulut et al., 2021; Larranaga et al., 1999).

1.2.2 Comparison of similarities and components

Here, I describe the similarities between evolutionary strategies, evolutionary programming, genetic algorithms, and genetic programming. When viewing each EA branch through the three phases of a simple EA, each branch follows the same phases: initialization, evaluation, selection, reproduction, evaluation, and selection. While each branch may follow the same phases, each phase is implemented differently. The pseudocode description for evolutionary programming (Algorithm 1.3), evolutionary strategies (Algorithm 1.2), genetic algorithms (Algorithm 1.4), and genetic programming (Algorithm 1.6) can be found in Section 1.1. Indeed, I illustrate the benefit of using both the pseudocode and the simple EA phase framework to describe each historical branch of EAs and how they operate. However, the ability to understand how an EA is operating is only half the battle, as it is also important to understand how each component is implemented.

Initialization

All four branches start by initializing a population of random solutions and evaluating those solutions on the problem at hand. While the creation of the starting population seems like a special phase on its own, I interpret it as a special case of reproduction where offspring are randomly created. In fact, reproduction components that generate random solutions are common, as randomly generated solutions are frequently added in the population for the Age-Layered Population Structure algorithm (Hornby, 2006) and added to increase exploration (Grefenstette, 1992). After these first two steps, while all branches appear to follow the same phases, the components are implemented differently.

Parent selection

Step (a) identifies a set of parents that are used to generate a set of offspring. However, the selection component is implemented differently across each branch. Interestingly, the implementation of evolutionary programming in Section B1.4.2.2 from Bäck et al. (1997) does not have this line and the entire population acts as parents, but I call out this additional phase to highlight the similarities. Evolutionary strategies select parents uniformly at random. Both genetic algorithms and genetic programming identify parents through the use of proportional selection techniques. Indeed, step (a) is established for all branches, but the selection component implementation differs.

Reproduction and offspring evaluation

Step (b) uses the set of parents identified from step (a) to generate a set of offspring. Note that step (b) is the same for evolutionary programming and evolutionary strategies. Both evolutionary programming and evolutionary strategies have each identified parent produce offspring asexually, where offspring receive mutations. Genetic algorithms, however, use crossover to generate offspring from the set of parents identified, where offspring also receive mutations. Genetic programming uses a variety of ways to generate offspring: clonal, crossover, and mutation. Once the set of offspring is constructed, each offspring is evaluated on the given problem in step (c). Indeed, step (b) is established for all branches, but the reproduction component implementation differs.

Survivor selection

Step (d) identifies the solutions that will survive and form the population for the next generation. Interestingly, this step can also be viewed as a selection phase, as solutions must be selected to continue to the following generation. For evolutionary strategies, evolutionary programming, and genetic algorithms, the surviving solutions come from both the set of offspring and the set of identified parents. Both evolutionary programming and evolutionary strategies only keep the top-performing solutions, also known as truncation selection (Crow and Kimura, 1979). Genetic algorithms use a variety of techniques to select solutions that continue to the following generation: proportional selection, rank selection, and tournament selection. Genetic programming, however, only uses the offspring generated to construct the following generation. Step (d) is not found within the canonical implementation of genetic programming, but I call out this additional phase to highlight the similarities. Indeed, step (d) is established for all branches, but the selection component implementation to identify survivors differs.

1.3 Evolutionary algorithms and benchmark suites

The flexibility in constructing EAs makes them applicable to a wide range of problems. In fact, an EA can be used to solve a problem as long as a problem representation can be defined with a variation operator that is capable of traversing the search space. Yet, even if state-of-the-art evolutionary algorithms are applied to a new problem, there is no guarantee that the algorithm will find an optimal solution. In order to maximize problem-solving success, practitioners are challenged with both deciding which EA to use and how to tune it appropriately. Tuning is especially challenging because an EA may have an intractable number of configuration options due to either combinatorics or continuous values. As such, the broad applicability of EAs to virtually *any* problem domain makes it difficult to formally analyze them. In the end, EAs tend to be individually customized for each given problem.

Running an EA on a benchmark suite is the standard approach for (1) predicting its usefulness, (2) comparing it to other approaches, and (3) measuring its strengths and weaknesses. Success on any problem in a benchmark suite demonstrates that it can be useful in at least some circumstances. However, using benchmark suites to broadly compare EAs is more challenging. Practitioners must individually determine the set of problems to evaluate EAs on, as no consensus benchmark suite exists (Hussain et al., 2017; Jamil and Yang, 2013), which can lead to biased results. Approaches to mitigate this bias focus on generating diverse benchmark suites through large numbers of problems (Whitley et al., 1996), using heuristics to select problems with desired features (Lang and Engelbrecht, 2021), or targeting an appropriate range of fitness landscape properties (Doerr et al., 2019). Furthermore, even if an EA performs well on a problem, it is not obvious how to use this knowledge to predict success on new instances of the same type of problem, let alone problems from other domains. Our inability to extrapolate problem-solving success is due to several factors: it is challenging to perform sufficient replications to get robust statistical results (Vermetten et al., 2022), statistical analyses are often incomplete (López-Ibáñez et al., 2021), existing benchmark suites are often limited in the types of problems they include (Garden and Engelbrecht, 2014), and even when problems are varied, the tests will often only reach a small region of the entire problem space (Lacroix and McCall, 2019).

The high-level mechanisms for how an EA moves a population through a search space are well-established: mutation transforms a solution into one of its neighboring solutions; recombination generates offspring somewhere between parents within the search space; and selection identifies regions that the population should continue to search. Yet, it is difficult to know how these properties manifest during actual problem-solving, as solution representations and complex search spaces are typically not conducive to an intuitive understanding. Many search space characteristics that make problems challenging are well-known (Malan and Engelbrecht, 2013; Sun et al., 2014; Weise et al., 2012), such as the number of peaks (modality), the number of basins of attraction, landscape ruggedness, neutrality, dimensionality, separability, deception, epistasis, genotype/phenotype redundancy, *etc.* Capturing these characteristics is feasible and can be intuitive for simple search spaces, but quickly becomes unmanageable as search space complexity increases. For example, measuring any of these characteristics for a tree-based genetic programming search space for a synthesis problem would be computationally intractable, as the search space is complex and unbounded.

Constructing new benchmark suites composed of handcrafted problems with targeted problem characteristics can be a useful addition to the current benchmarking standards. In Weise et al. (2008), the authors present the W-Model, a problem where multiple parameters are adjustable such that the problem difficulty and characteristics are tunable. Specifically, the W-Model defines the transformation of a bit string to a single fitness value, where the transformation applies a layer of neutrality, epistasis, multi-objectivity, overfitting, and ruggedness in that order. Each layer transforms the bitstring and passes the transformed bitstring to the following layer. The W-Model illustrates its importance by highlighting algorithm differences when difficulty is increased (Weise et al., 2020). It has also been used to illustrate how self-adapting mutation rates and population size influence problem-solving success (Rodionova et al., 2019). Indeed, the W-model makes a strong case to be considered part of the Black-Box Discrete Optimization Benchmarking Workshop at The Genetic and Evolutionary Computation Conference (Weise and Wu, 2018). Other examples of tunable, handcrafted problems include royal road (Mitchell et al., 1991), NK-Landscapes (Kauffman and Levin, 1987), One-Max (Bäck et al., 1997), and royal trees (Punch et al., 1996).

This dissertation builds on this idea of generating handcrafted problems to analyze EAs, where I develop a suite of diagnostics that focus on specific problem characteristics (*e.g.*, modality, multi-objectivity, epistasis, *etc.*). Additionally, I narrow my analysis of EAs by only focusing on the selection scheme and characterizing its abilities through the set of diagnostics. This approach forms a new methodology for studying EAs by isolating the selection scheme within an EA and comparing the results on the diagnostics, where only the selection scheme changes within the EA. Indeed, I help establish a better understanding of the similarities and differences between selection schemes. One argument against this approach is the fact that these handcrafted problems are unrealistic and are not found in the real world. My position is that success on real-world problems is important to prove the value of an EA, but these diagnostics are critical if we want to understand the underlying mechanisms that lead to that success. This understanding, in turn, is necessary to improve the mechanisms and apply them to other EAs more broadly.

1.4 Thesis Statement

Evolutionary algorithms (EAs) have proven to be extraordinarily successful on many classes of problems where humans do not have the knowledge to craft a more customized optimization algorithm. While many people have analyzed EAs from a theoretical perspective, a more formalized framework is required if we are to make substantial progress on turning EA construction from an art to a science. Furthermore, more targeted diagnostics can improve our analysis capabilities to not only measure an EAs performance (like so many benchmark suites do), but also provide us with a deeper intuition for how the underlying EA dynamics function.

1.5 Contributions

This thesis is divided into two parts: (1) Formalizing selection schemes within a generational EA and demonstrating that subtleties to a scheme alter performance; (2) Engineering a set of diagnostics to analyze selection schemes. Chapters 2, 3, and 4 focus on the former, while Chapters 5, 6, and 7 focus on the latter.

1.5.1 Formalizing selection schemes and demonstrating that subtleties alter performance

Evolutionary algorithms have multiple components that interact with one another and influence problem-solving success. To better understand the many strengths and weaknesses of an evolutionary algorithm, it is crucial to understand how each individual component influences success. This dissertation focuses on analyzing a key component found across many evolutionary algorithms — the selection scheme. While the number of selection schemes grows, there is still no consistent method to denote a selection scheme. Thus, the first task I attempt to overcome is formalizing what a selection scheme is, which was the focus of **Chapters 2**. More specifically, I demonstrate that a selection scheme can be described through three components: population structures, trait processing, and selectors. With this new framework, I easily alter and analyze different selection scheme configurations.

Chapters 3 and 4 present two case studies that demonstrate how subtle changes to a selection scheme can lead to different results. I leverage my selection scheme framework and alter different components of lexicase selection, ultimately creating new lexicase variants that I analyze in Chapters 3 and 4. The results from these changes should not be surprising, as it is common to see an evolutionary algorithm succeed in a particular problem domain or problem instance but fail when attempting to solve a different one. Thus, I identify differences in performance, but leave further investigation as to *why* these differences occur in later chapters as I build the necessary tools to do so.

Chapter 3 focuses on reducing the number of evaluations standard lexicase selection requires to identify a parent solution by incorporating two techniques: random subsampling of test cases each generation or by assigning test case partitions to subgroups of the population. The former is down-sampled lexicase and the latter is cohort lexicase. I evolved populations of linear genetic programs to solve five different programming synthesis problems, where standard, down-sampled, and cohort lexicase are used to identify parent solutions. Additionally, I used a variety of down-sampling levels and cohort sizes to better understand the impact of the subsampling techniques. For each problem and selection scheme combination, I analyzed and visualized the problem-solving success rates. I make two key findings in this chapter: (1) The random subsampling of test cases each generation can improve the problem-solving performance of lexicase selection, and (2) both cohort and down-sampled lexicase variants are successful approaches for applying random subsampling to standard lexicase. More specifically, I find that optimal configurations of down-sampled and cohort lexicase depend on the problem at hand. This poses the question, why did these improvements occur?

Chapter 4 attempts to shine a light on this question, as I take a deeper look into why subsampling could improve the performance of lexicase selection. I ran four experiments to characterize the effects of applying random subsampling to lexicase selection, where I evolved populations of linear genetic programs on four program synthesis problems. In Chapter 4, I present three key findings:

- With a fixed number of generations, rather than a fixed number of evaluations, down-sampled and cohort lexicase did not significantly outperform lexicase selection.
- Both down-sampled and cohort lexicase used significantly fewer evaluations than standard lexicase selection on all four problems to produce solutions (10% subsampling rate).
- Subsampling degrades lexicase selection's specialist maintenance.

Altogether, this chapter demonstrates how changes to lexicase selection cause different results, yet additional work is needed to identify why these differences occurred.

1.5.2 Building a set of diagnostics to analyze selection schemes

The previous chapters illustrate the importance of designing an algorithm best suited for the problem at hand, or the algorithm may fail to find high-performing solutions. Information about the strengths and weaknesses of various evolutionary algorithms will help with choosing and configuring an algorithm for a specific problem. Benchmark suites provide the standard approach for evaluating evolutionary algorithms. While benchmark suites provide useful insight into the kinds of problems an evolutionary algorithm is effective against, problems with complex search space topologies make it difficult to intuitively understand how each component is influencing problem-solving success. I propose using a set of carefully handcrafted search spaces with targeted problem characteristics to evaluate selection schemes, where the problem characteristics of interest are abstracted from real-world problems (*i.e.*, exploitation, exploration, modality, *etc.*).

Chapter 5 introduces the "exploration diagnostic" (later refined to be called "the multi-path exploration diagnostic") as a new tool for measuring the exploratory capacity of lexicase selection and several of its variants: epsilon lexicase, down-sampled lexicase, cohort lexicase, and novelty-lexicase. All optimization problems require a targeted balance of exploitation and exploration to find high-quality solutions; thus, capturing a selection scheme's exploitation and exploration abilities is useful information. The exploration diagnostic generates a search space with multiple pathways, differing in path length and peak height, but identical in slope. Selection schemes are challenged with navigating populations through the search space, with the goal of pursuing the pathway leading toward the global optimum. I found that lexicase selection facilitates better search space exploration than tournament selection on our diagnostic, and lexicase's exploratory capacity is sensitive to the ratio between population size and the number of test cases. Additionally, I found that epsilon lexicase outperforms standard lexicase selection, where the relaxation of lexicase selection's elitism is incorporated in epsilon lexicase. All other variants degraded the exploratory capacity of lexicase selection. These results demonstrate the importance of diagnostics, as I uncovered key differences between lexicase selection and its variants with the exploration diagnostic.

Chapter 6 expands the set of diagnostics, by introducing three additional entries:

- An "exploitation rate" diagnostic to measure a selection scheme's ability to exploit a smooth, non-epistatic fitness gradient.
- An "ordered exploitation" diagnostic to measure a selection scheme's ability to pursue a single, narrow gradient that leads toward a global optimum.
- A "contradictory objectives" diagnostic to measure a selection scheme's ability to locate and optimize conflicting objectives.

I use the diagnostics to evaluate six popular selection schemes: truncation selection, tournament selection, fitness sharing, lexicase selection, nondominated sorting, and novelty search. In general, our results are consistent with previous work. Truncation and tournament selection are heavily exploitative with poor capacities for exploration, and novelty search was purely exploratory with no mechanism for exploitation. Nondominated sorting excelled at managing multiple, contradictory objectives, but did not exploit gradients well. Fitness sharing consistently performed poorly across diagnostics, neither exploiting nor exploring particularly well. The results for fitness sharing illustrate that the current set of diagnostics is incomplete and missing some aspects of problem-solving. Lexicase selection effectively balanced exploration with exploitation, performing reasonably well across all diagnostics. These results further illustrate the importance of diagnostics, as these selection schemes that are typically used for different kinds of problems can now be compared.

Now that I have shown that these diagnostics can provide insights into configuring a complex selection scheme (lexicase selection), and intuitive baseline results for several other common selection schemes, in **Chapter 7**, I propose using the diagnostics to analyze population structures. This chapter will leverage the selection scheme framework by adding a layer of complexity to the selection schemes analyzed in Chapter 6. Specifically, I will integrate island structures into tournament selection, truncation selection, and lexicase selection. Because I know that tournament selection and truncation selection are extremely effective at exploitation, I can measure the tradeoffs an island structure has for its ability to explore. Lexicase selection performs fairly well on all diagnostics but does not outperform all other selection schemes, now I can measure the impact island structures has on its exploitation and exploration abilities. Given that island structures are intended for diversity maintenance, I hypothesized that all three selection schemes would generally see a decrease in performance for exploitation-based diagnostics while improving on the diagnostics that require exploration. I believe the difference in performance will be ultimately dictated by the population structure being used and its configuration. In fact, we find that island structures decrease the exploitation abilities of all three selection schemes. Lexicase selection's exploration abilities are negatively affected by island structures, while both truncation and tournament selection see an increase in exploration abilities.

Part I

Formalizing selection schemes and demonstrating that subtleties alter performance

Chapter 2
Selection Scheme Framework

Authors: Jose Guadalupe Hernandez, Alexander Lalejini, and Charles Ofria

This chapter presents a theoretical framework for selection schemes within a generational evolutionary algorithm. We dismantle a selection scheme into three fundamental components: population structures, trait processing, and selectors. Population structures determine how solutions interact with one another during selection. Trait processing specifies how traits are transformed and used to identify parents. Selectors are the procedures that use the population structure and processed traits to actually identify a parent. Indeed, this framework proves to be useful to engineer robust selection schemes in the following chapters.

2.1 Introduction

Most evolutionary algorithms (EAs) make use of a selection scheme – a set of well-defined procedures that identify the set of candidate solutions to act as parents to produce the following generation. The purpose of a selection scheme is to choose parents that, in the long run, are most likely to lead to the discovery of an optimal solution. In simple selection schemes, this process is done by choosing "elite" solutions; that is, solutions with comparatively high performance or otherwise beneficial qualities (Crow and Kimura, 1979). However, more nuanced techniques exist that explicitly attempt to cross fitness valleys or promote the simultaneous exploration and exploitation of many regions of the search space. The consensus across different evolutionary algorithm practitioners is that selection schemes should favor high-quality solutions, but the question remains – how should we decompose a selection scheme into meaningful and comparable components?

Solutions in the population must be evaluated on some problem prior to being considered by a selection scheme. Evaluation sometimes entails processing an individual solution directly using a simple fitness function, but can also involve more complex analyses such as simulations in virtual environments. Sometimes, multiple measures of performance need to be taken (such as evaluating a genetic program on many different test cases) or even measures of non-performance characteristics (such as the size of the underlying representation, where size may be independent of performance). The traits that a candidate solution earns from its evaluation on the given problems provide useful information for identifying solutions that are exploring potentially promising regions of the search space. Indeed, selection schemes act on these trait characteristics with the goal of leading a population to an optimal (or nearly optimal) solution in the long term. In the case of problems with many conflicting objectives, the goal is often to cover the Pareto front as well as possible.

Prior to entering the evaluation phase during an evolutionary run, practitioners must determine the set of *metrics* that measure problem-solving success and other non-performance characteristics of interest. For example, problem-solving metrics could be test cases for a problem, or desired traits to have in a solution (such as low production cost, high reliability, or minimal resource requirements). Non-performance metrics, on the other hand, could be a solution's underlying representation size or measures for the solution's modularity, which may not be immediately beneficial, but may help with future evolvability. Once these metrics are determined, they can be used directly or passed as inputs to calculate the set of traits for the candidate solution. In practice, it is common for simple selection schemes to use a single value to represent a solution's quality, often an aggregate of all metric evaluations, especially if each metric evaluation is a performance measurement on a separate test case. This scenario is commonly found to be the case within genetic programming, whereas many other EA branches typically receive one metric evaluation for the instance of the problem they are attempting to solve. This reduction to a single value is not always the case, as more advanced selection schemes often incorporate more than one trait value (Helmuth et al., 2015). Because the evaluation phase has a major impact on a selection scheme's ability to reach optima, trait processing is a vital component that impacts a selection scheme's problem-solving abilities.

After the metrics are determined and traits are assigned to all candidate solutions, a selection scheme can begin identifying parent solutions. As previously mentioned, the simplest approach is to choose the top-performing solutions (truncation selection), but techniques take highly varied approaches, such as creating different kinds of competitions to identify parents or altering the criteria from one selection to the next. Tournament selection is an example of repeated competitions; in each round, a random subset of the population is placed in a tournament and the "elite" solution from this subset is selected as a parent. The population structure for tournament selection is typically *well-mixed*: all solutions in the population have the same opportunity to be included in each tournament. Alternative population structures are also used with various selection schemes. For example, it is possible to group solutions according to a trait such as age (Hornby, 2006), to separate them into fixed groups such as island models that facilitate allopatric speciation (Cohoon et al., 1987), or to spread them across space where individuals only compete with nearby neighbors such as cellular models (Tomassini, 2005). This example highlights the need for two additional components to break down a selection scheme: the procedures used to identify parents (selectors) and the mechanisms by which candidate solutions can interact (population structure).

2.2 Selection Scheme Components

Selection schemes can be difficult to characterize into components from either a written description (which tends to be presented holistically, focusing on the big picture) or even from software implementations (which typically intertwine multiple parts for the sake of efficiency). I propose a framework for selection schemes that accomplishes two important tasks: it formalizes the notation of what a selection scheme is, as well as the external factors that affect selection. More importantly, describing selection schemes with this new conceptual framework illustrates that compatible selection scheme components act as interchangeable parts.

Below are the descriptions of the three selection scheme components in detail, but first we define our notation. The notation used in this work is inspired by Blickle and Thiele (1995). Let P represent the population of N solutions, $\mathbb{J}$ represent the entire solution space, and j_i represent an individual solution from $\mathbb{J}$. For example, if our solutions are bitstrings of length 100, each j_i would be an individual bitstring, while $\mathbb{J}$ would be the collection of all 2^{100} possible length-100 bitstrings. This allows us to construct $P = \{j_1, ..., j_N\}$. Note, it is possible for two members of a population to be identical (i.e., j_i may be the same as j_k.)

2.2.1 Population structure

The concept of a population structure has long been used in biology to understand how organisms interact with one another, and how they interact with the abiotic world around them. In nature, organisms are inherently separated over space, but can also be subject to additional spatial constraints such as barriers, different environments, and different size scales. These constraints on biological interactions create a population structure, which can be mimicked within an EA to alter evolutionary dynamics.

All EAs possess a population structure that defines how candidate solutions in the population interact throughout evaluation, selection, and reproduction. Typically, EAs allow all solutions to interact with one another throughout the entire evolutionary search (Alba and Tomassini, 2002;

Sprave, 1999; Tomassini, 2005), but more complex structures exist that restrict the interactions between solutions. Island models are an early example of population structures where the candidate solutions are partitioned into separate islands. Age-layered population structure (ALPS) is another example, where populations are partitioned according to how long ago their lineage was introduced (Hornby, 2006). A more modern example is MAP-Elites, where candidate solutions are grouped based on non-fitness traits that are believed to have an indirect effect on fitness (Mouret and Clune, 2015). Ultimately, population structures provide another tool for selection schemes to find high-quality solutions.

In practice, grouping solutions prior to executing a selection scheme is a mechanism for diversity maintenance; common criteria used to construct population structures include genotypic (Ohira and Islam, 2020) or phenotypic (Hu et al., 2005) similarity, test cases (Chapter 3), age (Hornby, 2006), or simply random assignment (Tomassini, 2005). Individual candidate solutions can be separated by hard barriers (*e.g.*, different islands) or soft (*e.g.*, cellular models), and migration can be incorporated. A selection scheme must be supplied with the group of individuals it can use to identify parents, be this a whole population (*e.g.*, well-mixed), a fixed subset (*e.g.*, an island), or a dynamic subset (*e.g.* a local neighborhood on a grid).

Recall from earlier that the population is given by $P = \{j_1, ..., j_N\}$. As in Sprave (1999), let Π denote the complete set of interactions defined by the population structures with I being the total number of interaction sets in Π. This notation gives us $\Pi = \{\pi_1, ..., \pi_I\}$, where each π_i is an individual set of interactions. The complete set of interactions Π must be set and given to the selection scheme prior to a selection event occurring, where Π may be changing throughout an evolutionary run. More sophisticated examples to depict population structures exist, such as hypergraphs in (Sprave, 1999). We recognize the value in these more sophisticated representations, as they make particular analyses easy to conduct. However, we intentionally chose a simple and generic representation to describe population structure within our framework to simplify interoperability. Next, we provide some examples of well-known population structures using our notation.

First, we define Π for the simple case for a well-mixed structure:

$$\Pi = \{\pi\}$$

Given all solutions interact with one another, the set of interactions is just the original population.

As such, $\pi = \{j_1, ..., j_N\}$.

Next, we define Π for age-layered population structure (ALPS):

$$\Pi = \{\pi_1, ..., \pi_I\}$$

Each $\pi_i \in \Pi$ represents a set of solutions that are allowed to interact for a selection event, and all solutions fall within a specific age range. For this example, I is the number of age groupings generated within ALPS. Island models would have a similar formulation to ALPS, but where each π_i are fixed groups whose sizes are maintained over time.

The population structures help selection schemes leverage characteristics that each structure enforces, which can help with finding high-quality solutions. This component may often go unrecognized, as it is common in practice to have a selection scheme that implicitly uses a well-mixed structure without calling out this important design choice.

2.2.2 Trait processing

This component formalizes the protocols used to construct a single or set of traits that will later be used to select parents. Traditionally, a single fitness value is used to measure the quality of a solution. For genetic programming, this value is often obtained by aggregating performances across multiple fitness cases, while many other EAs solving an instance of a problem will receive only one performance value. More advanced methods for producing a single fitness value exist (Goldberg and Richardson, 1987; Lehman et al., 2008), while more complex selection schemes use more than one value to measure the qualities of a solution (Helmuth et al., 2015; Srinivas and Deb, 1994). The term fitness is often used in selection scheme literature, but I will refrain from using it here to avoid confusion with the biological definition of fitness (which is a measured value, not assigned) and to more comfortably discuss multi-trait selection schemes.

For example, if the problem at hand is to evolve a virtual robot that can walk as far as possible, evaluation may be a full simulation of the individual solution. In a simple selection scheme, trait processing might simply pull the distance walked from the evaluation and set that as the individual's fitness. Sometimes, a practitioner may want to optimize multiple traits along a Pareto front (distance walked, price to build the robot, reliability, *etc.*). At other times, traits not directly associated with overall quality may also be used. In such a case, trait processing might store not just the distance moved, but how often different actuators were used, the vertical height the robot was able to maintain, how many individual steps were taken, etc. Some of these subsidiary traits may be helpful in longer-term evolution; for example, rewarding vertical height might select for robots that can stand and balance properly. Each practitioner might have a different opinion about which specific traits will prove most important to evolving high-quality solutions.

Evaluations for the predetermined metrics must be calculated before traits can be constructed. It is common in practice for metric evaluations to consist of performances on a set of test cases for a given problem, but evaluations can also capture non-performance characteristics. Traits are then derived from the metric evaluations that measure solution characteristics on the given problem, and the final set of traits must be constructed before a selection scheme can identify parents. It is possible for multiple transformations of values to occur before the final set of traits is constructed. For example, the nondominated sorting genetic algorithm first generates a score based on nondominated front ranking, then that score is adjusted using fitness sharing (Srinivas and Deb, 1994). Once the traits are constructed, they are assigned to a candidate solution as a trait vector $\mathbf{t}$.

Let the set of metric functions be given by $M = \{m_1, ..., m_T\}$ for some problem, where T denotes the total number of metric functions. Each metric function m_i takes in a candidate solution as input and returns an evaluation as output. After evaluating all candidate solutions on the set of metrics, an evaluation matrix can be constructed. The metric evaluation matrix is given by,

$$E = \begin{pmatrix} m_1(j_1) & ... & m_T(j_1) \\ . & & . \\ . & ... & . \\ . & & . \\ m_1(j_N) & ... & m_T(j_N) \end{pmatrix}$$

Each row in E represents all the metric evaluations for a solution across all the metric functions in M. For example, the first row represents solution j_1's evaluations across all metrics in M. Let $\mathbf{e}_j$ represent an evaluation vector for some arbitrary candidate solution j in the population. The metric evaluation matrix is compressed to,

$$E = \begin{pmatrix} m_1(j_1) & ... & m_T(j_1) \\ . & & . \\ . & ... & . \\ . & & . \\ m_1(j_N) & ... & m_T(j_N) \end{pmatrix} = \begin{pmatrix} \mathbf{e}_1 \\ . \\ . \\ . \\ \mathbf{e}_N \end{pmatrix}$$

The matrix E can be thought of as the metric evaluation matrix for all candidate solutions in the population.

Let $\mathbb{T}$ represent the trait matrix that is produced after constructing the set of traits for each candidate solution from its evaluation vector,

$$\mathbb{T} = \begin{pmatrix} \mathbf{t}_1 \\ . \\ . \\ . \\ \mathbf{t}_N \end{pmatrix}$$

The elements in $\mathbb{T}$ represent an individual solutions trait vector. It is important to note that practitioners must describe how a solution's trait vector is constructed. For example, if we use tournament selection then only a single trait must be constructed. So, the trait vector with a single value for candidate solution j is given by,

$$\mathbf{t}_j = \sum_{e \in \mathbf{e}_j} e$$

If instead, we were to use lexicase selection, then we need to construct one trait per metric evaluation. So, the trait vector for candidate solution j is given by,

$$\mathbf{t}_j = \mathbf{e}_j$$

While the previous examples are simple, this component description helps practitioners better understand what candidate solutions are assigned after being evaluated on their specified metrics. This is crucial as it is important for practitioners to understand how a solution's quality is being measured. Ideally, the metrics would be well defined, such as stating where they came from (*e.g.*, benchmark suite, website, data, *etc.*).

2.2.3 Selector

Once the population structure is constructed and each candidate solution is assigned a trait vector, a selection scheme can use them to identify parents. Indeed, a set of parents must be produced by the end of a selection phase, but how are parents *actually* being selected? Describing the procedures used to identify parents is a crucial component of a selection scheme. We suggest that these procedures should be described algorithmically, so that it is easy to identify how this process is done. In the naive case, parents can be randomly chosen, but we would not expect progress to be made in this circumstance unless other forms of implicit selection are introduced.

The selector describes the set of procedures that result in the selection of candidate solutions to act as parents. It is common in practice to execute the selector multiple times, where the number of selector calls is dependent on the number of parents and offspring required, but some selectors may choose parents in groups. A selection scheme is provided both the population structure Π and the trait matrix $\mathbb{T}$ as input, where the goal of a good selector is to leverage these inputs to effectively traverse the search space.

There are numerous ways to select parent solutions, but what is the best way to do so? Clearly, selecting random solutions to serve as parents will not allow the population to exploit promising regions of the search space, so sophisticated procedures were developed to obtain better results. The key takeaway for a selector is that once the inputs I and $\mathbb{T}$ are constructed, practitioners must describe how the selector is leveraging both of the inputs.

2.3 Example: tournament selection

Here we present an example application of the selection scheme framework for a typical use of tournament selection.

2.3.1 Population structure

Tournament selection is most commonly used with a well-mixed population structure, so $\Pi = \{\pi\}$, where the set of interactions is just the original population. As such, $\pi = \{j_1, ..., j_N\}$.

2.3.2 Trait processing

Assuming the metric evaluation matrix E is constructed, candidate solution j's trait vector is defined by

$$\mathbf{t}_j = \sum_{e \in \mathbf{e}_j} e$$

For tournament selection, this trait vector must consist of only one single value.

2.3.3 Selector

Tournament selection conducts tournaments of a subset of k candidate solutions to identify an individual parent. These procedures can be found in Algorithm 2.1. A similar procedure can be used for most other common selection schemes.

```
1. Generate tournament of solutions from $\pi \in \Pi$:
    (a) $\{j_a, ..., j_z\} = random\_subset(\pi, k)$
2. Identify 'elite' solution $j^*$ from the tournament:
    (b) $j^* = max(\mathbf{t}_{j_a}, ..., \mathbf{t}_{j_z})$
3. Return $j^*$
```

Algorithm 2.1: Tournament selection selector.

Chapter 3
Random subsampling improves performance in lexicase selection

Authors: Jose Guadalupe Hernandez, Alexander Lalejini, Emily Dolson, and Charles Ofria

This chapter is adapted from (Hernandez et al., 2019), which appeared in the companion proceedings of the 2019 Genetic and Evolutionary Computation Conference.

In this work, we integrate random subsampling within lexicase selection to reduce the number of evaluations needed to find high-performing solutions. The selection scheme framework assisted in developing cohort lexicase and down-sampled lexicase, where we modified the trait processing component of lexicase selection. Cohort lexicase partitions the population and test cases, pairs the population and test case partitions, and runs standard lexicase to identify parents within each pairing. Down-sampled lexicase samples a set of test cases each generation. Both new lexicase selection variants reduce the test cases needed to identify an individual parent. We find that both lexicase variants can increase problem-solving success for evolving linear genetic programs to solve five different programming synthesis problems, where problem-solving success varied by subsampling rate.

3.1 Introduction

We often apply evolutionary computation to test-based problems where the quality of a candidate solution is assessed by evaluating it on a large set of test cases. For such problems, we must select parents (*i.e.*, genetic source material) for each generation based on how well individuals solve each test case. In many test-based problems, the space of possible test cases is either infinite or so large that it is not computationally feasible to evaluate a candidate solution on every possible test case. In the absence of extensive domain knowledge, it can be challenging to find an optimal test set size. Too small, and we risk overfitting. Too large, and the demand on computational resources will bring adaptive evolution to a crawl.

Lexicase selection is a more recent technique developed for genetic programming (GP) that has been demonstrated to be particularly effective for solving challenging test-based problems (Helmuth et al., 2015; Martínez et al., 2017; Spector, 2012). The lexicase algorithm chooses each parent for the next generation by sequentially applying test cases, in a random order. Only the best performers on each test case are kept until a single individual is identified. This sequential filtering approach is a departure from traditional parent-selection methods that calculate an absolute fitness metric by summing an individual's performance across all test cases. Because lexicase changes the ordering of test cases for every parent-selection event, individuals that perform well on different subsets of test cases can co-exist. This dynamic allows lexicase selection to maintain specialists on tests that the majority of the population fail, preserving potentially important genetic material (Dolson and Ofria, 2018; Helmuth et al., 2016a) and thus searching for a perfect solution from many directions at once.

The drawback of lexicase (and many other test-based selection schemes) is that assessing candidate solutions on a large set of test cases can be computationally expensive, especially if individual evaluations are costly. A simple speed-up might seem to be cutting down the number of evaluations by limiting the number of successive filtering steps taken during each lexicase selection event, shifting to a random selection if multiple solutions are still available (*e.g.*, truncated lexicase (Spector et al., 2018)). However, in practice, each candidate solution must still be evaluated on most test cases every generation.

We could trivially decrease the number of evaluations per generation by statically reducing the total number of test cases used during the evolutionary search. For example, a 50% reduction in test cases would allow us to run our search for about twice as many generations. However, simply reducing the total number of tests is more likely to result in prospective solutions overfitting the reduced test set. Reducing computational effort on test-based problems is a long-standing endeavour for GP (Gathercole and Ross, 1994). Many techniques have been proposed that dynamically subsample the set of tests (from a large pool) used for candidate solution assessment and selection (see Martínez et al. (2017) and Hmida et al. (2017) for recent reviews). Subsampling techniques have been employed to reduce computational effort in GP (Curry and Heywood, 2004; Gathercole and Ross, 1994) and to improve the generalizability of evolved programs (Gonçalves et al., 2012; Martínez et al., 2017). Can we apply test-case subsampling techniques to lexicase selection?

Here, we examine two lexicase selection variants that leverage random subsampling to reduce the number of evaluations per generation: *down-sampled lexicase* and *cohort lexicase*. Down-sampled lexicase selects parents based on a random subset of test cases each generation, guaranteeing that individuals are only evaluated against test cases in the subset. Cohort lexicase uses all test cases each generation, but divides both tests and individuals into cohorts, ensuring that each individual is evaluated against only a subset of tests. By reshuffling which test cases are experienced every generation, *lineages* will eventually encounter all test cases. We compare the results of different configurations of down-sampled and cohort lexicase across five program synthesis problems. Additionally, we compare the performance of our proposed lexicase variants to that of standard lexicase with a reduced number of total tests.

3.2 Lexicase Selection

Lexicase selection is a method for choosing a candidate solution from a population to use as a parent (*i.e.*, to provide genetic source material for a new individual in the next generation). Each such parent is selected individually, with replacement, such that individuals may be chosen multiple times. In lexicase, a large number of test cases are used as criteria for evaluation. Unlike many traditional parent-selection methods, lexicase does not aggregate performance across test cases to calculate a single fitness score. Instead, each time a parent is needed, test cases are successively applied in a random order, keeping only the most fit candidates on each. This process continues until the population is filtered down to either a single candidate or a set of equivalent candidates (at which point one is selected randomly). Because the ordering of test cases changes for every parent-selection event, individuals that perform well on different subsets of test cases are able to co-exist (Dolson and Ofria, 2018; Helmuth et al., 2016a). A more detailed description of lexicase selection can be found in (Helmuth et al., 2015; Spector, 2012).

Spector (Spector, 2012) initially proposed lexicase selection as a GP selection scheme for modal problems where qualitatively different modes of response are required for inputs from different regions of the problem domain. Subsequent work demonstrated lexicase selection's efficacy relative to traditional parent-selection algorithms on *uncompromising* problems where solutions must perform optimally over the entire space of possible test cases (Helmuth et al., 2015). Part of lexicase selection's success is attributed to its effectiveness at diversity maintenance; lexicase maintains specialists on test cases that the majority of the population fail, preserving potentially important genetic material (Dolson and Ofria, 2018; Helmuth et al., 2016a). For an analysis of lexicase selection in the context of ecological theory, see (Dolson and Ofria, 2018).

Several variants of lexicase selection have previously been proposed (Spector et al., 2018). We propose two new lexicase variants that relax the need to evaluate all candidate solutions against most test cases, thus allowing computational resources to be reallocated to additional search time, larger population sizes, *et cetera.*

3.3 Down-sampled Lexicase Selection

In each generation of standard lexicase selection, every test in the test case set is available as evaluation criteria for selection events; thus, all individuals must be evaluated against *most* test cases in each generation. Assuming we can store and reuse previously computed performances for each repeated application of a test case during parent selection events, lexicase selection's worst-case number of per-generation evaluations is equal to the size of the test case set multiplied by the population size (*i.e.*, every member of the population is evaluated against every test case once).

Down-sampled lexicase applies the random subsampling technique (Gonçalves et al., 2012) to lexicase selection. Each generation, down-sampled lexicase selects a random subset of the test cases to use for all selection events, guaranteeing that unselected test cases are not evaluated at all. Here, we refer to our 'down-sample factor' (the subsample rate) as D. For example, $D = 10$ implies a tenfold subsample rate (*i.e.*, each generation, we use $\frac{1}{10}$ of the total test case set to evaluate individuals). This down-sampling divides the worst-case number of evaluations performed each generation by D, allowing us to run our evolutionary search for more generations (or with a larger population size) than standard lexicase selection. Here, we exclusively apply random subsampling to every generation; however, as discussed by Gonçalves et al. (2012), we could also vary the number of generations at which we apply random subsampling.

Why is down-sampling the test case set preferable to simply reducing the number of test cases? In down-sampled lexicase selection, lineages are likely to be tested against a large portion of the full test set over several generations. Each generation, a candidate solution will encounter a proportion of test cases equal to $\frac{1}{D}$; thus, $1 - \frac{1}{D}$ gives the proportion of test cases *not* encountered by a candidate solution in a given generation. The *expected* proportion of test cases not encountered by a *lineage* after G generations is $(\frac{D-1}{D})^G$. To calculate the expected number of generations for a lineage to be evaluated against proportion T of the test cases for a known down-sampling rate $(\frac{1}{D})$, we can solve for G in Equation 3.1.

$$G = \frac{log(1-T)}{log(D-1) - log(D)} \tag{3.1}$$

Note that a lineage will always encounter proportion $T \leq \frac{1}{D}$ in a single generation, and T asymptotically approaches 1.0 as the number of generations increases.

3.4 Cohort Lexicase Selection

Cohort lexicase selection makes use of the full test case set each generation but ensures that each prospective solution is evaluated against only a subset of them. Every generation, cohort lexicase randomly partitions both the population and test case set into K equally-sized sub-groups (cohorts). Each of the K candidate solution cohorts is then paired with a test case cohort, and each candidate solution in a cohort is evaluated against all test cases in the associated test case cohort. This means that the number of evaluations performed each generation (relative to standard lexicase selection) is divided by K. Candidate solutions only compete within their cohort, and within-cohort competition is arbitrated by the test cases in the associated cohort of tests. In this way, cohorts impose a sort of island model (Wright, 1943) on standard lexicase selection where each island's membership (candidate solutions) and environment (test cases) is transient, and randomized every generation.

Our formulation of cohort lexicase follows the same expectations as down-sampled lexicase for the number of generations before a lineage is expected to encounter proportion T test cases (given by Equation 3.1). Cohort lexicase's K and down-sampled lexicase's D create an equivalent down-sampling rate. Note, however, in our implementation of cohort lexicase, tests are not repeated across cohorts; though, there is no reason why they could not be repeated.

3.5 Methods

To test the utility of down-sampled and cohort lexicase selection, we used both selection schemes to evolve linear genetic programs to solve five test-based problems from the program synthesis benchmark suite (Helmuth and Spector, 2015): Small or Large, For Loop Index, Compare String Lengths, Median, and Smallest. A description of our GP system (including source code) can be found in supplemental material (Lalejini and Hernandez, 2019).

3.5.1 Program Synthesis Problems

Problems in the general program synthesis benchmark suite were selected from sources for introductory computer science programming problems; while not particularly challenging for experienced human programmers, they can be challenging for current GP systems (Forstenlechner et al., 2018; Helmuth and Spector, 2015). These benchmarks have been used to compare lexicase selection against other, more traditional selection schemes (Helmuth and Spector, 2015). Previous studies (using PushGP (Helmuth and Spector, 2015) and G3P (Forstenlechner et al., 2018)) have shown standard lexicase selection to be capable of solving the five problems used in this work, making them good choices for evaluating random test subsampling in the context of lexicase selection.

Each problem is defined by a set of test cases in which programs are given input data and is scored on how well their output matches the correct output (assigning scores on a gradient or pass-fail basis as appropriate). During an evaluation, the total number of steps (instructions) a problem could execute varied by problem.

During evolution, programs were assessed using a training set of test cases, which defined the selection criteria used for lexicase selection. To qualify as a solution, a program needed to perfectly pass all test cases in a separate testing set (withheld generalization examples) in addition to passing all tests in the training set used during evaluation. For all problems, we used the same training and testing sets (100 training cases and 1,000 testing cases) and the same input constraints as in (Helmuth and Spector, 2015). The exact training and testing sets used can also be found in our supplemental material (Lalejini and Hernandez, 2019).

For a more detailed description of the five benchmark problems used here (Small or Large, For Loop Index, Compare String Lengths, Median, and Smallest), see (Helmuth and Spector, 2015) or our supplemental material (Lalejini and Hernandez, 2019). For each problem, we added problem-specific instructions (see (Lalejini and Hernandez, 2019)) to our GP instruction set to allow programs to load test case inputs into memory and submit output.

3.5.2 Experimental Design

We evolved populations of 1,000 programs under a range of subsampling levels (*i.e.*, the percent of the training set used to assess candidate solutions) using both down-sampled and cohort lexicase: 5%, 10%, 25%, 50%, and 100% (no reduction). Additionally, we evolved programs using standard lexicase selection (no subsampling) with 5%, 10%, 25%, 50%, and 100% (no reduction) of the training set; when reducing the training set for standard lexicase selection runs, we randomly selected the appropriate percentage of test cases from the full training set (*e.g.*, 5 of the 100 total test cases when using 5% of the training set), and the reduced training set remained static for the duration of evolutionary search.

We ran 100 replicates of all conditions, each for a fixed budget of 30,000,000 evaluations (*i.e.*, 300 generations when using the full training set). Conditions where we subsampled or reduced the training set ran for more generations than conditions using the full training set (5%: 6,000 generations; 10% 3,000 generations; 25%: 1,200 generations; 50%: 600 generations). For each problem and selection condition, we compared the problem-solving success rates (*i.e.*, the number of runs in which a perfect solution evolved) of using fewer training cases (via cohorts, down-sampling, or static reduction) versus using the full training set during selection (Fisher's exact test with a significance level of 0.05 and a Holm-Bonferonni correction for multiple comparisons). All statistical analyses were performed using the R Statistical Computing Platform (R Core Team, 2016). The source code for our analyses and data visualizations can be found in our supplemental material (Lalejini and Hernandez, 2019).

3.6 Results and Discussion

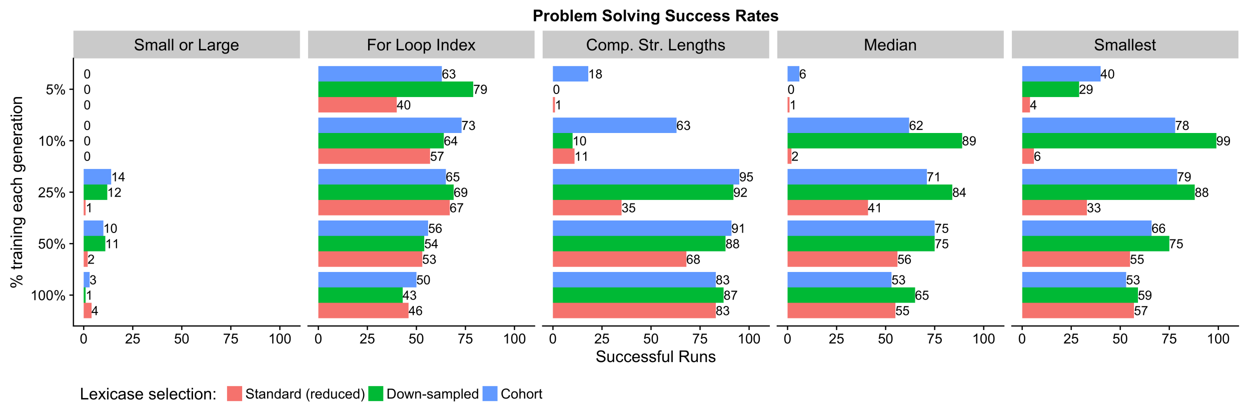

Figure 3.1: Problem-solving success rates (*i.e.*, the number of runs in which a perfect solution evolved) for each program synthesis problem. Note that, here, all conditions using 100% of the training set (regardless of lexicase variant) are qualitatively identical conditions.

Figure 3.1 shows the problem-solving success for all experimental conditions across all five problems after a fixed number of test case evaluations; see our supplemental material (Lalejini and Hernandez, 2019) for more detailed statistical analyses. With the exception of the For Loop Index problem, reducing the size of the training set for standard lexicase selection (resulting in more generations of evolution) did not improve (by a statistically significant amount) problem-solving success. Indeed, on the Compare String Lengths, Median, and Smallest problems, reducing the training set beyond a critical threshold (which varied by problem) when using standard lexicase selection significantly reduced problem-solving success relative to using the full test case set (*e.g.*, Compare String Lengths, 50% training: $p < 0.021$; Median, 10% training: $p < 3.68e\text{-}10$; Smallest, 25% training: $p < 0.003$). These reduced success rates are likely due to overfitting: we sufficiently reduced the training set such that it does not adequately represent the full space of test cases, and as a result, evolved programs fail to generalize. On the For Loop Index problem, using standard lexicase with only 25% of the full training set has a significantly *higher* success rate than when using the full training set ($p = 0.017$); in this case, reducing the size of the training set to rapidly progress through more generations pays off, which suggests that the full training set for this problem is unnecessarily large to thoroughly assess candidate solutions.

Multiple configurations of down-sampled lexicase significantly improved problem-solving success relative to standard lexicase across all problems except Compare String Lengths where improvements are not statistically significant (*e.g.*, Small or Large, 50% training: $p < 0.015$; For Loop Index, 25% training: $p < 0.002$; Median, 25% training: $p < 0.007$; Smallest, 50% training: $p < 0.024$). Similarly, at least one configuration of cohort lexicase significantly improved problem-solving success relative to standard lexicase across all problems (*e.g.*, Small or Large, 25% training: $p < 0.034$; For Loop Index, 10% training: $p < 0.006$; Compare String Lengths, 25% training: $p < 0.023$; Median, 50% training: $p < 0.006$; Smallest, 25% training: $p < 0.001$). The particular configurations of down-sampled and cohort lexicase that work best depend on the problem. Neither cohort or down-sampled lexicase consistently outperformed the other on any of the five problems.

These results suggest that: (1) random subsampling can be used to improve the problem-solving performance of lexicase selection, and (2) both cohort and down-sampled lexicase are successful approaches for applying random subsampling to standard lexicase.

3.7 Conclusion

We presented two extensions of the lexicase parent selection algorithm that incorporate random subsampling techniques: down-sampled lexicase and cohort lexicase. Using these techniques, we confirm that random subsampling can be successfully applied to lexicase selection, allowing the evolutionary search to more rapidly progress through generations and improving problem-solving success rates. Our experimental results suggest that the best configuration of down-sampled and cohort lexicase depends on the problem. Future studies will tease apart how different levels of subsampling impact lexicase selection (*e.g.*, diversity maintenance).

Chapter 4
Characterizing the Effects of Random Subsampling on Lexicase Selection

Authors: Austin J. Ferguson, Jose Guadalupe Hernandez, Daniel Junghans, Alexander Lalejini, Emily Dolson, and Charles Ofria

This chapter is adapted from (Ferguson et al., 2020), which appeared in Genetic Programming Theory and Practice XVII.

In this work, we investigate why differences occurred between standard lexicase selection and its subsampling variants. We conducted four experiments to characterize the effects of subsampling within both cohort and down-sampled lexicase selection, where populations of linear genetic programs were evolved to solve four program synthesis problems. We make three key findings: (1) both cohort and down-sampled lexicase do not out-perform standard lexicase selection with a fixed generational budget, (2) both cohort and down-sampled lexicase require fewer test case evaluations than standard lexicase selection to produce solutions on all four problems, and (3) subsampling degrades lexicase selection's ability to maintain specialists within the population.

4.1 Introduction

Evolutionary computation is often used to solve complex, multi-faceted problems where the quality of a candidate solution is measured according to its performance on a large set of test cases. For these test-based problems, we must somehow meld performances across many test cases to select individuals to serve as parents for the next generation. In many test-based problems, we cannot exhaustively evaluate a candidate solution over the entire space of possible test cases. As a result, it can be challenging to balance the trade-off between using a large enough test set to thoroughly evaluate candidate solutions while keeping the test set small enough to preserve computational resources and rapidly progress through generations.

Lexicase selection is a relatively new parent-selection algorithm developed for genetic programming (GP) and has been demonstrated as an effective tool for solving difficult test-based problems (Helmuth and Spector, 2015; Helmuth et al., 2015; Spector, 2012). Many traditional selection strategies for solving test-based problems score potential solutions by aggregating their fitness across all test cases. The lexicase algorithm, however, chooses each parent for the next generation by sequentially applying test cases in a random order, keeping only the best performers on each test case until the population has been winnowed to a single individual. Because the ordering of test cases changes for every parent selection event, individuals that perform well on different subsets of test cases are able to co-exist (Dolson and Ofria, 2018; Helmuth et al., 2016a).

The drawback of many test-based selection schemes, including lexicase, is that assessing individuals using a large set of test cases can be computationally expensive; this drawback is exacerbated when tests are costly to perform (*e.g.*, robotics simulations). Using a large number of test cases constrains the number of generations we are able to run an evolutionary search. Using too few test cases, however, may fail to accurately represent the problem domain and lead to overfitting. To combat this, many techniques dynamically subsample test cases (from a large pool representative of the problem domain) for candidate solution evaluation and selection (see (Hmida et al., 2017; Martínez et al., 2017) for recent reviews). Indeed, Subsampling has been used to reduce computational effort in GP (Curry and Heywood, 2004; Gathercole and Ross, 1994) and to improve the generalizability of evolved programs (Gonçalves et al., 2012; Martínez et al., 2017).

In this chapter, we characterize the effects of random subsampling on the lexicase parent-selection algorithm. Previous work has shown that lexicase selection performs well when combined with random subsampling. Moore and Stanton applied random subsampling to lexicase selection in the context of an evolutionary robotics problem because evaluating robot controllers on test cases (simulation environments) was too costly to permit exhaustive assessments (Moore and Stanton, 2017, 2018, 2019). In Chapter 3, we proposed down-sampled and cohort lexicase selection, two variants of standard lexicase that employ random subsampling to reduce the number of per-generation evaluations required by lexicase selection. We demonstrated that both down-sampled and cohort lexicase could yield higher problem-solving success than standard lexicase on a fixed evaluation budget in the context of program synthesis (Chapter 3).

Here, we explore *why* random subsampling can improve lexicase selection's problem-solving success. Additionally, we characterize the effect of subsampling on diversity and specialist maintenance, both of which have been shown to be important factors behind lexicase selection's efficacy (Dolson and Ofria, 2018; Helmuth et al., 2016a, 2019; Moore and Stanton, 2018). We show that the improvement in problem-solving success gained from subsampling is due to its facilitation of *deeper* evolutionary searches (*i.e.*, consisting of more generations relative to standard lexicase) given a fixed evaluation budget. Moreover, we show that both down-sampled and cohort lexicase find solutions with less computational effort than standard lexicase. While we predicted that subsampling would degrade diversity, we find no evidence for systematic degradation of phenotypic diversity. However, as the level of subsampling increases, cohort lexicase generates and maintains more phylogenetic diversity than down-sampled lexicase. As expected, we find that random subsampling degrades specialist preservation relative to standard lexicase. Our phenotypic diversity results seem to contradict our specialist preservation findings; this could be because of the particular problems we are using or because of our choice of time to measure phenotypic diversity (at the time a solution was found). Future work will continue investigating how subsampling affects diversity maintenance in an expanded problem domain and with more fine-grained data collection and analysis.

4.2 Lexicase Selection

Since its conception, lexicase selection has been successfully applied in the field of genetic programming. Such applications include program synthesis (Helmuth and Spector, 2015) and regression (La Cava et al., 2016). Lexicase selection has also been in other areas such as evolutionary robotics (Moore and Stanton, 2017), genetic algorithms (Metevier et al., 2019), and learning classifier systems (Aenugu and Spector, 2019). See Spector (2012), Helmuth et al. (2015), and Section 3.2 for a more detailed description of lexicase selection.

4.2.1 Applying Subsampling to Lexicase Selection

Several variants of lexicase selection (and lexicase-inspired selection algorithms) exist, such as ϵ-lexicase, truncated lexicase, batch-tournament, batch-lexicase, down-sampled lexicase, and cohort lexicase (Aenugu and Spector, 2019; De Melo et al., 2019; Spector et al., 2018). Here, we investigate down-sampled and cohort lexicase, both of which leverage random subsampling to reduce the number of per-generation evaluations required for lexicase selection. A more detailed description of down-sampled lexicase (Section 3.3) and cohort lexicase (Section 3.4) can be found in Chapter 3.

4.3 Methods

We conducted a series of experiments to characterize the effects of applying random subsampling to lexicase selection. In all evolution experiments, we evolved populations of linear genetic programs to solve four program synthesis problems. Using this setup, we replicated previous results (Chapter 3), tested the effect of the additional generations afforded by subsampling, and investigated how different types of subsampling affect the computational effort expended to solve problems. Additionally, we analyzed how these subsampling techniques affect both population diversity and specialist maintenance.

4.3.1 Evolutionary System

For each of our evolution experiments, we evolved populations of 1,000 linear genetic programs on four program synthesis problems (each described in detail in Section 4.3.2). Our linear-GP representation used:

- an instruction set that includes arithmetic, memory management, flow-control, and additional problem-specific instructions
- memory accessed with binary tags (Lalejini and Ofria, 2019)
- modules referenced via binary tags (Lalejini and Ofria, 2018; Spector et al., 2011)

A more detailed description of our GP system (including source code) can be found in the supplemental material (Ferguson, 2020).

We propagated programs asexually, subjecting offspring to mutations. Single-instruction insertions, deletions, and substitutions were applied, each at a per-instruction rate of 0.005. Modules were duplicated and deleted at a per-module rate of 0.05. We also applied 'slip' mutations (Lalejini et al., 2017), which have the possibility of duplicating or deleting sequences of instructions, at a per-program rate of 0.05. Program-tags were mutated at a per-bit rate of 0.001. The run-termination criteria varied per experiment and are included in each experiment description.

4.3.2 Program Synthesis Problems

For all evolution experiments, we evolved programs to solve problems from the general program synthesis benchmark suite (Helmuth and Spector, 2015). To test our hypotheses, we needed a set of problems known to be challenging but not impossible for GP systems to solve. The general program synthesis benchmark suite comprises introductory-level computer science programming questions, many of which have been solved using lexicase selection (Forstenlechner et al., 2018; Helmuth and Spector, 2015). We used the following four program synthesis problems in our experiments: *Smallest*, *Median*, *For Loop Index*, and *Grade*. A description of each problem is given below:

Smallest: Programs are given four integer inputs ($-100 \leq input_i \leq 100$) and must output the smallest value. We measured program performance on a pass-fail basis. We limited program length to a maximum of 64 instructions and also limited the maximum number of instruction-execution steps to 64.

Median: Programs are given three integer inputs ($-100 \leq input_i \leq 100$) and must output the median value. We measured program performance against test cases on a pass-fail basis. We limited program length to 64 instructions and also limited the maximum number of instruction-execution steps to 64.

For Loop Index: Programs receive three integer inputs *start* ($-500 \leq start \leq 500$), *end* ($-500 \leq end \leq 500$), ($start < end$), and *step* ($1 \leq step \leq 10$). Programs must output the following sequence:

$$n_0 = start$$
$$n_i = n_{i-1} + step$$

for each $n_i < end$. We limited program length to a maximum of 128 instructions and also limited the maximum number of instruction-execution steps to 256. Program performance against a test case was measured on a gradient, using the Levenshtein distance between the program's output and the correct output sequence.

Grade: Programs receive five integers in the range $[0, 100]$ as input: A, B, C, D, and *score*. A, B, C, and D define the minimum score needed to receive that letter grade. These are specified such that $A > B > C > D$ (*i.e.*, they are monotonically decreasing and unique). The program must read in these thresholds and return the appropriate letter grade for the given *score*, or F if $score < D$. We limited program length to a maximum of 64 instructions and also limited programs' maximum instruction-execution steps to 64. On each test, we evaluated programs on a pass-fail basis.

For these experiments, the *Smallest*, *Median*, and *For Loop Index* problems have an associated training set of 100 test cases, and a separate validation set of 1,000 test cases (withheld during fitness evaluations). We used 200 training cases and 2,000 validation cases for the *Grade* problem. A program had to solve all test cases in both the training and validation sets to be considered a "perfect" solution. All training and validation sets can be found in the supplemental material (Ferguson, 2020).

4.3.3 Experimental Design

We conducted five experiments: (1) we replicated a previous experiment (Chapter 3) to evaluate subsampling's effect on lexicase selection's problem-solving success; (2) we tested whether or not subsampling improves problem-solving success because it facilitates deeper evolutionary searches; (3) we evaluated whether subsampling can reduce the computational effort expended by lexicase selection to solve problems; (4) we tested the effect of random subsampling on lexicase selection, comparing the diversity maintenance of standard, down-sampled, and cohort lexicase; (5) we compared each of standard, down-sampled, and cohort lexicase's capacity to maintain specialist candidate solutions (*i.e.*, programs with low aggregate fitness that solve test cases that the majority of the population fails).

Does subsampling improve lexicase selection's problem-solving success given a fixed computation budget?

First, we replicated the experiment conducted in Chapter 3 where both down-sampled and cohort lexicase improved problem-solving success relative to standard lexicase selection. To evaluate whether subsampling improves lexicase's problem-solving success, we evolved programs using down-sampled, cohort, and standard lexicase selection to solve each of the four program synthesis problems (described in Section 4.3.2). While the sets of program synthesis problems are not identical, the main difference between the two experiments is that our previous work included a test case that was designed to minimize program size of candidate solutions that solved all normal test cases; this minimizing test case was discarded for all experiments in this work. For a control, we also tested *reduced lexicase*: standard lexicase performed on a statically reduced training set that was randomly sampled at the beginning of the run. Reduced lexicase is similar to down-sampled lexicase, with the exception that test cases remain constant throughout the evolutionary search and are not sampled every generation.

All three of these lexicase variants were tested at five subsampling levels: 100% (identical to standard lexicase), 50%, 25%, 10% and, 5% ($D = 1, 2, 4, 10$, and 20, respectively). For standard lexicase and each variant, we limited each instance to a maximum computation budget of 30,000,000 evaluations[1]. Thus, standard lexicase ran for 300 generations, and the subsampled variants ran for 300, 600, 1,200, 3,000, and 6,000 generations, respectively. We compared the problem-solving

[1]Evaluating a single program on a single test case is one test case evaluation.

success (*i.e.*, the number of replicates that produced a perfect solution) of each variant to standard lexicase. For each problem, we ran 50 replicates (each with a unique random seed) of each subsampled configuration, and 250 replicates (each with a unique random seed) of standard lexicase (50 replicates for each subsampling level).

Does subsampling improve lexicase selection's problem-solving success because it facilitates deeper searches?

Both down-sampled and cohort lexicase perform fewer test case evaluations per generation than standard lexicase, allowing us to run evolutionary searches for more generations given a fixed computation budget (*i.e.*, a fixed number of total test case evaluations). We expected that subsampling improves lexicase's problem-solving success because it enables deeper searches. To test this hypothesis, we repeated the performance experiment (described previously in Section 4.3.3), except we evolved *all* populations (regardless of selection scheme and subsampling level) for 300 generations. We compared the number of successful replicates from each of down-sampled, cohort, and standard lexicase. If down-sampled and cohort lexicase lose their performance edge over standard lexicase, the distinction must come from the time after the 300 generation limit that they would have continued evolving. This finding would suggest that subsampling's improved problem-solving success results from its facilitation of deeper evolutionary searches.

Does random subsampling reduce the computational effort required to solve problems with lexicase selection?

Our previous work (Chapter 3) shows that subsampling can improve lexicase selection's problem-solving success given a fixed computational budget. Here, we are interested in whether or not subsampling reduces the total computational effort required to find solutions; that is, do down-sampled and cohort lexicase generally find solutions using fewer total evaluations than standard lexicase selection? We evolved programs on the four program synthesis problems described previously (Section 4.3.2) using down-sampled, cohort, and standard lexicase (at a 10% subsampling level for down-sampled and cohort lexicase). For each condition, we ran 50 replicate populations. Because we wanted to compare how much computational effort it generally took for a particular selection scheme to solve a problem, we only used data from the first 25 replicates of each condition to solve the problem (*i.e.*, the 25 replicates per condition that used the least computational effort). We also included truncated lexicase (Spector et al., 2018), another lexicase selection variant that works to reduce the rigidness in lexicase selection by limiting the number of test cases used in a selection event before a candidate solution is selected. Truncated lexicase also has the potential to reduce the computational effort needed to find solutions. For our truncated lexicase condition, we used a truncation level equal to 10% of the training set.

Does subsampling degrade lexicase selection's diversity maintenance?

Part of lexicase selection's success is known to be the result of its effectiveness at diversity maintenance (Dolson and Ofria, 2018; Helmuth et al., 2016a; Moore and Stanton, 2018). Subsampling, however, is likely to degrade diversity maintenance because it both reduces the total number of niches available each generation (*i.e.*, there are fewer possible orderings of test cases) *and* decreases niche stability from generation to generation (*i.e.*, the set of possible test case permutations changes every generation). Thus, we expected populations evolved using down-sampled and cohort lexicase selection to have lower overall diversity and more frequent selective sweeps (coalescence events) than those evolved with standard lexicase selection. Additionally, cohort lexicase inherently buffers populations against selective sweeps, slowing down the rate at which a lineage can take over a population by limiting competition each generation to within cohorts. As such, we expected cohort lexicase to have fewer selective sweeps (and thus more phylogenetic diversity) than down-sampled lexicase.

To test our hypotheses, we replicated the experiment in Section 4.3.3, running both subsampling lexicase variants (at a range of subsampling levels) and standard lexicase for 30,000,000 total evaluations. In these runs, we collected data on genotypic, phenotypic, and phylogenetic diversity. We measured genotypic and phenotypic diversity with the Shannon diversity index. To assess phylogenetic diversity, we used a suite of phylogenetic diversity metrics (see Dolson et al. (2018) for a review). After all replicates terminated, we analyzed the results of each of these diversity measures *at the time solutions were found.*[2] Within each subsampling level, we compared cohort,

[2]Choosing when to measure diversity in evolutionary computation is an interesting problem. In evolutionary computation, diversity maintenance is often viewed as a mechanism to avoid premature convergence on suboptimal solutions. If our goal is to compare how well different selection schemes maintain diversity, *when* should we measure diversity? Measuring diversity *after* a global solution is found is not particularly meaningful, as finding the solution often causes the population to converge, decreasing diversity. We measured diversity at the time the solution is found to mitigate this problem. However, this solution only partially addresses the underlying problem: the process of evolution often involves many selective sweeps and subsequent divergences and we cannot know where in this cycle our measurements occurred.

Parameter	Values
Population size	10, 20, and 100
# test cases	10, 20
Generalist pass rate on non-focal tests	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0

Table 4.1: Generated population configurations. We generated 100 populations for all combinations of the parameters given in this table.

down-sampled, and standard lexicase selection.

Does subsampling reduce lexicase selection's capacity to maintain *specialists*?

Recent work by Helmuth et al. (2019) demonstrates lexicase's tendency to select specialist individuals (*i.e.*, individuals that have a low aggregate fitness but perform well on a subset of tests that the majority of the population fails). Helmuth *et al.* found that lexicase's ability to select specialists is a major driver behind its problem-solving success. Just as we expected subsampling to degrade lexicase selection's diversity maintenance, we also expected subsampling to inhibit specialist maintenance. Because specialists perform well on a few test cases (and potentially poorly on the rest), a specialist's likelihood of being selected by lexicase selection is reduced if any of the test cases it passes are not sampled. Thus, we hypothesized that both down-sampled and cohort lexicase reduce lexicase selection's capacity to maintain specialist individuals.

To test our hypothesis, we investigated the extreme case of populations with a single specialist. We generated hypothetical populations, each containing a 'specialist' and many 'generalists'. In each generated population, the specialist individual was able to solve only one focal test case, and none of the generalists were allowed to solve the focal test case. We varied the probability at which generalists could solve each non-focal test case, ranging from 0.1 to 1.0 (where all generalists solved all non-focal test cases). We also varied the population size and the total number of test cases. Table 4.1 shows all parameter values used in this experiment. We generated 100 populations for each combination of these parameters.

For each population, we calculated the probability of each candidate solution being selected *at least once* to be a parent in the next generation under standard, down-sampled, and cohort lexicase selection. For standard lexicase selection, we calculated exact probabilities: we enumerated all possible orderings of test cases, counting the number of enumerations where each candidate solution is selected. This is intractable for the subsampled lexicase variants, so we took a sampling approach. To approximate the selection probability in the lexicase variants, we randomly subsampled the population according to the selection scheme being tested. After subsampling, down-sampled lexicase is equivalent to standard lexicase with fewer test cases, while cohort lexicase is equivalent to standard lexicase conducted separately on each cohort. Thus, we calculated the selection probabilities for each candidate solution with that particular random subsampling. This process was repeated 100,000 times to approximate the true selection probabilities under down-sampled and cohort lexicase. These calculations allowed us to compare the specialist's selection probability across configurations.

4.3.4 Statistical Analyses

All statistics were calculated using the R statistical computing language v3.6.0 R Core Team (2019), and all figures in this work were created using the ggplot2 R package Wickham (2016a). We compared problem-solving success rates among different independent conditions using Fisher's exact tests, and we corrected for multiple comparisons using the Holm-Bonferroni method where appropriate. For measures of computational effort and diversity, we performed a Kruskal-Wallis test to look for statistically significant differences among independent conditions. For comparisons in which the Kruskal-Wallis test was significant (significance level of 0.05), we performed a post-hoc Mann-Whitney test between relevant conditions (with a Holm-Bonferonni correction for multiple comparisons where appropriate). Statistical analyses for the specialist experiment also used a Kruskal-Wallis test, but swapped the Mann-Whitney test for a Wilcoxon test because the data were paired. Analysis and visualization scripts can all be found in the supplemental material Ferguson (2020).

4.4 Results and Discussion

4.4.1 Subsampling improves lexicase selection's problem-solving success

Figure 4.1 shows the fraction of replicates where a perfect solution evolved within 30,000,000 evaluations under each of down-sampled, cohort, reduced, and standard lexicase selection. For each program synthesis problem, we conducted a Fisher's exact test (0.05 significance level) between the 250 standard lexicase replicates and the 50 subsampled replicates of each experimental condition; we corrected for multiple comparisons using the Holm-Bonferonni method.

Our data are largely consistent with previous work in Chapter 3. For three of the four problems (Smallest, Median, and Grade), statically reducing the training set beyond a critical threshold significantly decreased problem-solving success. For example, at 5% and 10% subsampling levels,

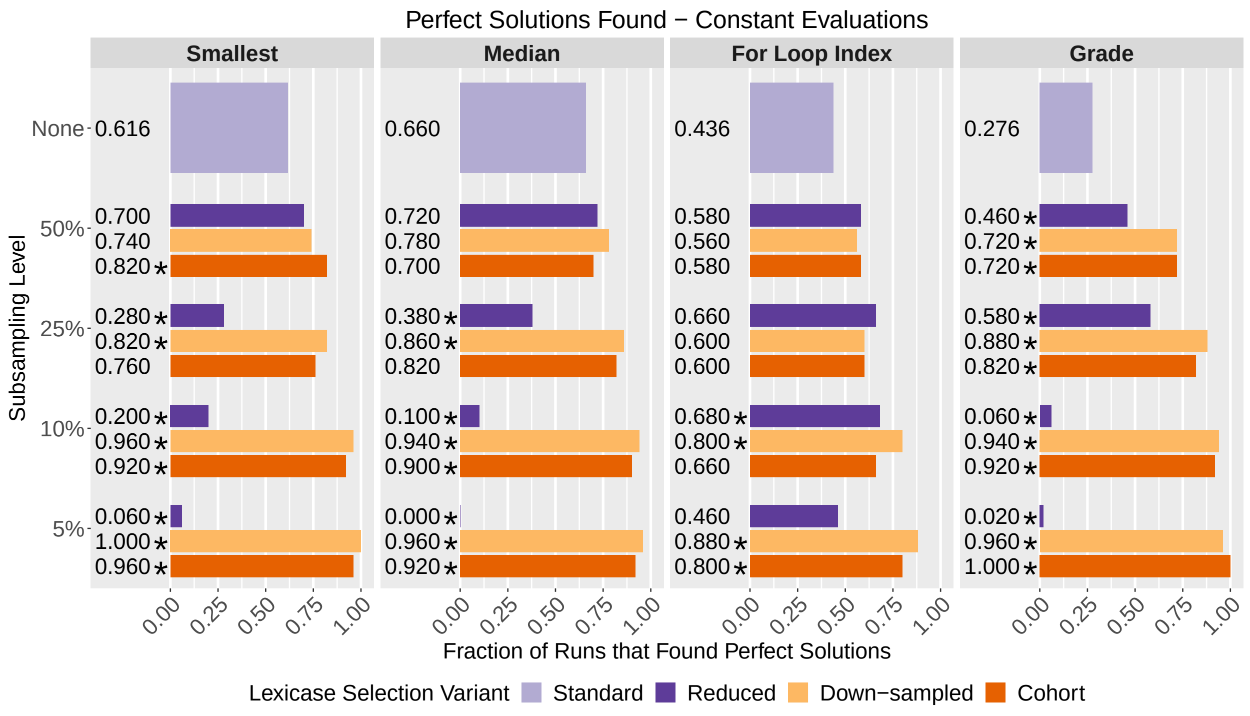

Figure 4.1: Problem-solving success after 30,000,000 evaluations. Bars show the fraction of replicates that found a perfect solution. An asterisk (*) to the left of a bar denotes a significant difference compared to the standard lexicase results (using a Holm-Bonferroni correction for multiple comparisons). Results for standard lexicase (light purple) consist of 250 replicates per problem, while results for reduced lexicase (dark purple), down-sampled lexicase (yellow), and cohort lexicase (orange) consist of 50 replicates for each configuration.

reduced lexicase performs significantly worse than standard lexicase in each of the Smallest, Median, and Grade problems. Reduced lexicase rarely outperformed standard lexicase, only doing so in three cases: Grade at 25% and 50% subsampling, and For Loop Index at 10% subsampling. Statically reducing the size of the training set did not inhibit our capacity to solve the For Loop Index problem; we suspect this is because the training set (100 test cases) is much larger than necessary. The same trend is true for 50%- and 25%-reduced lexicase on the Grade problem.

Both down-sampled and cohort lexicase performed significantly better than standard lexicase on at least one subsampling level for every problem. Specifically, down-sampled lexicase significantly outperformed standard lexicase on all problems at the 5% and 10% subsampling levels, while cohort lexicase also outperformed standard lexicase at 5% and 10% subsampling on all problems except For Loop Index at the 10% subsampling level. Neither down-sampled nor cohort lexicase performed significantly worse than standard lexicase in any experimental configuration.

These results achieved better performance on more extreme subsampling levels than in Chapter 3; this is because we removed all selection pressure to reduce program size. In this previous work, we included a single test case that favored small programs that only took effect when a program solved all other test cases *it was evaluated against*. At high subsampling levels (*e.g.*, 5%), it is easy for programs that do not generalize well to prematurely trigger this size-minimization test case, which negatively impacted problem-solving success rates.

These results support our previous claim that subsampling can improve lexicase selection's problem-solving success. Although there is evidence that subsampling can improve solution rates, a different approach is needed to tease apart *why* this difference exists, or how down-sampled and cohort lexicase actually differ.

4.4.2 Deeper evolutionary searches contribute to subsampling's success

Figure 4.2 shows the fraction of replicates where a perfect solution evolved after 300 generations under each of down-sampled, cohort, and standard lexicase selection. After 300 generations, conditions with aggressive subsampling (*e.g.*, 5%) have made fewer total evaluations than conditions with milder subsampling (*e.g.*, 50%) or standard lexicase. To be exact, 50%, 25%, 10%, and 5% subsampling complete 15,000,000, 7,500,000, 3,000,000, and 1,500,000 evaluations, respectively. We hypothesized that random subsampling improves lexicase selection because it allows evolutionary searches to run for more generations given a fixed evaluation budget. By terminating all replicates

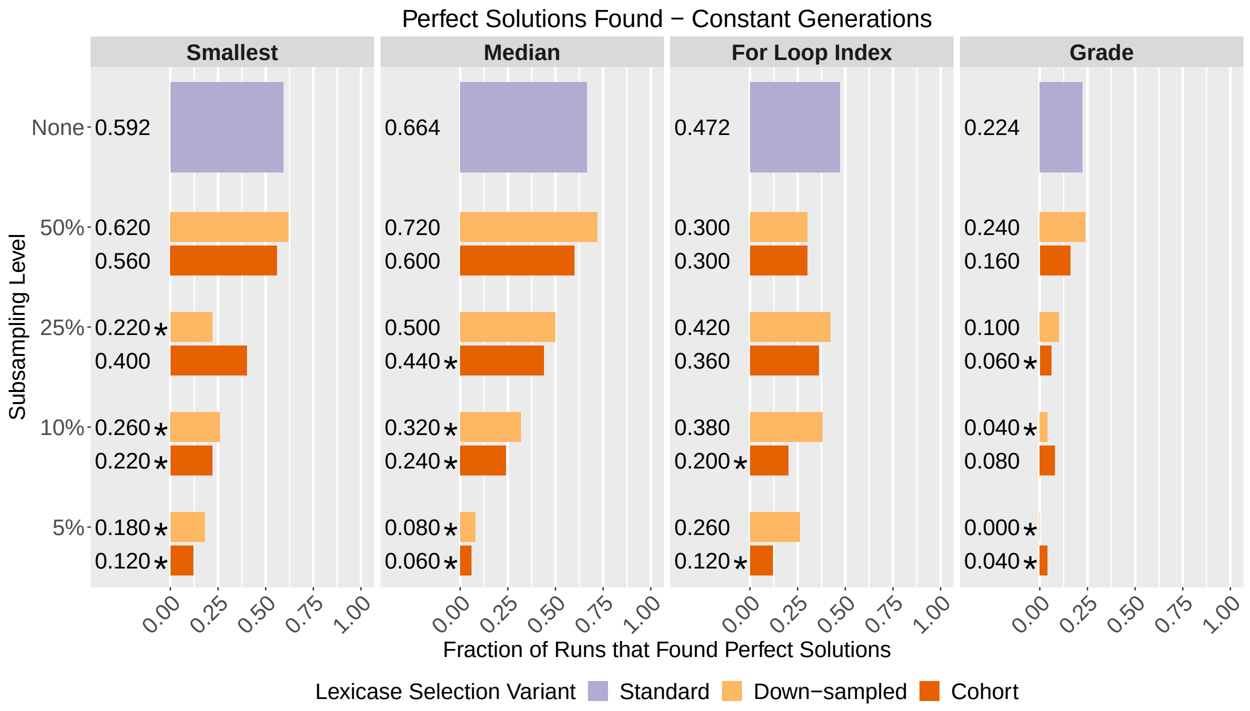

Figure 4.2: Evolutionary results at the end of 300 generations. Bars show the fraction of replicates that found a perfect solution on or before 300 generations. An asterisk (*) to the left of a bar denotes a significant difference compared to the standard lexicase results. Results for standard lexicase (light purple) consist of 250 replicates per problem, while results for down-sampled lexicase (yellow) and cohort lexicase (orange) consist of 50 replicates for each experimental configuration.

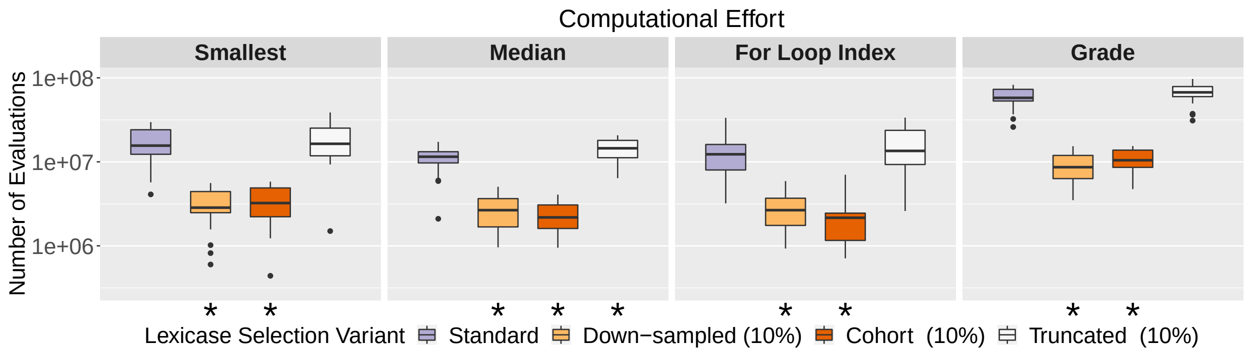

Figure 4.3: The number of evaluations required for each treatment to solve the specified problems. The 25 replicates with the fewest evaluations for each treatment are shown. An asterisk (*) under a box denotes a significant difference between that treatment and standard lexicase.

after 300 generations, we expected subsampling to lose its advantage over standard lexicase.

Given a fixed number of generations, neither down-sampled nor cohort lexicase significantly outperformed standard lexicase at any subsampling level. In fact, down-sampled and cohort lexicase performed significantly worse than standard lexicase on all problems with 5% and 10% subsampling rates except in three cases: cohort at 10% subsampling on Grade, down-sampled at 10% and 5% subsampling on For Loop Index.

As shown in Section 4.4.1, when given equivalent computational budgets (*i.e.*, total number of training case evaluations), subsampling significantly improves lexicase's problem-solving success. However, this experiment shows that when we restrict down-sampled and cohort lexicase to the same number of *generations* as standard lexicase, they both have significantly diminished success on the same problems. These data support our hypothesis that deeper evolutionary searches contribute to the success of the subsampled variations on lexicase selection.

4.4.3 Subsampling reduces computational effort

Next, we explored how subsampling affects the amount of computational effort required to solve problems in the context of lexicase selection. For this experiment, we removed all evaluation and generation termination criteria. Figure 4.3 shows the number of test case evaluations in each of the first 25 replicates for each condition in which a solution evolved (*i.e.*, the 25 replicates that required the least computational effort to solve the problem). We performed a Kruskal-Wallis test (significance level 0.05) to look for significant differences among selection schemes for each program synthesis problem. For problems in which the Kruskal-Wallis test was significant, we performed a post-hoc Mann-Whitney test between standard lexicase and each of the down-sampled, cohort,

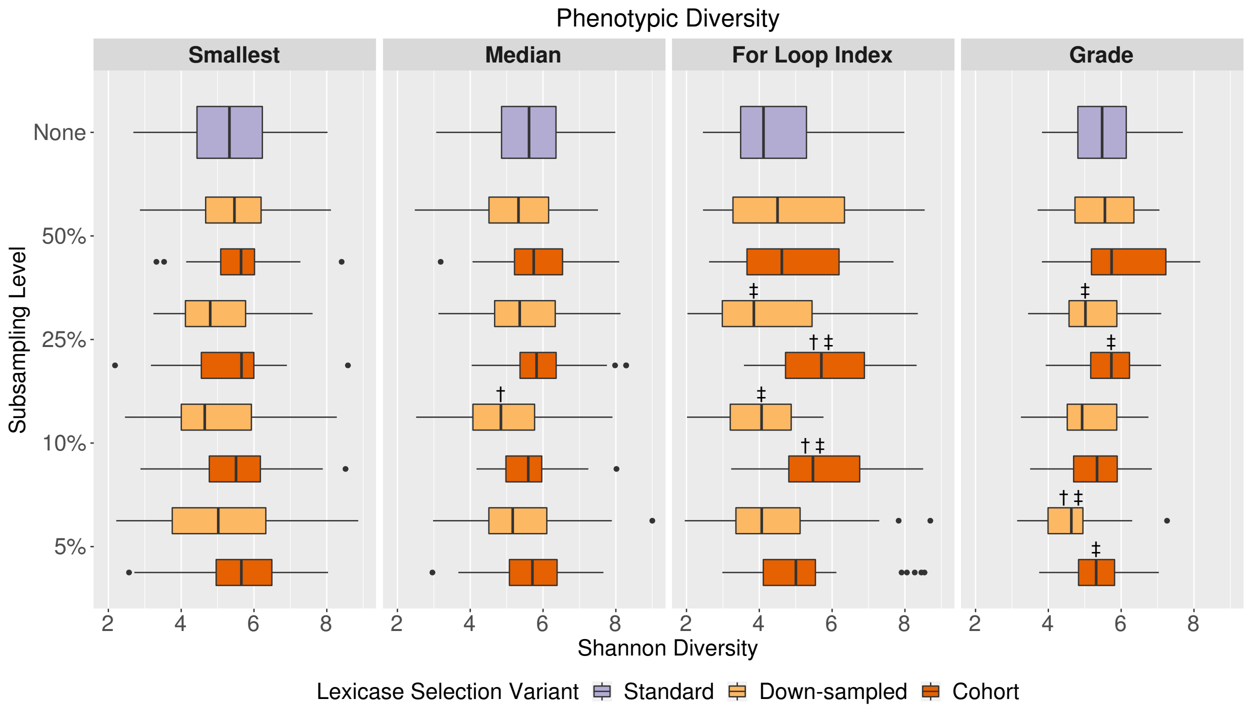

Figure 4.4: Shannon diversity of candidate solution phenotypes at the first generation a perfect solution was found; individual phenotypes were measured as a program's performance on each test from the training and validation sets. A dagger (†) above a box denotes a significant difference with standard lexicase. A double dagger (‡) denotes a significant difference between cohort lexicase and down-sampled lexicase at that subsampling level. Results consist of replicates that found a perfect solution out of 250 replicates for standard lexicase on each problem (purple boxes) and 50 replicates for each combination of problem and subsampling level for down-sampled lexicase (yellow boxes) and cohort lexicase (orange boxes).

and truncated lexicase (with a Holm-Bonferonni correction for multiple comparisons).

Both down-sampled and cohort lexicase used significantly fewer evaluations than standard lexicase on all four problems. Across all problems, truncated lexicase did not use significantly fewer evaluations than standard lexicase; on the Median problem, truncated lexicase actually used significantly *more* evaluations than standard lexicase. The data show a clear trend that 10% subsampling, whether via down-sampling or cohorts, can significantly reduce the number of evaluations needed to solve these program synthesis problems. However, truncated lexicase (using 10% of the training cases per selection event) causes either no effect or a significant increase in required evaluations.

4.4.4 Subsampling does not systematically decrease phenotypic diversity in lexicase selection

Mutations to the binary tags used by the programs to reference modules and memory are often silent (*i.e.*, the phenotype and fitness remain the same) allowing populations to endure high mutation rates that drive adaptive evolution. As a result, almost all replicates maximize genotypic diversity, rendering comparisons uninformative. Therefore, we examined the phenotypic diversity of lexicase and the two subsampled variants.

When evolution produced a candidate solution capable of solving all test cases in the training set, we immediately tested that solution on the cases in the reserved validation set as well. If this candidate solution continued to pass all test cases, we declared it a "perfect solution" and proceeded to measure the phenotypic diversity of the population it arose from. To do so, we tested all programs in the population on all test cases across both the training and validation sets. We designated each candidate solution's performances (in sequence) on all test cases as that solution's phenotype. Figure 4.4 shows the Shannon diversity of these results.

Minimal evidence was found to support our hypothesis that subsampling results in a reduction of phenotypic diversity. After comparing the phenotypic diversity of both down-sampled and cohort lexicase to the standard algorithm, only 2 of 32 configurations resulted in a significant decrease in phenotypic diversity, both of which were down-sampled configurations. Conversely, cohort lexicase actually had significantly *higher* phenotypic diversity than standard lexicase in two configurations. Further, cohort lexicase results had significantly higher phenotypic diversity than down-sampled lexicase in 4 of 16 comparisons.

With only two configurations leading to decreased phenotypic diversity, we cannot conclude that there is a systematic decrease in phenotypic diversity due to subsampling for these program synthesis problems. However, these results hint at a difference between diversity due to down-sampled lexicase and cohort lexicase; we plan to explore this difference in future work.

4.4.5 Cohort lexicase enables more phylogenetic diversity than down-sampled lexicase

As with phenotypic diversity, we recorded the phylogenetic diversity metrics at the time point when populations first found a perfect solution. This timing was necessary; the discovery of a perfect solution is likely to produce a selective sweep, radically altering the structure of the phylogeny. An unavoidable side effect is that the measurements are taken after different numbers of generations have elapsed in different replicates. This discrepancy is potentially concerning, as phylogenetic diversity measurements are sensitive to the number of generations represented within the phylogeny. Adding more generations will, in many cases, legitimately increase the diversity of evo-

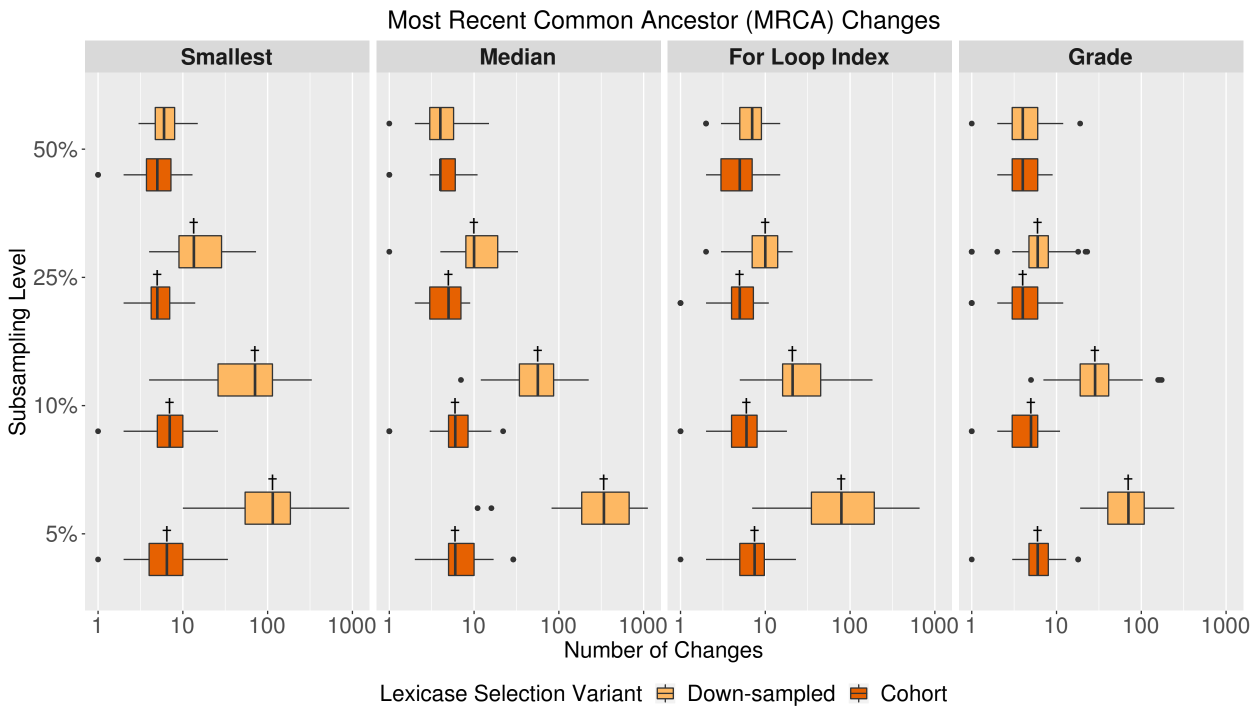

Figure 4.5: Number of times the most recent common ancestor (MRCA) of all extant candidate solutions changed for each evolutionary run. Changes are shown on a logarithmic scale. A dagger (†) above a box denotes a significant difference between cohort lexicase and down-sampled lexicase at that subsampling level. All results shown are from the replicates that found a perfect solution out of 50 replicates per experimental condition.

lutionary history that a population contains. However, the number of generations elapsed can have a disproportionately large effect on a phylogenetic diversity metric, swamping out other effects. In this case, it is these other effects that we are most interested in, as we have already analyzed the causes and effects of the number of generations a population goes through. Fortunately, our results comparing down-sampled vs. cohort lexicase do not appear to be driven by variation in the number of generations elapsed, as the distribution of generations at which the first perfect solution was found did not vary consistently *within* any subsampling level. Because this distribution did vary *among* subsampling levels, we are not attempting to make any strong claims about the relationship between phylogenetic diversity and the degree of subsampling. Here we examine only two of the phylogenetic metrics that were calculated; plots, descriptions, and statistics of all recorded metrics can be found in the supplemental material (Ferguson, 2020).

The most recent common ancestor (MRCA) is the most recently evolved candidate solution from which all extant candidate solutions descend. For this experiment, we tracked the MRCA throughout the evolutionary search, and we examined the number of selective sweeps by counting

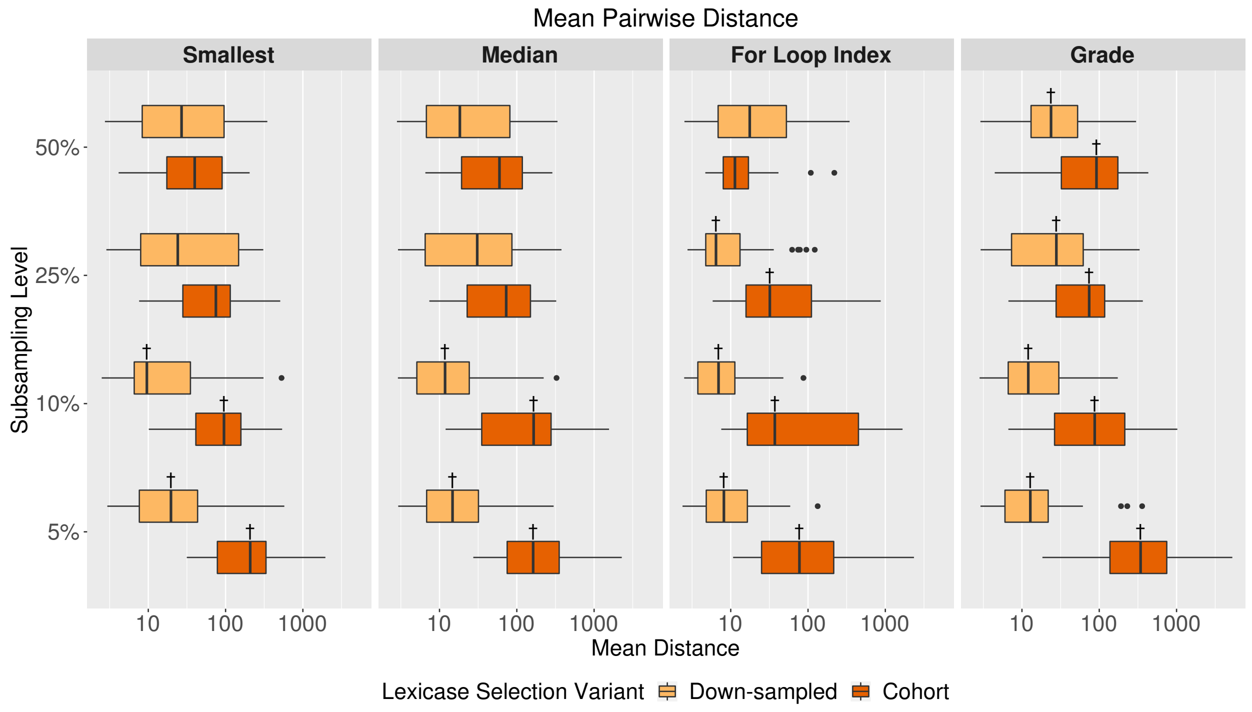

Figure 4.6: Mean distance between all pairs of extant taxa in the phylogenetic tree for runs of both subsampled lexicase variants at different subsampling levels. A dagger (†) above a box denotes a significant difference between cohort lexicase and down-sampled lexicase at that subsampling level. All results shown consist of the replicates that found a perfect solution out of 50 replicates per experimental condition.

the number of times the MRCA changed (see Figure 4.5). For all problems tested, cohort lexicase has significantly fewer MRCA changes than down-sampled lexicase for 5%, 10%, and 25% subsampling levels. This pattern suggests that cohort lexicase inhibits selective sweeps in a way that down-sampled lexicase does not. A likely mechanism for this behavior is that, by explicitly fragmenting the population into groups, cohort lexicase prevents any single candidate solution from sweeping more than one cohort per generation.

Another phylogenetic measure we examined was the phylogenetic divergence (*i.e.,* how distinct the extant taxa are from each other) (Dolson et al., 2018). Here we quantify phylogenetic divergence via mean pairwise distance of the extant solutions in the phylogeny. This metric is calculated as the average distance in the phylogenetic tree between each pair of extant candidate solutions (see Figure 4.6) (Webb, 2000). Cohort lexicase has a significantly higher mean pairwise distance than down-sampled lexicase for all problems at the 5% and 10% subsampling levels. This result indicates that cohort lexicase has significantly higher phylogenetic divergence than down-sampled lexicase, providing further evidence that cohort lexicase is better than down-sampled lexicase at maintaining

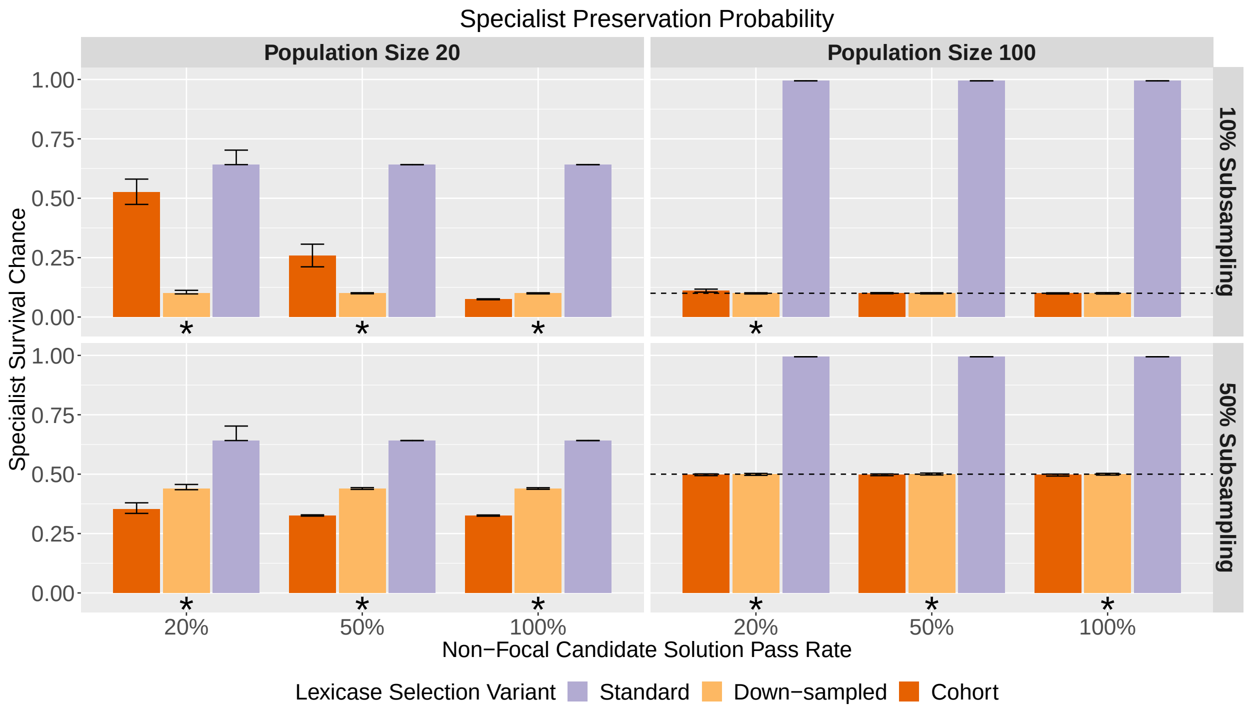

Figure 4.7: Bars show the median probability that a focal specialist will be selected as a parent in the next generation *at least once*; data are aggregated over 100 experimental populations. Error bars show the minimum and maximum probabilities across all populations for that configuration. The dashed lines show the expected probability for both subsampled lexicase variants for configurations where the population size is 100. An asterisk (*) denotes a significant difference between cohort lexicase and down-sampled lexicase; standard lexicase was always significantly different. All configurations shown are for 20 test cases.

phylogenetic diversity. Other phylogenetic diversity metrics were consistent with these results.

Because the differing generation counts prevent us from meaningfully comparing phylogenetic diversity across subsampling levels, all we can say conclusively is that subsampling does not appear to decrease phylogenetic diversity. That said, it may well be the case that greater phylogenetic diversity helps produce better candidate solutions. If so, this factor could explain why more generations (as opposed to more evaluation thoroughness) increase the computational efficiency of lexicase selection. A more targeted investigation will be required to determine how important phylogenetic diversity is to the success of lexicase selection variants.

4.4.6 Subsampling degrades specialist maintenance

Across experimental conditions, lexicase selection has a significantly higher probability of selecting the specialist than either subsampled variant (see Figure 4.7). This result supports our hypothesis that subsampling degrades specialist preservation. Interestingly, down-sampled and cohort lexicase behave differently across the conditions. Exploring these differences can help us better understand the mechanisms that cause a lexicase variant to favor specialists.

When the population size is large, down-sampled and cohort lexicase behave nearly identically. At higher subsampling rates specialists have a higher survival probability in both treatments. At smaller population sizes, higher subsampling rates continue to demonstrate a higher survival probability of specialists in down-sampled lexicase, but not always in cohort lexicase.

At the extreme, when population size, subsampling rate, and generalist pass rate are all small, cohort lexicase has a drastically higher probability of specialist survival than down-sampled lexicase. In this case, the specialist benefits from the low generalist pass rate, since many non-specialists will fail to solve many of the test cases. Specifically, if *all* candidate solutions competing against the specialist fail a given test case, it will be non-discriminatory and effectively ignored. This effect is more pronounced in cohort lexicase, when the specialist is competing only within its cohort (*e.g.*, a cohort of size 2 for a population size of 20 with 10% subsampling), rather than the full population. At a population size of 100, this benefit is lessened because cohorts still contain a relatively large number of candidate solutions. In the remaining configurations, down-sampled lexicase has a higher probability of specialist survival than cohort lexicase.

To better understand these probabilities, consider a situation with two constraints: 1) the specialist solves only its one assigned test case, and 2) every other candidate solution can solve all test cases but the specialist's (*i.e.*, the generalist pass rate is 1.0). While the situation is improbable, it is the worst-case scenario for selecting the specialist; relaxing either constraint could only increase the chance of selecting the specialist. In this situation, the specialist's odds of selection *in a single selection event* under lexicase selection is $\frac{1}{T}$ where T is the number of test cases; that is, the probability of its focal test case being chosen first. The specialist's probability of selection for the entire next generation can be expressed as Equation 4.1 where N is the total population size (Dolson and Ofria, 2018) (for further discussion of selection probabilities under full lexicase selection, see La Cava et al. (2018)).

$$P_{lexicase} = 1 - (1 - \frac{1}{T})^N \tag{4.1}$$

We can modify Equation 4.1 to accommodate down-sampled lexicase by accounting for two cases. First, the specialist's sole test case can be included in the test cases used for this generation, in which case the specialist has a $\frac{D}{T}$ chance of being selected (recall D is the down-sample factor, which divides the number of training cases such that each organism sees $\frac{1}{D}$ of the full training set each generation). Otherwise, the specialist's test case is not included, and the specialist has no chance of being selected. Thus, we arrive at Equation 4.2.

$$P_{down-sampled} = \frac{1 - (1 - \frac{D}{T})^N}{D} \tag{4.2}$$

Finally, we can also account for cohort lexicase selection. Cohort lexicase also gives the specialist a $\frac{1}{D}$ chance of being evaluated against its sole test case. The only difference is in the number of selection events; cohort lexicase can be thought of as standard lexicase being conducted on each cohort. Thus, in the case where the specialist is in the same cohort as its test case, it does not have N selection events to be selected, but instead $\frac{N}{D}$. This gives us the final equation, Equation 4.3.

$$P_{cohort} = \frac{1 - (1 - \frac{D}{T})^{\frac{N}{D}}}{D} \tag{4.3}$$

Plotting these equations, we can see both that down-sampled and cohort lexicase approach a maximum specialist survival probability of $\frac{1}{D}$, and that down-sampled approaches that limit at lower population sizes than cohort lexicase (see Figure 4.8). The plots also show that increasing the number of training cases increases the required population size to reach the $\frac{1}{D}$ limit. Thus the two subsampled lexicase variants have the same maximum specialist selection probability, but smaller populations will see a lower value for cohort lexicase. These theoretical findings help explain our empirical results.

Again, this is the worst-case scenario for the specialist. Further work is needed to see how specialist preservation changes under different situations (*e.g.*, more copies of the specialist, less elite generalists, specialists that solve more than one test case, *etc.*) Figure 4.8 shows only the lower bound on the specialist selection probability.

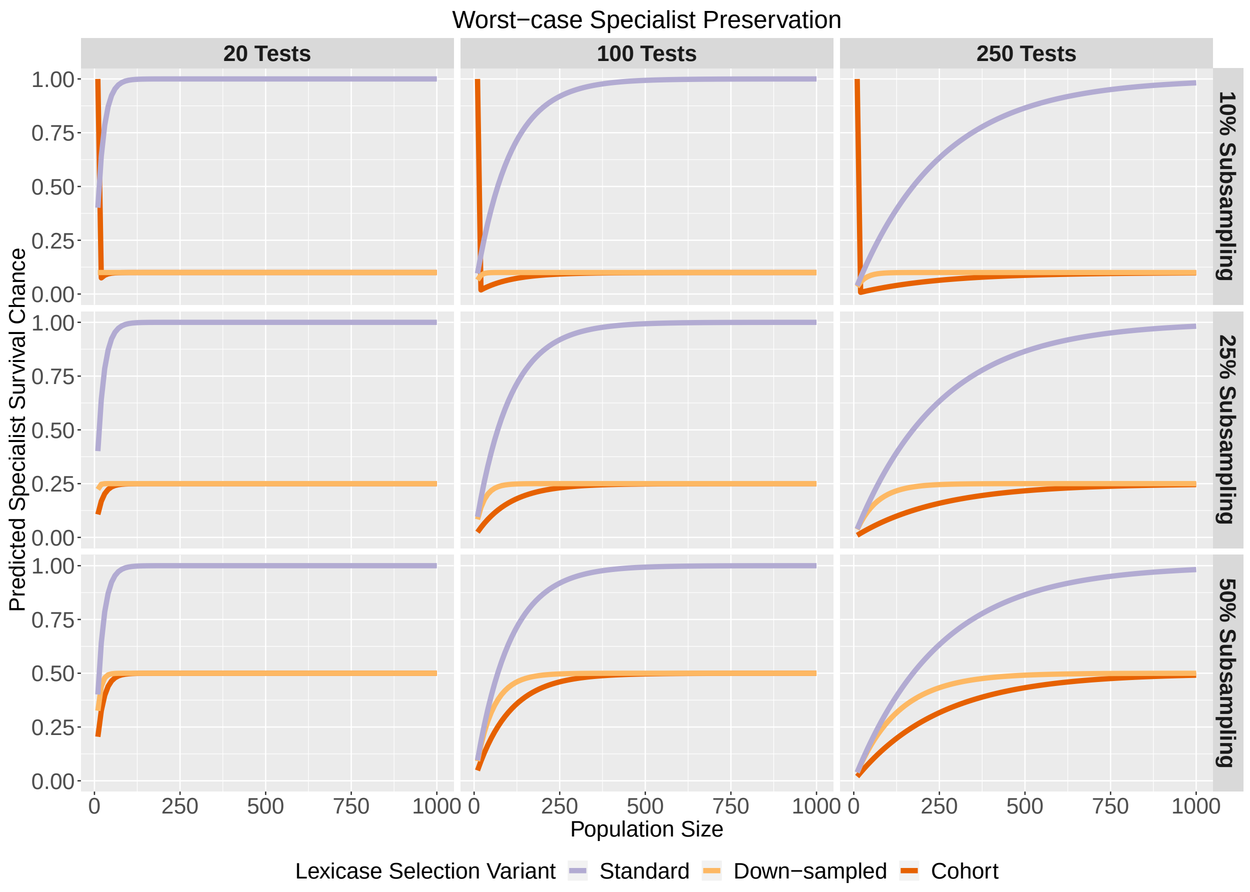

Figure 4.8: Probabilities that the focal specialist will be selected to be a parent in the next generation *at least once* in the situation where there is one specialist, which solves only one test case, but is also the only candidate solution to solve that specific test case. Meanwhile, all other candidate solutions solve all other test cases. Note the special case of a population size of 10 with 10% subsampling. Here, each cohort has one solution, which guarantees selection exactly once with no selective pressure.

4.5 Conclusion

Here, we investigated the effects of random subsampling on lexicase selection. We replicated previous results (Chapter 3), demonstrating that subsampling improves lexicase's problem-solving success, and we have shown that subsampling's success is a result of it enabling deeper evolutionary searches (*i.e.*, running searches for more generations). Moreover, we have shown that subsampling reduces the total computational effort required to evolve solutions in the context of lexicase selection. We expected that applying subsampling to lexicase selection would degrade phenotypic diversity, but have found no evidence of systematic degradation. However, we did find evidence that cohort lexicase is better at generating and preserving phylogenetic diversity than down-sampled lexicase. Finally, we have shown that subsampling does reduce lexicase's capacity to maintain specialist individuals.

Overall, our results highlight the value of random subsampling in lexicase selection, showing that it can improve problem-solving success and save computational effort. However, we also demonstrate that subsampling degrades specialist preservation, and as such, for problems where maintaining specialists is especially important, subsampling might have an overall negative effect on problem-solving success. Future work should explore how subsampling affects both overall population diversity and specialist maintenance at a fine-grained scale and on a wider range of problem types.

Part II

Characterizing search strategies for selection schemes

Chapter 5
An Exploration of Exploration: Measuring the ability of lexicase selection to find obscure pathways to optimality

Authors: Jose Guadalupe Hernandez, Alexander Lalejini, and Charles Ofria

This chapter is adapted from (Hernandez et al., 2022b), which appeared in Genetic Programming Theory and Practice XVII.

In this work, we introduce the *exploration diagnostic* to measure the exploratory capacity of lexicase selection and several of its variants. We find that lexicase selection facilitates better search space exploration than tournament selection, where lexicase selection's ability to explore is sensitive to the ratio between population size and the number of test cases. Additionally, we find that all lexicase variants degraded lexicase selection's exploration capacity, except for epsilon lexicase.

5.1 Introduction

Lexicase-based parent selection algorithms have proven to be highly successful for finding effective solutions to test-based problems in genetic programming (GP) (Helmuth and Abdelhady, 2020; Helmuth and Spector, 2015; Orzechowski et al., 2018). Lexicase selection's success is rooted in its ability to balance strong search space exploration with simultaneous exploitation. That is, lexicase selection maintains meaningfully diverse populations (Helmuth et al., 2016a, 2020) by promoting the coexistence of subpopulations that are each focused on different aspects of a problem (*e.g.*, on different test cases or selection criteria) (Dolson and Ofria, 2018). As such, lexicase selection algorithms are able to explore many promising problem-solving pathways in parallel, optimizing each until an overall solution is found.

Many genetic programming problems are multi-faceted where the quality of a candidate solution must be measured according to its performance on a set of test cases. For such problems, we must decide how to combine performances across many test cases in order to select promising individuals to produce offspring for the next generation. Traditional parent selection algorithms assess the quality of an individual by aggregating their performance on all test cases. The lexicase selection algorithm, however, chooses each parent based on the relative performances of candidate solutions on random permutations of the test set. Specifically, each time a parent is needed, the entire population is considered as candidates for selection, and the full set of test cases are shuffled; each test case is applied sequentially (in the given shuffled order) to the current set of candidates, removing all but the best candidates from consideration until only a single individual remains to be selected (Helmuth et al., 2015). Because the ordering of test cases is different for each parent selection event, individuals that perform well on different subsets of problems are able to coexist (Dolson and Ofria, 2018). Moreover, lexicase selection exerts strong selection pressure to optimize each subpopulation, as only the best candidates on different sequences of test cases are selected.

Indeed, the successes of the original lexicase selection algorithm have inspired numerous variants, each either specialized for solving different categories of problems or designed to address potential shortcomings of the original lexicase algorithm (*e.g.*, computational efficiency). Such variants include epsilon lexicase (La Cava et al., 2018, 2016), down-sampled lexicase (Chapter 3), novelty-lexicase (Jundt and Helmuth, 2019), ALPS lexicase (Helmuth and Abdelhady, 2020), and batch-lexicase selection (Aenugu and Spector, 2019). Many of these variants have been rigorously benchmarked on their problem-solving success and on their ability to maintain phenotypic and phylogenetic diversity (Helmuth et al., 2016a,b; Spector et al., 2018). However, benchmarking is often performed in the context of a particular GP system and with the overall goal of measuring performance on challenging computational problems (*e.g.*, program synthesis benchmark problems from Helmuth and Spector 2015 and Helmuth and Kelly 2021). While such benchmarking is critical for understanding the real-world applicability of a selection scheme, the specific problems used do not always allow us to disentangle the particular pros and cons of each scheme (Hooker, 1995). For this paper, we focus on one important aspect of lexicase-based selection schemes: How do we isolate the *exploration* capabilities of lexicase selection and its variants?

We introduce an "exploration diagnostic" and use it to test how well a set of parent selection algorithms can explore a simple landscape with many uphill pathways of differing peak fitnesses. Our exploration diagnostic allows for the total number of possible evolutionary pathways to be tuned, enabling practitioners to find where an algorithm's exploratory abilities begin to fall off. First, we verify established expectations that lexicase selection better facilitates search space exploration than tournament selection, a more traditional selection algorithm. Next, we evaluate lexicase selection on our exploratory diagnostic with an increasing number of possible pathways and identify its exploratory limitations. Finally, we apply our exploration diagnostic to four variants of lexicase selection: epsilon lexicase, down-sampled lexicase, cohort lexicase, and novelty-lexicase selection.

We find that lexicase selection drives performance improvement at each of the exploration diagnostic difficulty levels that we evaluated. Lexicase selection finds nearly perfect solutions for fitness landscapes with a small number of pathways to be explored, and performance gradually declines as the number of possible evolutionary pathways increases. Additionally, we show that lexicase selection can be sensitive to the ratio between population size and the number of test cases used for evaluating candidate solutions. For small values of ϵ, epsilon lexicase improves the exploratory capacity of lexicase selection. Random subsampling via either down-sampled or cohort lexicase degrades exploratory capacity, but cohort partitioning better preserves lexicase's exploratory capacity than down-sampling. Finally, we did not find compelling evidence that novelty-lexicase improves performance on the exploration diagnostic relative to standard lexicase selection; in fact, the addition of novelty test cases can substantially degrade lexicase's diagnostic performance.

5.2 Exploration diagnostic

Understanding how parent-selection algorithms affect exploration and exploitation within a search space is crucial to tackling increasingly challenging problems. This information can help determine what modifications to an evolutionary algorithm may be needed to improve the likelihood of finding a high-quality solution. Different selection schemes (or other components of an evolutionary algorithm) can alter the trade-off between exploitation and exploration (Eiben and Schippers, 1998). An exploitation-only selection scheme will push the population to the closest optimum and not allow it to explore other promising regions of the search space. Conversely, an exploration-only selection scheme will scatter the population across the entire search space but is unlikely to reach nearby optima. Hence, striking a balance between exploration and exploitation is critical to finding high-quality solutions. Here, we introduce a diagnostic that challenges selection schemes to explore multiple avenues of a search space, each with an upward pathway, with the goal of finding the best avenue to hill climb.

We balanced both exploitation and exploration in our diagnostic. Specifically, we designed a problem with many upward pathways that all have identical slopes, but vary in total length. Since shorter pathways are always equivalent to the beginning of longer pathways, exploration is critical for finding the longest pathway (which will lead to the global optimum). In the end, the only way for an evolving population to determine the length of a pathway is to follow it.

Candidate solutions for this diagnostic are numerical vectors of a designated size (its "cardi-

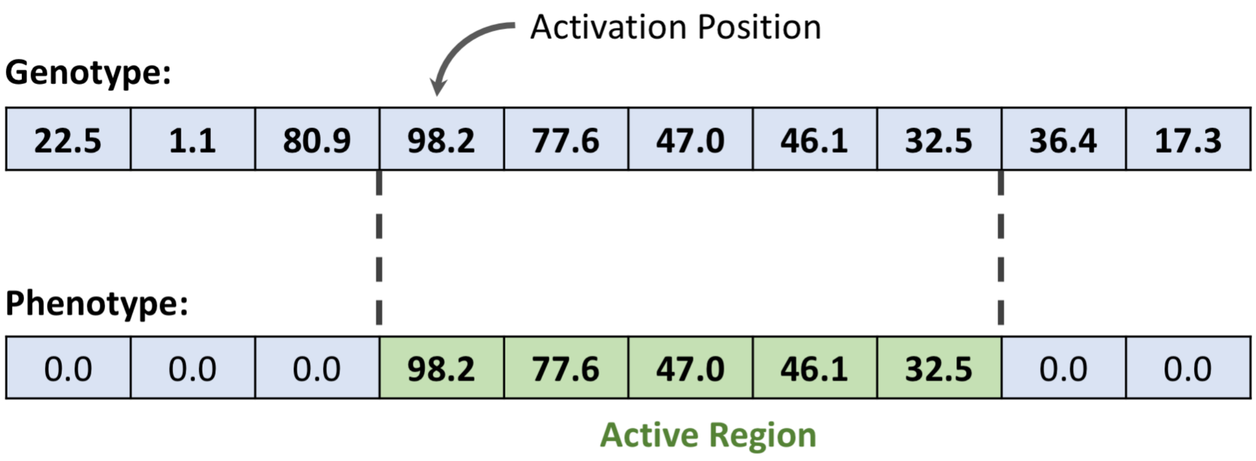

Figure 5.1: An example evaluation with the exploration diagnostic. A candidate solution with a cardinality of 10 is analyzed. The highest value in its vector is identified as 98.2, and its position is marked as the beginning of the active region. The next four values are all in a decreasing sequence (77.6, 47.0, 46.1, and 32.5) and are thus all considered part of the active region. The value after that (36.4) is greater than its predecessor and thus left inactive, closing the active region. All values not in the active region are expressed in the phenotype as 0.0. The total fitness of the sequence is the sum of the values in the phenotype or 0.0 + 0.0 + 0.0 + 98.2 + 77.6 + 47.0 + 46.1 + 32.5 + 0.0 + 0.0 = 301.4.

nality" – we used 100 as the default cardinality in this work). Cardinality determines the number of pathways to local optima in the fitness landscape. Each value in a candidate solution is a floating-point number between 0.0 and 100.0. To evaluate a candidate solution, we first scan its vector to find the maximum value and designate its position as the "activation position" for calculating its fitness. From an intuitive perspective, the activation position defines which peak the candidate solution is climbing toward. Beginning at the activation position, we sum all consecutive values that are less than or equal to each previous position. We stop when either a position is no longer monotonically non-increasing or we reach the end of the vector. We refer to this consecutive sequence of scored values as the "active region" of the candidate solution. All values outside of the active region have zero fitness contribution. The fitness contributions of each position (*i.e.*, each trait) define the "phenotype" of the candidate solution; two candidate solutions that differ only in inactive regions will have identical phenotypes. Figure 5.1 shows an example fitness calculation. Given this search space, the optimal solution will have a 100.0 in every position of its vector starting from the very first, making the entire candidate solution active and each value maximized. However, any candidate solution with an activation position other than the first will not have a pathway to the global optimum that is reachable via hill climbing alone.

Given the large number of pathways that need to be simultaneously explored, this diagnostic allows us to compare the exploration capacity of different selection schemes. Additionally, this diagnostic allows researchers to test the exploration breaking point of a given selection scheme, as increasing the cardinality of the diagnostic increases the exploratory capacity needed to find the best activation position. In this work, we use this diagnostic to test the exploratory limits of lexicase selection along with a number of its variants.

5.3 Lexicase selection

Spector (2012) introduced the lexicase parent selection algorithm for solving GP problems that require programs to produce qualitatively different modes of response for different inputs. Since its introduction, lexicase selection has been demonstrated to be successful across a broad range of problem domains, including automatic program synthesis (Helmuth and Spector, 2015), symbolic regression (La Cava et al., 2016), evolutionary robotics (Moore and Stanton, 2017), genetic algorithms (Metevier et al., 2019), and learning classifier systems (Aenugu and Spector, 2019). See Spector (2012), Helmuth et al. (2015), and Section 3.2 for a more detailed description of lexicase selection. Algorithm 5.1 details the lexicase selection algorithm.

```
1. Mark entire population as current candidates under consideration.
2. Shuffle test_cases into a random order.
3. For each case in test_cases:
    (a) Evaluate each candidate in candidates on case.
    (b) Identify the best_score on case of all candidates.
    (c) Remove each entry from candidates with a score on case worse than
        best_score.
4. Select a random entry from candidates.
```

Algorithm 5.1: Lexicase selection for a single parent. Adapted from (Helmuth et al., 2015).

Many variants of lexicase selection have been proposed, each either specialized for solving a particular type of problem or designed to address potential shortcomings of the original lexicase selection scheme. Below, we describe each of the four variants of lexicase selection examined in this work.

5.3.1 Epsilon lexicase selection

Epsilon lexicase selection relaxes the elitism of the filtering step in standard lexicase selection (step 3c in Algorithm 5.1). When filtering candidates on a given test case, epsilon lexicase retains all individuals with performances within some threshold (ϵ) of the best performance on that test case. The ϵ parameter can be tuned by the practitioner and can be applied either as a proportion of the optimal performance on a given test case or as an absolute threshold.

Epsilon lexicase selection specializes standard lexicase selection for problems where performances on selection criteria are measured using real-valued numbers, such as symbolic regression problems (La Cava et al., 2016; Orzechowski et al., 2018; Spector et al., 2018) or evolving robot controllers (Moore and McKinley, 2016; Moore and Stanton, 2017). The standard lexicase selection algorithm assumes that individuals with equivalent performances on a given test case will have equal scores for that test case. Inconsequential noise in an individual's score on a particular test case could result in arbitrary, but consequential differences in which individuals are selected by the standard lexicase algorithm. By allowing a small ϵ difference between individuals, epsilon lexicase addresses this potential problem.

In this work, we vary ϵ to investigate how it affects exploration. La Cava et al. (2016) observed that behavioral diversity increases at larger values of ϵ. Given ϵ's effect on behavioral diversity, we hypothesize that increasing ϵ will increase the exploration capacity of epsilon lexicase. However, at too high of an ϵ value, we expect *meaningful* exploration to degrade. That is, as ϵ increases beyond a certain point, different adaptive pathways blur together as meaningful differences in test case performances become indistinguishable.

For simplicity, we apply ϵ as a fixed absolute error threshold in this work. Future work, however, should investigate how different applications of ϵ further influence lexicase's exploration capacity (*e.g.*, semi-dynamic and dynamic applications of ϵ from La Cava et al. 2018).

5.3.2 Down-sampled lexicase selection

Down-sampled lexicase applies random subsampling to the selection criteria in order to reduce the per-generation computational effort required by lexicase selection (Chapter 3). Down-sampled lexicase uses a random subset of test cases each generation, which reduces the number of test cases on which each individual in the population must be evaluated every generation. After down sampling, the standard lexicase procedure is used to choose parents.

For an equivalent number of total evaluations, down-sampled lexicase allows practitioners to run their evolutionary computing system for more generations or with a larger population size; both of which have been shown to improve problem-solving success (Chapter 4 and Helmuth and Spector (2020)). In this work, we investigate how down sampling affects lexicase selection's exploratory capacity. While we found no evidence that down sampling reduces phenotypic diversity across a range of program synthesis problems in Chapter 4, we did find that down sampling degrades specialist maintenance. We hypothesize that down sampling's negative effect on specialist maintenance harms its exploratory capacity. Entire categories of test cases may be excluded on any given generation, and candidate solutions specializing on those test cases may be lost as a result. Such dynamics may prevent extensive exploration of valuable niches.

5.3.3 Cohort lexicase selection

Cohort lexicase partitions the test case set and the population each into an equal number of cohorts (Chapter 3). Each generation, cohort membership is randomly assigned, and each cohort of candidate solutions is paired with a cohort of test cases. Each cohort of candidate solutions is evaluated only on the test cases in the paired test case cohort, which, like down-sampled lexicase, reduces the required number of per-generation evaluations relative to standard lexicase selection. Unlike down-sampled lexicase, however, cohort lexicase ensures that every test case in the full set is used every generation, as each cohort of candidate solutions competes on a different subset of the full set. To select a parent, cohort lexicase first selects a cohort to choose from; previous work guaranteed an equal number of parents were selected from each cohort each generation (Chapter 3 and 4). Candidate solutions only compete against other solutions within their respective cohort, and within-cohort competition is arbitrated by the test cases in the associated cohort of tests.

In this work, we investigate how the number of cohorts that we partition the population and test set into influences lexicase selection's capacity for exploration. For similar reasons to down-sampled lexicase, we expect cohort lexicase selection to degrade lexicase selection's exploratory capacity. However, because cohort lexicase uses every test case in every generation, we expect it to better support exploration than down-sampled lexicase. As we increase the size of cohorts (and decrease the number of cohorts), we expect cohort lexicase to approach the exploratory abilities of standard lexicase selection. This could be due to the fact that as cohort size increases, the chances of a specialist being paired with the test cases it specializes on also increases.

5.3.4 Novelty-lexicase selection

Novelty-lexicase selection combines standard lexicase selection with novelty search (Jundt and Helmuth, 2019). Novelty search disregards functional objectives and instead searches for behavioral novelty, steering populations to continuously explore new regions of the search space (Lehman and Stanley, 2011a). As such, novelty search is argued to be well-suited for solving problems with deceptive fitness landscapes where local gradients lead *away* from the global optimum (Lehman et al., 2008). Novelty-lexicase selection incorporates ideas from novelty search into lexicase selection.

Novelty-lexicase selection (as introduced in Jundt and Helmuth 2019) requires that the entire population be evaluated on all test cases. For each member of the population, novelty-lexicase selection computes their “novelty score” on each test case. A novelty score measures how different a candidate solution’s output on a given test case is from the rest of the population. Here, a candidate solution’s novelty score on a test case equals the average distance between its output and the k nearest neighbor outputs for that test case. Novelty-lexicase selection incorporates novelty scores by augmenting the test case set with an additional novelty test case for every original test case. Using this augmented set of test cases, the standard lexicase procedure is used to choose parents.

In this work, we use our exploration diagnostic to compare the exploratory capacity of novelty-lexicase selection (at k =1, 2, 4, 8, 15, 30, and 60) and standard lexicase selection ($k = 0$). Jundt and Helmuth (2019) found that novelty-lexicase selection generally maintained more behavioral diversity than standard lexicase selection on several program synthesis problems. As such, we expect the addition of novelty score test cases to improve lexicase selection’s exploratory capacity on our exploration diagnostic.

5.4 Diagnosing the exploratory capacity of lexicase selection and its variants

We conducted a series of experiments to analyze the exploratory limits of standard lexicase selection and four of its variants: epsilon lexicase, down-sampled lexicase, cohort lexicase, and novelty-lexicase. For each experiment, unless stated otherwise, we evolved populations of 500 numerical vectors on our exploration diagnostic with a cardinality of 100 for 50,000 generations. Across all experiments, we ran 50 replicates of each constituent treatment. We initialized populations to the lowest point in the fitness landscape, vectors of all 0.0s.

When evaluating a candidate solution, we calculated a score associated with each position in its vector according to the exploration diagnostic (Figure 5.1). We used this collection of scores as test case qualities for lexicase selection and its variants. For this work, we report quality directly; for comparison to other studies, note that test case *error* is the amount that quality is below 100. When a single fitness value was required (*e.g.*, for tournament selection), we summed the individual test case qualities to determine the solution's aggregate fitness.

Selected candidate solutions reproduced asexually, and we applied point-mutations to offspring at a per-position rate of 0.7%. The magnitude of each mutation was drawn from a normal distribution with a mean of 0.0 and a standard deviation of 1.0 ($\mathcal{N}(0,1)$). When mutations would raise a trait to a value x where $x > 100$, we rebounded that trait to $200 - x$, ensuring that each trait value remained less than or equal to 100. When mutations would lower a trait below 0.0, we reset that trait to 0.0.

For each replicate of each experiment, we extracted the most performant individual in the population (*i.e.*, the individual with the highest aggregate score) to compare across treatments. For different diagnostic cardinalities (*i.e.*, different numbers of test cases), the range of possible aggregate scores differs; as such, we normalized all aggregate scores by dividing by the cardinality, which results in a value between 0.0 and 100.0.

To identify the number of pathways being explored by a population, we measured the number of unique activation positions within each population. Using this measurement, we calculated "activation position coverage" as the fraction of possible activation positions represented in a population.

For each experiment, we report both mean performance and mean activation position coverage over time (each with a bootstrapped 95% confidence interval), and we compare measurements from the final generation across treatments. For each comparison, we performed a Kruskal-Wallis test to determine if there were significant differences; if so, we applied a Wilcoxon rank-sum test to distinguish between pairs of treatments, applying Bonferroni corrections for multiple comparisons where appropriate.

The software used to conduct experiments, statistical analyses, experimental data, and guides for replication are included in our supplemental material (Hernandez et al., 2021). See Section 5.6

for more details.

5.4.1 Lexicase selection out-explores tournament selection

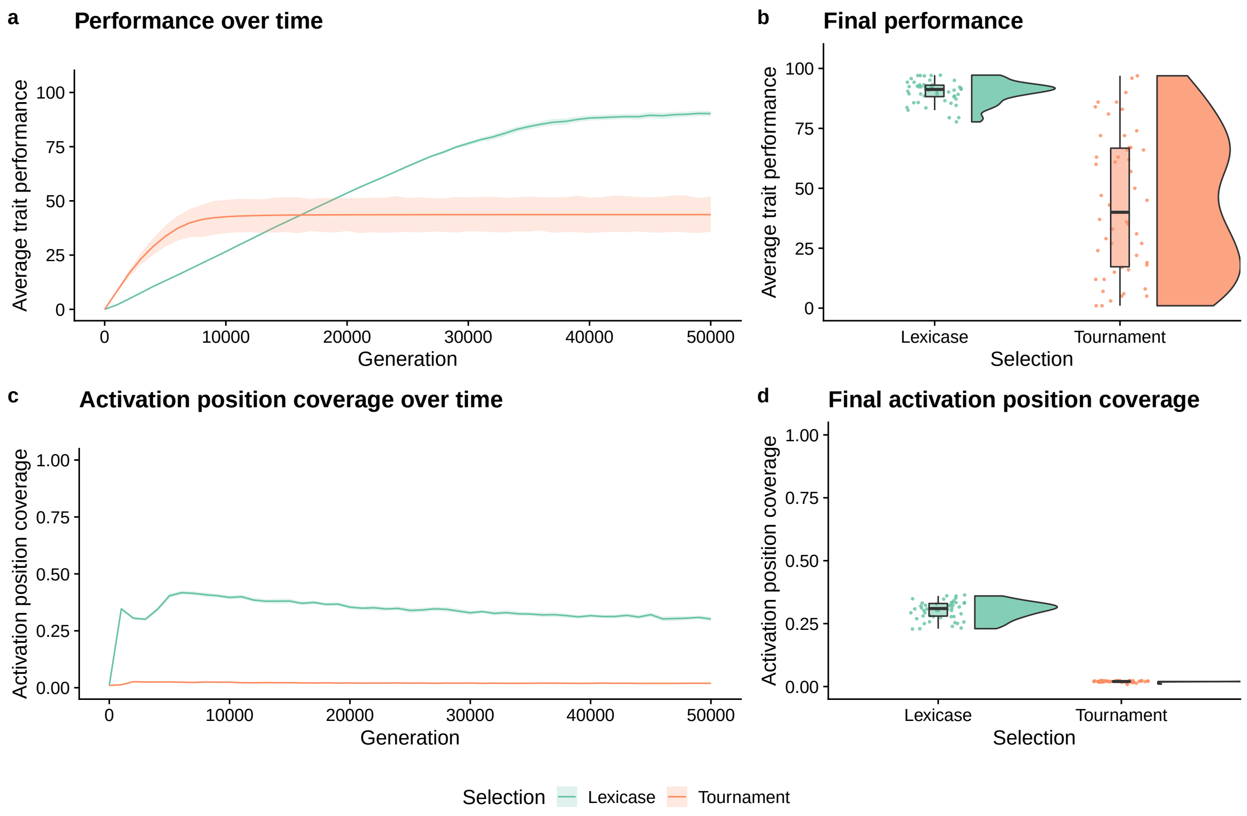

Figure 5.2: Lexicase selection versus tournament selection on the exploration diagnostic. Panels (a) and (b) show performance over time and at the end of 50,000 generations, respectively. Likewise, panels (c) and (d) show activation position coverage over time and at the end of 50,000 generations, respectively. For panels (a) and (c), each line gives the mean value across 50 replicates, and the shading around each mean gives a 95% confidence interval.

First, we used the exploration diagnostic to test well-established expectations that lexicase selection improves search space exploration relative to tournament selection. Unlike lexicase selection, tournament selection does not reliably maintain multiple niches within a population (Dolson and Ofria, 2018); as such, we expected it to perform worse than lexicase selection on the exploration diagnostic. For this experiment, we used tournaments of eight individuals.

Consistent with our expectations, we found that lexicase selection outperforms tournament selection on the exploration diagnostic (Figure 5.2; Wilcoxon rank-sum test: $p < 10^{-4}$). Early on, populations evolving under tournament selection converge to a single local optimum in the exploration diagnostic (*i.e.*, a single activation position); without a mechanism to escape, populations become stuck and fail to continue exploring the search space. Lexicase selection, however, rewards specialists for different activation positions, allowing the population to continuously explore different evolutionary pathways. Indeed, we found that lexicase selection maintains substantially more "activation-position" specialists than tournament selection (Figure 5.2; Wilcoxon rank-sum test: $p < 10^{-4}$).

5.4.2 The exploratory capacity of lexicase selection degrades as we increase diagnostic cardinality

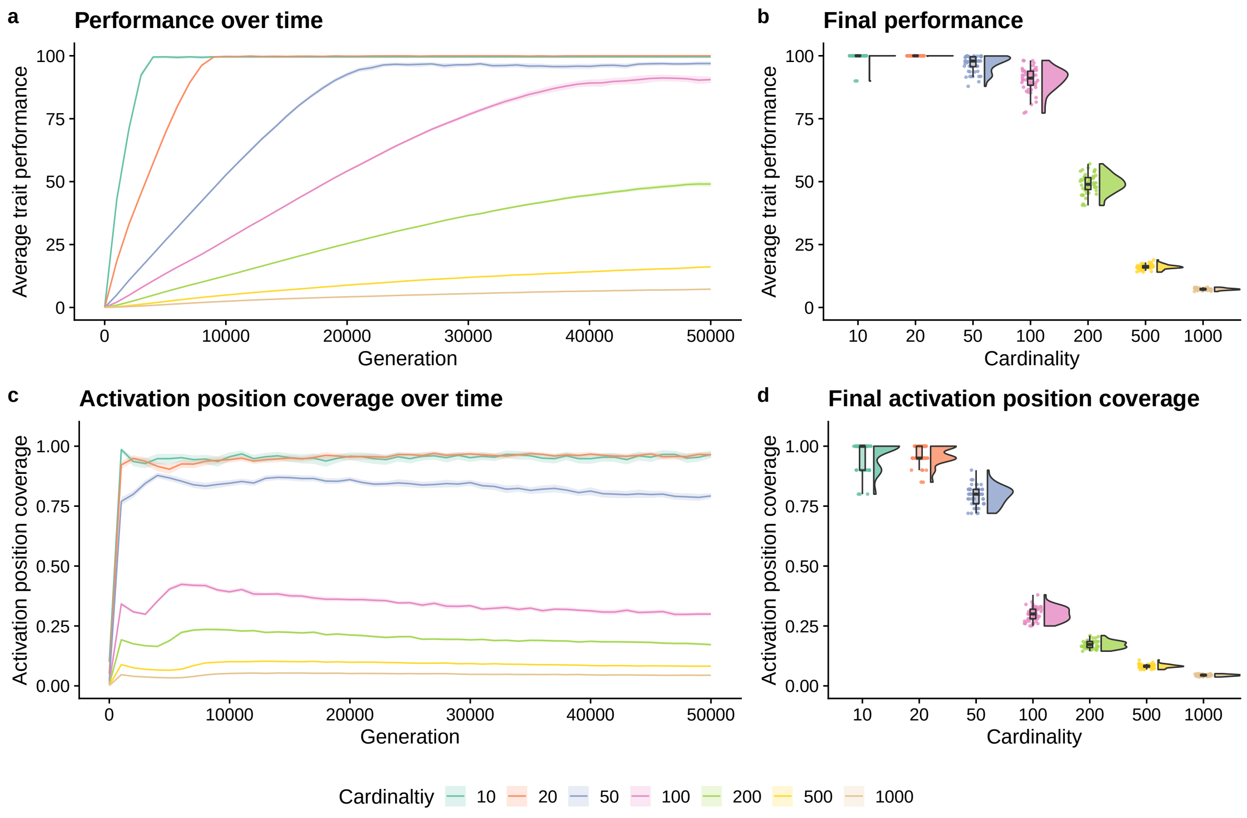

Figure 5.3: Lexicase selection at a range of exploration diagnostic cardinalities. Panels (a) and (b) show performance over time and at the end of 50,000 generations, respectively. Likewise, panels (c) and (d) show activation position coverage over time and at the end of 50,000 generations, respectively. For panels (a) and (c), each line gives the mean value across 50 replicates, and the shading around each mean gives a 95% confidence interval.

Next, we evaluated standard lexicase selection on the exploration diagnostic at cardinalities 10, 20, 50, 100, 500, and 1,000. Cardinality defines the number of potential pathways that must be explored by a population to guarantee to find the global optimum; increasing cardinality obscures the path to optimality. Cardinality also corresponds to the number of test cases (*i.e.*, niches) that individuals can specialize on. For a fixed population size, increasing the number of test cases decreases the long-term survival probability of any single specialist under lexicase selection (Dolson and Ofria, 2018), which could negatively affect lexicase's capacity to fully explore pathways in the search space. For these reasons, we expected lexicase selection's performance on the exploration diagnostic to degrade as we increased cardinality.

Figure 5.3 shows lexicase selection's performance at each cardinality of the exploration diagnostic. Across all cardinalities, lexicase selection improves performance over time. Notably, treatments with cardinalities 10, 20, and 50 each perform near optimally after 50,000 generations, and populations evolved under cardinality 100 perform relatively well. Higher cardinalities (*e.g.*, 200, 500, and 1000), however, perform substantially worse (Wilcoxon rank-sum tests: $p < 10^{-4}$) and appear to need more time to converge on their maximal performance. These data verify that increasing diagnostic cardinality also increases the exploration diagnostic's difficulty, as lexicase selection's performance degrades as cardinality increases.

We also found that populations that evolved at lower diagnostic cardinalities maintained a larger coverage of unique activation positions than populations that evolved at higher diagnostic cardinalities (Figure 5.3). Such diversity maintenance likely drove lexicase selection's ability to continuously explore pathways in the search space.

In these experiments, we used a population size of 500, resulting in 500 selection events per generation. In each selection event, scores for vector positions (Figure 5.1) are prioritized in a random order. Across a population, we expect that positions that are consistently rewarded should maintain solutions that start at that position. The optimal solution requires the initial position to be the highest in the population, but this position may, by chance, never be evaluated first during lexicase selection. The probability of this occurring varies with cardinality. With a population size of 500 and a vector with 50 positions (*i.e.*, a diagnostic cardinality of 50), there is a 0.004% chance (1 in 25,000) of the initial position never being chosen first in a generation, making it unlikely to go unselected. Increasing the cardinality to 100, however, increases the chance for the first position to go unselected to 0.657% (1 in 152)—a much more likely occurrence that may explain the reduced performance at cardinality 100 relative to cardinality 50. By cardinality 200, the probability for the first position to go unselected within a given generation rises to 8.157%, an even more likely occurrence.

One way to combat these dynamics is to increase population size, which would allow lexicase selection to support higher levels of exploration by reducing the chances of any given starting position from being skipped over by selection in any single generation. However, increasing population size can be computationally expensive, as more individuals would need to be evaluated every generation. Decreasing the depth of evolutionary search by reducing the number of generations evaluated is one way to balance the cost of increasing population size. For a fixed computational budget, can increasing population size at the expense of evaluating fewer generations of evolution pay off under lexicase selection?

5.4.3 Increasing population size can improve lexicase selection's exploratory capacity

To test whether increasing population size can improve lexicase selection's exploratory capacity, we extended the runtime of our experiment and compared lexicase selection's performance on the exploration diagnostic (with a cardinality of 100) at two population sizes: 500 and 1,000. Because increasing population size increases per-generation computational effort, we ran both conditions for a fixed number of test case evaluations, evolving populations of 500 individuals for twice as many generations as populations of 1,000 individuals (1,000,000 and 500,000 generations, respectively). As such, lineages from 500-individual populations take two reproductive steps in the search space for every one step reproductive step taken by a 1000-individual population. This difference may allow the smaller populations to more rapidly exploit their initial position in the search space. However, if larger populations are able to maintain more pathways in the search space, they may eventually outperform smaller populations.

As expected, we found that increasing population size allows lexicase selection to maintain more starting positions for the entire duration of our experiment (Figure 5.4). Smaller populations initially outperform larger populations (given a fixed computational budget); however, despite running for fewer total generations, larger populations eventually outperform the smaller populations (Figure 5.4; Wilcoxon rank-sum test: $p < 10^{-4}$). These data suggest that, for a fixed number of test case evaluations, we can indirectly tune lexicase selection's level of search space exploitation and exploration by adjusting our allocation of computational resources between generations of evolution and population size.

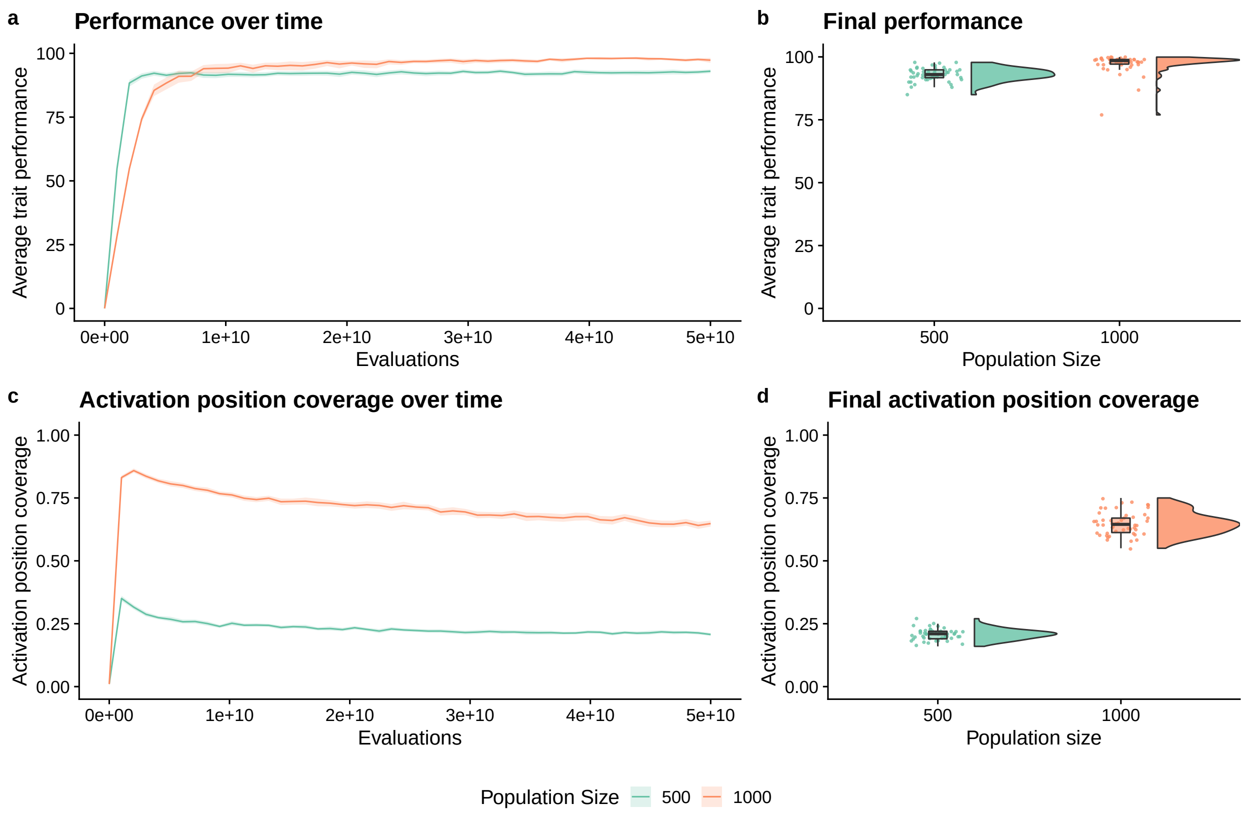

Figure 5.4: Lexicase selection's performance on the exploration diagnostic at different population sizes. Panels (a) and (b) show performance over time and at the end of the experiment, respectively. Likewise, panels (c) and (d) show activation position coverage over time and at the end of the experiment, respectively. For panels (a) and (c), each line gives the mean value across 50 replicates, and the shading around each mean gives a 95% confidence interval.

5.4.4 Relaxing lexicase selection's elitism can improve exploration

As discussed in Section 5.3.1, epsilon lexicase relaxes the elitism of lexicase selection. To test whether this relaxation of elitism affects exploration, we compared standard lexicase selection and epsilon lexicase selection on the exploration diagnostic. Specifically, we evolved 50 replicate populations at each of the following ϵ values: 0.0 (standard lexicase), 0.1, 0.3, 0.6, 1.2, 2.5, 5.0, and 10.0.

Epsilon lexicase with small values of ϵ (0.1 and 0.3) outperforms standard lexicase selection on the exploration diagnostic (Figure 5.5; Wilcoxon rank-sum tests: $p < 10^{-4}$). Extreme values of ϵ (5.0 and 10.0) significantly degrade performance relative to standard lexicase selection (Wilcoxon rank-sum tests: $p < 10^{-4}$). Interestingly, intermediate values of ϵ (0.6 and 1.2) perform best during the first approximately 20,000 generations, but are eventually outperformed by treatments with

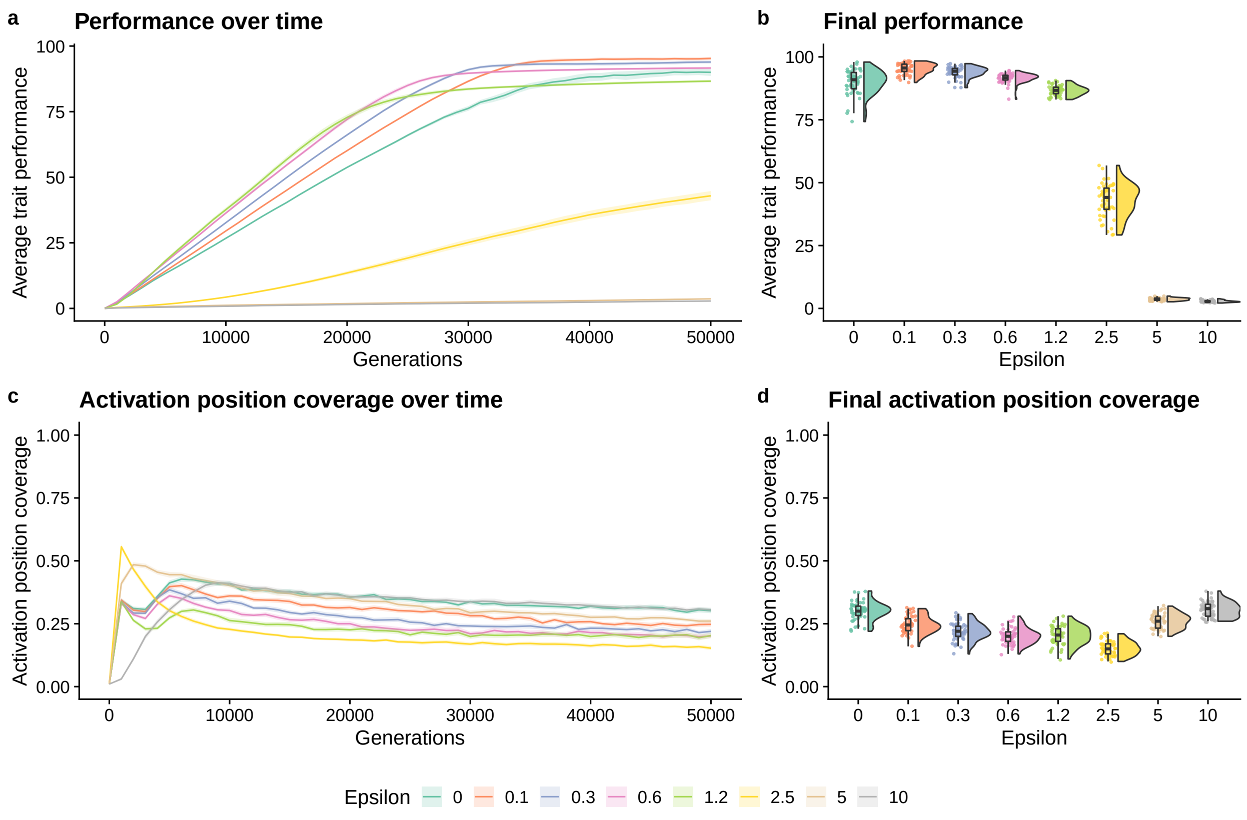

Figure 5.5: Epsilon lexicase selection's performance on the exploration diagnostic at a range of ϵ values. Panels (a) and (b) show performance over time and after 50,000 generations of evolution, respectively. Likewise, panels (c) and (d) show activation position coverage over time and after 50,000 generations of evolution, respectively. For panels (a) and (c), each line gives the mean value across 50 replicates, and the shading around each mean gives a 95% confidence interval.

smaller values of ϵ. Unlike previous experiments, the relative levels of activation position coverage among conditions do not correspond with diagnostic performance.

In general, epsilon lexicase is expected to have two main advantages over standard lexicase selection (La Cava et al., 2016): (1) it allows small amounts of noise in the evaluation data to be ignored, and (2) it prevents nearly identical scores from determining which candidate solutions win, potentially allowing for greater coexistence. While the first mechanism cannot be at play here (since all scores are deterministic), the second advantage could provide additional support for solutions further along a given pathway. That is, solutions that begin optimizing at an earlier point in their vector, by definition, must have slightly lower values for later positions in their activated region. In standard lexicase, when two solutions had overlapping activation regions, the one that starts later would have an advantage at all overlapped sites. In epsilon lexicase, however, the earlier start (*i.e.*, the one with more long-term potential) now has a better chance to pass lexicase selection's selective filter.

5.4.5 Down-sampling degrades lexicase selection's exploratory capacity

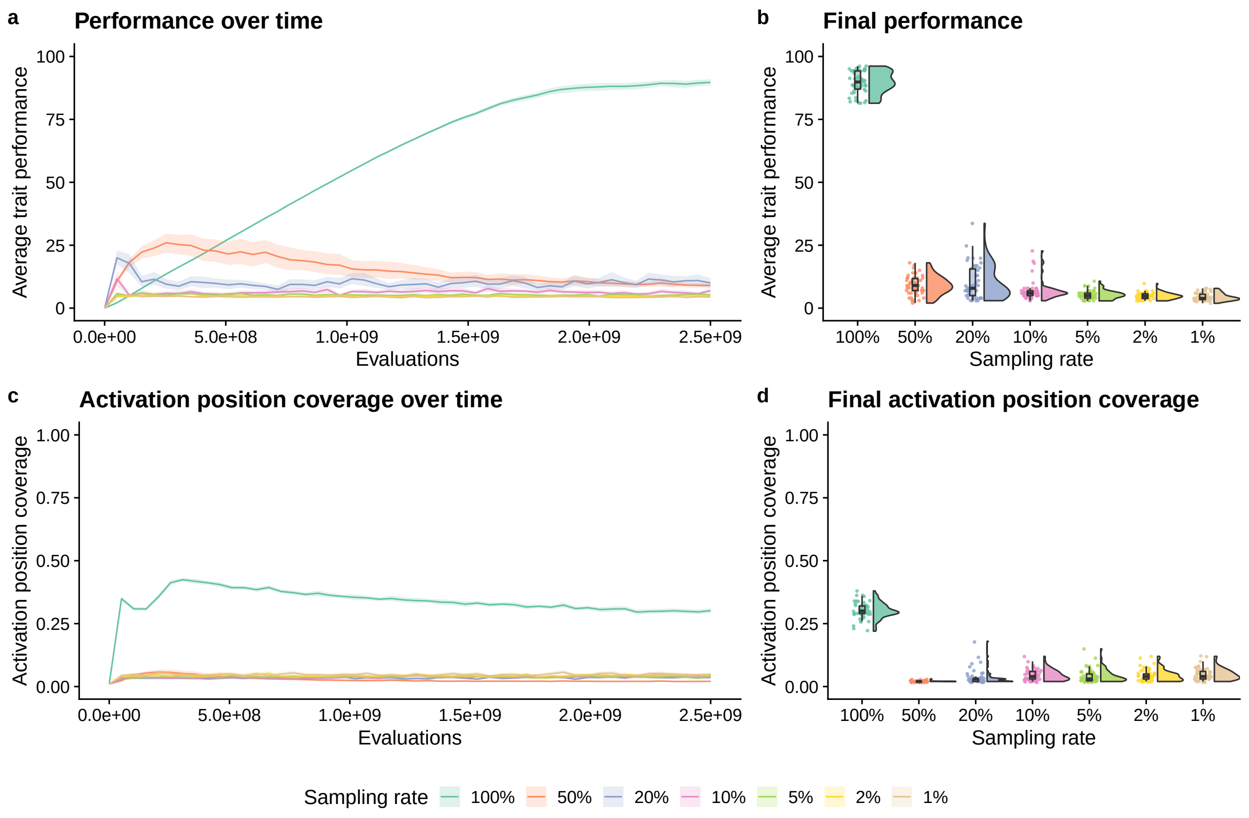

Figure 5.6: Down-sampled lexicase selection's performance on the exploration diagnostic at a range of subsampling rates. Panels (a) and (b) show performance over time and at the end of the experiment, respectively. Likewise, panels (c) and (d) show activation position coverage over time and at the end of the experiment, respectively. For panels (a) and (c), each line gives the mean value across 50 replicates, and the shading around each mean gives a 95% confidence interval.

Next, we investigated whether down-sampling affects lexicase selection's exploratory capacity by comparing the performance of lexicase selection at a range of sampling rates: 100% (standard lexicase), 50%, 20%, 10%, 5%, 2%, and 1%. For example, a 10% sampling rate means that in each generation we randomly selected 10 of the 100 possible test cases (for a diagnostic cardinality of 100) to be used for parent selection. Down-sampling reduces the per-generation computational effort required for parent selection by conducting fewer test case evaluations (Section 5.3.2). For a fair comparison across different sampling rates, we limited the computational budget to a maximum of 2.5×10^9 test case *evaluations* by varying the number of generations of evolution for each subsampling rate (100%: 50,000 generations, 50%: 100,000 generations, 20%: 250,000 generations, 10%: 500,000 generations, 5%: 1,000,000 generations, 2%: 2,500,000 generations, and 1%: 5,000,000 generations).

Any amount of down-sampling significantly degraded lexicase selection's performance on the exploration diagnostic for the allotted computational budget (Figure 5.6; Wilcoxon-rank sum tests: $p < 10^{-4}$). Down-sampled lexicase selection's drop in performance is likely attributed to frequent mismatches between candidate solutions and the test cases that they are specialized on. As the proportion of test cases used in each generation decreases, so too does the probability of a solution encountering the same set of test cases for multiple generations in a row. As such, a solution has a reduced chance of encountering the test cases for which it is most optimized (Chapter 4). These dynamics will repeatedly remove solutions with small active regions, thereby reducing population diversity. Indeed, we found that down-sampling substantially reduces the number of activation position specialists represented in the population (Figure 5.6; Wilcoxon rank-sum tests: $p < 10^{-4}$). In fact, any down-sampling used appears to have a strong negative effect, substantially reducing performance in all cases.

We repeated this experiment, except we increased population size instead of increasing generations of evolution for down-sampled lexicase; that is, we ran each condition for an equivalent number of generations but differing population sizes to maintain a fixed number of evaluations. We report these data in our supplemental material (Hernandez et al., 2021). Overall, the patterns were similar to that of increasing generations of evolution. Initially, down-sampled lexicase outperforms standard lexicase on the exploration diagnostic; however, standard lexicase eventually outperforms down-sampled lexicase across all subsampling rates (Hernandez et al., 2021).

5.4.6 Cohort partitioning degrades lexicase selection's exploratory capacity

Next, we evaluated whether partitioning the population and test cases into cohorts affects the exploration capacity of lexicase selection. We compared the performance of standard lexicase to that of cohort lexicase at a range of cohort sizes (given as the proportion of the population and the set of test cases used in each cohort): 100% (standard lexicase), 50%, 20%, 10%, 5%, 2%, and 1%. For example, a cohort size of 10% means that the population (of 500 individuals) is divided into 10 cohorts of 50 individuals each, and the test cases (100 total) are also divided into those same 10 cohorts, with 10 test cases in each. Like down-sampled lexicase, cohort lexicase reduces the per-generation computational effort required for parent selection by evaluating each cohort of candidate

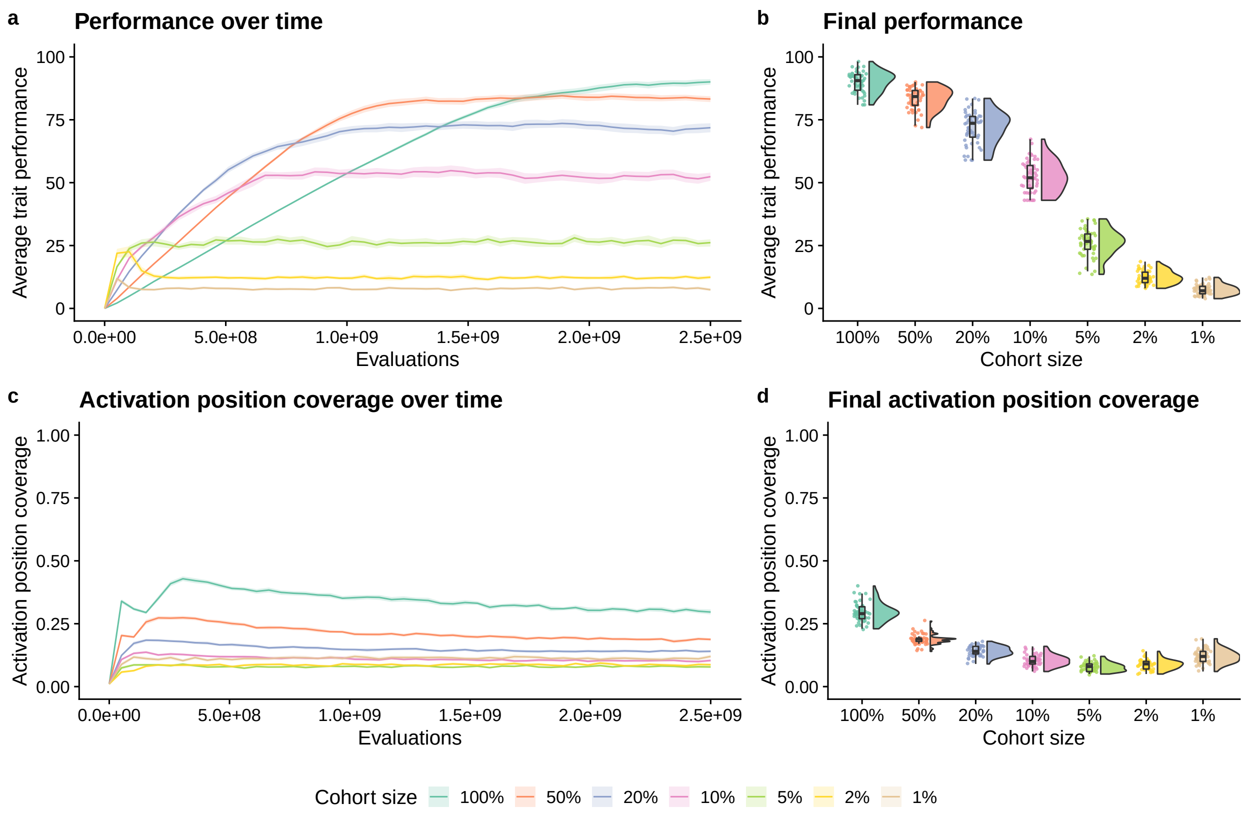

Figure 5.7: Cohort lexicase selection's performance on the exploration diagnostic at a range of partitioning rates. Panels (a) and (b) show performance over time and at the end of the experiment, respectively. Likewise, panels (c) and (d) show activation position coverage over time and at the end of the experiment, respectively. For panels (a) and (c), each line gives the mean value across 50 replicates, and the shading around each mean gives a 95% confidence interval.

solutions on only one of the test case cohorts (Section 5.3.3). Likewise, for a fair comparison across different cohort sizes, we limited the computational budget to a maximum of 2.5×10^9 test case evaluations by varying the number of generations of evolution for each cohort size (100%: 50,000 generations, 50%: 100,000 generations, 20%: 250,000 generations, 10%: 500,000 generations, 5%: 1,000,000 generations, 2%: 2,500,000 generations, and 1%: 5,000,000 generations).

As with down-sampled lexicase, any level of cohort partitioning degrades lexicase's performance on the exploration diagnostic for the allotted computational budget (Figure 5.7; Wilcoxon rank-sum tests: $p < 10^{-4}$). However, cohort lexicase does not appear to degrade lexicase selection's performance to the same degree as down-sampled lexicase for a given subsampling rate (Figure 5.6). Moreover, standard lexicase took longer (more total evaluations) to outperform cohort lexicase than to outperform down-sampled lexicase. These data suggest that cohort partitioning (with intermediate levels of partitioning) may be a better method of random subsampling in the context of lexicase selection.

We repeated this experiment, except we increased population size instead of increasing generations of evolution for cohort lexicase; that is, we ran each condition for an equivalent number of generations but differing population sizes to maintain a fixed number of evaluations. We report these data in our supplemental material (Hernandez et al., 2021). The overall patterns were qualitatively different and warrant further exploration in future work. We found no compelling evidence that cohort lexicase outperformed standard lexicase in the given computational budget; however, we did find that populations evolving under cohort lexicase (with larger population sizes) maintained more activation position coverage than standard lexicase selection (Hernandez et al., 2021). Further, some of the cohort sizes were on an upward trajectory when the runs finished and may eventually outperform standard lexicase given a larger computational budget.

5.4.7 Cohort lexicase out-explores down-sampled lexicase

Next, we independently verified that cohort lexicase out-explores down-sampled lexicase on the exploration diagnostic. To do so, we compared the performance of cohort lexicase and down-sampled lexicase with their most performant parameterizations: a 50% cohort size and a 50% sampling rate, respectively. We again limited the computational budget to a maximum of 2.5×10^9 test case evaluations (100,000 generations of evolution for both conditions), and we ran 50 new replicates of each condition for comparison.

As expected given Figures 5.6 and 5.7, cohort lexicase outperformed down-sampled lexicase by a substantial margin for the given computational budget (Figure 5.8; Wilcoxon rank-sum test: $p < 10^{-4}$). Interestingly, down-sampled lexicase appears to briefly outperform cohort lexicase in the first few thousand generations but is quickly overtaken by cohort lexicase. Both cohort and down-sampled lexicase offer equivalent per-generation evaluation savings, but cohort lexicase uses every test case for parent selection in every generation. This could play a role in problem-solving success, as a test case that rewards exploration at any given activation position in the exploration diagnostic is used every generation. Indeed, populations evolving under cohort lexicase selection maintained a higher diversity of activation positions than populations evolving under down-sampled lexicase selection (Figure 5.8; Wilcoxon rank-sum test: $p < 10^{-4}$).

Previous work predicted the potential for such differences between cohort and down-sampled

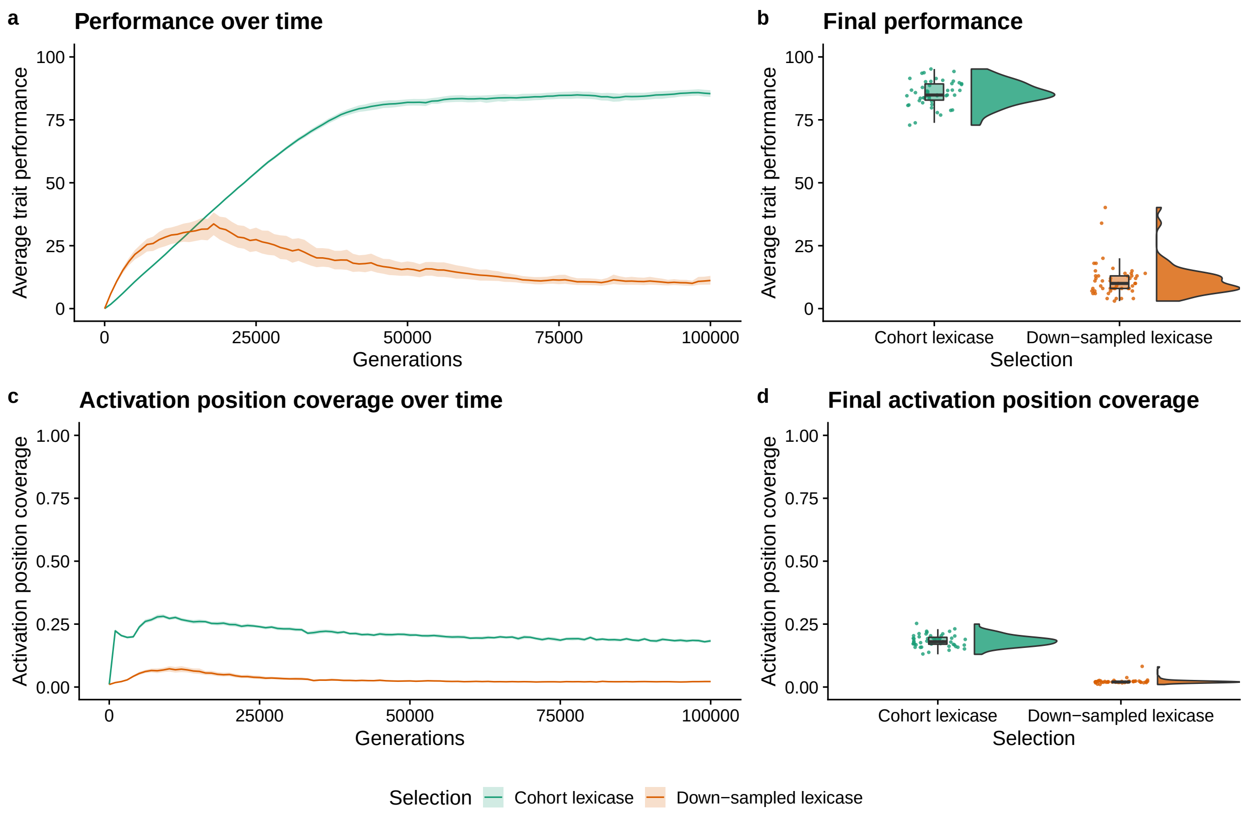

Figure 5.8: Down-sampled versus cohort lexicase on the exploration diagnostic. Panels (a) and (b) show performance over time and at the end of the experiment, respectively. Likewise, panels (c) and (d) show activation position coverage over time and at the end of the experiment, respectively. For panels (a) and (c), each line gives the mean value across 50 replicates, and the shading around each mean gives a 95% confidence interval.

lexicase. In Chapter 4, we found that cohort lexicase better maintained phylogenetic diversity than down-sampled lexicase, as phylogenies coalesced less frequently under cohort lexicase selection (maintaining deeper, more divergent branches). Despite this difference in diversity maintenance, we did not find significant differences in problem-solving success across a set of program synthesis benchmark problems, which suggests that the test cases used in these benchmark problems were more robust to random subsampling than the test cases for the exploration diagnostic. Indeed, each individual test case for the exploration diagnostic uniquely represents a single activation position; that is, test cases are minimally redundant with one another. In many program synthesis benchmark problems, however, individual test cases are often intentionally redundant to others, differing only in the particular values of their inputs and outputs and not necessarily different in the functional specialization they reward. Such redundancies prevent candidate solutions from memorizing particular input-output pairings, forcing candidate solutions to generalize in order to achieve high fitness across redundant test cases. This detail could explain why the exploration diagnostic reveals substantial performance differences between cohort and down-sampled lexicase where more standard benchmark problems failed to do so.

5.4.8 Novelty test cases degrade lexicase selection's exploratory capacity

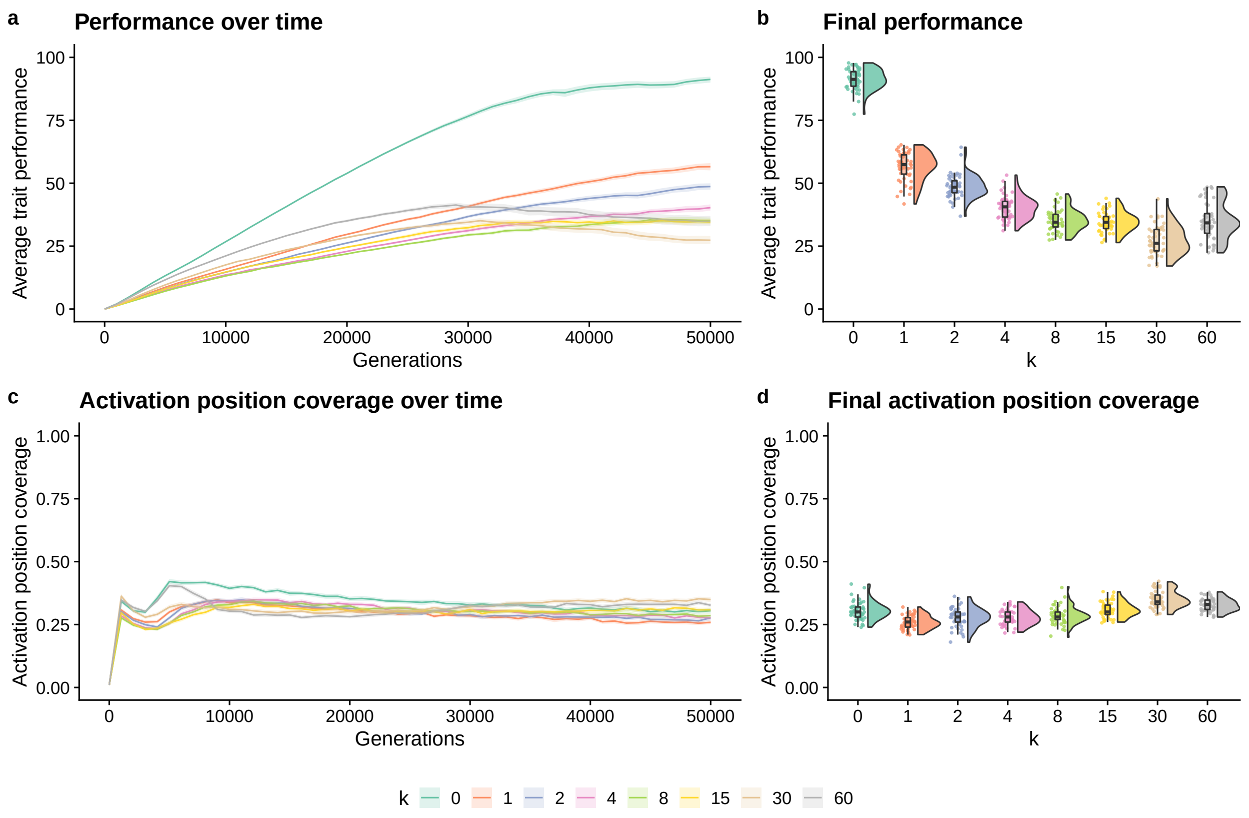

Figure 5.9: Novelty-lexicase selection's performance on the exploration diagnostic at a range of nearest-neighbor parameterizations. Panels (a) and (b) show performance over time and after 50,000 generations of evolution, respectively. Likewise, panels (c) and (d) show activation position coverage over time and after 50,000 generations of evolution, respectively. For panels (a) and (c), each line gives the mean value across 50 replicates, and the shading around each mean gives a 95% confidence interval.

Finally, we evaluated how incorporating novelty test cases into lexicase selection impacts exploration. We compared the performance of standard lexicase to that of novelty-lexicase for a range of k-nearest neighbors: 0 (standard lexicase), 1, 2, 4, 8, 15, 30, and 60.

Contrary to our expectations, we found that the addition of novelty test cases degrades performance on the exploration diagnostic in all cases (Figure 5.9; Wilcoxon rank-sum test: $p < 10^{-4}$). Though, novelty-lexicase generally maintains similar levels of activation position diversity in the population relative to standard lexicase, and by the end of the experiment, some parameterizations of novelty lexicase maintain more activation positions, though none of the differences appear to be substantial (Figure 5.9).

Novelty search favors solutions that have never been seen before, regardless of their impact on fitness. Based on previous studies, we expected novelty-lexicase to outperform standard lexicase on the exploration diagnostic (Jundt and Helmuth, 2019). However, novelty-lexicase appears to hinder lexicase's ability to fully exploit pathways in the diagnostic's search space.

While past work has demonstrated that novelty search can be effective at producing solutions for complicated problems, the exploration diagnostic does not have any of the hidden intricacies that novelty search excels at disentangling. Indeed, novelty search appears to thrive under conditions where there are more non-linearities between genotype and phenotype. The underlying representation used here is purposely simple numerical vectors, as opposed to an artificial neural network (Lehman et al., 2008) or PushGP (Jundt and Helmuth, 2019) where internal architectures can change and qualitatively different outputs are possible. For example, in this case, all sites in a genome are optimal at one end of their range of values, whereas most complex problems are assumed to have pockets of solutions throughout the genotype-phenotype map. Additionally, our results also used a single, limited form of novelty lexicase. We did not use a seed bank (the importance of which has previously been stressed), and we used k-nearest neighbors euclidean distances to measure novelty instead of a direct measure of behavioral uniqueness. These differences in problems may shine a light as to why novelty-lexicase did not outperform standard lexicase selection on the exploration diagnostic.

Our results from varying diagnostic cardinality (Section 5.4.2) may also offer insights into the unexpectedly poor performance of novelty-lexicase selection. Novelty-lexicase selection increases the number of test cases used for parent selection (in this work, doubling the number of test cases from 100 to 200). Increasing the number of test cases (without simultaneously increasing the population size) is not without cost, degrading specialist maintenance and performance on the exploration diagnostic (Figure 5.3). This dynamic is likely to be at play in our novelty-lexicase experiment, as population size was constant for both standard lexicase and novelty-lexicase selection.

5.5 Conclusion

In this work, we introduced a new diagnostic to investigate the exploratory limits of lexicase selection along with several of its variants: epsilon lexicase, down-sampled lexicase, cohort lexicase, and novelty-lexicase. First, we verified well-established expectations that lexicase selection better facilitates search space exploration than tournament selection. Across all exploration diagnostic difficulty levels (*i.e.*, cardinalities), lexicase selection drove improvements in performance (Figure 5.3), while tournament selection repeatedly failed to escape early local optima (Figure 5.2). As we increased the cardinality of the diagnostic, lexicase selection's specialist maintenance and overall performance waned. Conditions with larger diagnostic cardinalities used more test cases to evaluate individuals, and as such had more possible specialists (*i.e.*, niches). Given a fixed population size, lexicase maintained a smaller fraction of possible specialists as the number of possible niches increased, which, in turn, decreased overall performance (Figure 5.3).

Interestingly, we found that allocating a computational budget (*i.e.*, candidate solution evaluations) toward increasing generations versus increasing population size is not necessarily a straightforward choice when using lexicase selection. In our case, a larger population size enabled better specialist maintenance and ultimately higher performance on the exploration diagnostic with standard lexicase (Figure 5.4). This finding is interesting in light of Helmuth and Spector (2021)'s work investigating the problem-solving benefits of down-sampled lexicase; on a suite of program synthesis problems, Helmuth and Spector found that some problems benefited from an increased population size (at the cost of running for fewer generations), some problems benefited from an increase in generations, and most problems were unaffected by their choice of increasing population size versus generations evaluated.

Overall, these results suggest that lexicase selection can be sensitive to expanding the set of test cases used for evaluation, especially if each test case uniquely represents a distinct, desirable trait. Moreover, our results suggest the importance of more deeply examining the benchmark problems that we use and the characteristics of the search spaces that they represent. Given a fixed computational budget, why do some problems benefit from running deeper evolutionary searches while others benefit from increased population sizes under lexicase selection? For many problems, different categories of test cases have uneven representation in the test set. We hypothesize that the distribution of test cases among categories plays a role in lexicase selection's success and the optimal balance between population size and depth of search (generations of evolution). For example, if the number of test cases is similar to population size, lexicase selection may fail to maintain specialists on categories that are underrepresented in the test cases and instead favor overrepresented categories. In future work, we will develop novel diagnostic tools for investigating the sensitivity of selection schemes to test case set composition.

We found that each of the lexicase variants that we evaluated—epsilon lexicase, down-sampled lexicase, cohort lexicase, and novelty-lexicase—affected lexicase selection's exploratory capacity. For small values of ϵ, epsilon lexicase outperformed standard lexicase selection on the exploration diagnostic, while large values of ϵ substantially degraded performance. Surprisingly, we found that novelty-lexicase degrades performance on the exploration diagnostic relative to standard lexicase selection.

Our experiments are also the first to demonstrate consequential differences between down-sampled and cohort lexicase selection, as previous work generally failed to distinguish the problem-solving performance of these two lexicase variants (Chapter 4). Cohort lexicase substantially outperformed down-sampled lexicase (Figure 5.8). Both down-sampled and cohort lexicase offer equivalent per-generation evaluation savings, so our results suggest that cohort partitioning may often be a better subsampling method than down-sampling for lexicase selection. Future work should examine whether this difference between cohort partitioning and down-sampling holds across different selection schemes.

Given equivalent computational budgets, we found that standard lexicase selection eventually outperforms both cohort and down-sampled lexicase on the exploration diagnostic (Figures 5.6 and 5.7). This result diverges from recent benchmarking studies where subsampling substantially improved performance on a range of program synthesis problems (Helmuth and Spector, 2020, 2021). Future work will develop diagnostic problems to help identify when subsampling (*e.g.*, via either cohort partitioning or down-sampling) is likely to improve versus impede lexicase selection's performance.

In each of our experiments, we focused our analyses on performance and activation position diversity maintenance. Future work should more deeply examine the evolutionary histories of evolving populations using phylodiversity metrics (Dolson et al., 2020). Along with this, other parameter values and configurations of each of the variants evaluated in this work could be tested in order to develop a more complete understanding of how parameterization affects exploration.

We intend for this work to demonstrate how diagnostics (*e.g.*, the exploration diagnostic introduced here) can be valuable tools for evaluating the pros and cons of different selection schemes. We plan to implement a larger suite of selection scheme diagnostics, each targeted toward evaluating a particular aspect of problem-solving. Such diagnostics will complement conventional benchmarking experiments in our community's effort to understand how different selection schemes steer evolutionary search.

5.6 Data and Software Availability

Our supplemental material (Hernandez et al., 2021) is hosted on GitHub and contains the software, data analyses, and documentation associated with this work. Our experiments are implemented using the Empirical library (Ofria et al., 2020), and we used a combination of Python and R version 4 (R Core Team, 2020) for data processing and analysis. We used the following R packages for data wrangling, statistical analysis, graphing, and visualization: ggplot2 (Wickham et al., 2021), tidyverse (Wickham, 2019), knitr (Xie, 2020b), cowplot (Wilke, 2020), viridis (Garnier, 2018), RColorBrewer (Neuwirth, 2014), rstatix (Kassambara, 2021), ggsignif (Ahlmann-Eltze and Patil, 2021), Hmisc (Harrell, 2020), and kableExtra (Zhu, 2021). We used R markdown (Allaire et al., 2020) and bookdown (Xie, 2020a) to generate web-enabled supplemental material. Our experimental data is available on the Open Science Framework at https://osf.io/xpjft/ (Lalejini and Hernandez, 2021).

Chapter 6
A suite of diagnostic metrics for characterizing selection schemes

Authors: Jose Guadalupe Hernandez, Alexander Lalejini, and Charles Ofria

This chapter is adapted from (Hernandez et al., 2022c), which is a preprint being finalized for submission at the Evolutionary Computation journal.

In this work, we introduce three additional diagnostics: the exploitation rate diagnostic, the ordered exploitation diagnostic, and the contradictory objective diagnostic. We use the diagnostics to evaluate six popular selection schemes: truncation selection, tournament selection, fitness sharing, lexicase selection, nondominated sorting, and novelty search. We find results that are consistent with previous work and find key differences among the six selection schemes.

6.1 Introduction

Evolutionary algorithms have become an effective general-purpose technique for solving complex real-world optimization problems. Many different types of evolutionary algorithms exist, differing in selection schemes, representations, variation operators, and other factors. However, choosing which algorithm to use for a given problem—let alone configuring it—remains more of an art than a science (Jong, 1993). Numerous benchmarking suites are available to assess the strengths and weaknesses of these algorithms (Hansen et al., 2021; Jamil and Yang, 2013; Li et al., 2013b,c), but do so indirectly by focusing on success with exemplar problems. Here, we introduce a set of diagnostics that we crafted to highlight specific strengths and weaknesses of selection schemes. We diagnose a range of common selection schemes for their ability to exploit and explore four handcrafted search spaces with targeted properties, as well as their ability to manage contradictory objectives. Each diagnostic is designed to be lightweight and easily understood, allowing it to be evaluated quickly while producing results that are intuitive to interpret.

Selection schemes determine which individuals contribute genetic material to the next generation, thus driving an evolutionary algorithm's search strategy. Given that problems may differ in search space topology, strategies that are effective in one search space may be ineffective in another. Selection schemes vary in the criteria they use (e.g., problem-solving performance, genetic distinctness, phenotypic rarity, age, *etc.*) and how these criteria are used for selecting solutions (e.g., choosing values that are best, diverse, novel, *etc.*). Understanding how effective different selection schemes are for a given set of search space characteristics is crucial for making an efficient and productive choice when solving a particular problem. For example, evolving populations might need to exploit narrow gradients, balance conflicting objectives, deal with noise, or develop building blocks to scaffold complexity, all while minimizing computational costs. Different selection schemes balance different trade-offs of these capabilities in order to find high-quality solutions.

Benchmark suites provide the standard approach to understanding a selection scheme's overall problem-solving capabilities through an assortment of curated challenges. Generally speaking, these challenges can be classified into two broad categories: real-world problems and test functions (Jamil and Yang, 2013). Real-world problems are typical challenges that researchers encountered "in the wild" and used evolutionary algorithms to solve. These problems proved interesting, so they were chosen to provide insight into which problem domains a selection scheme is best suited for (Hussain et al., 2017; Jamil and Yang, 2013). Test functions, on the other hand, are usually created explicitly to test evolutionary algorithms. They are well-documented mathematical functions that are typically fast to evaluate and usually represent idealized versions of search spaces encountered in real-world problems (Hansen et al., 2021; Hussain et al., 2017; Jamil and Yang, 2013; Li et al., 2013b,c). Additionally, test functions are often tunable, which allows researchers to easily expose selection schemes to numerous scenarios.

Complexity in the techniques integrated within selection schemes has grown in order to contend with more challenging problems. Successful techniques increase the chances of finding high-quality solutions, but disentangling their individual impact on search may be counter-intuitive, leading to unexpected results in new problem domains or when multiple techniques are combined. While benchmark suites provide valuable high-level information about a particular problem domain, it is difficult to abstract a selection scheme's low-level characteristics. Thus, it is hard to predict how a selection scheme's efficacy will be altered by subtle changes to the structure of the search space or transfer to another domain. Most benchmark problems possess numerous integrated characteristics (e.g., modality, deception, separability, *etc.*) that each impact a selection scheme's problem-solving success. Unfortunately, the effects of each problem characteristic cannot be disentangled without extensive experimentation and analysis. We aim to be able to shine more light on the capabilities of any given selection scheme by testing it on a carefully constructed set of diagnostic problems. Each diagnostic problem is a simple test function that uses a handcrafted search space to isolate specific problem characteristics.

In this work, we evaluate six popular categories of selection schemes on a set of diagnostics: truncation selection, tournament selection, fitness sharing, lexicase selection, nondominated sorting, and novelty search. We propose four diagnostics, each requiring different degrees of exploitation and exploration to find high-quality solutions: (1) An exploitation rate diagnostic to measure a selection scheme's ability to exploit a smooth fitness gradient. (2) An ordered exploitation diagnostic to measure a selection scheme's ability to pursue a single, narrow gradient that leads toward a single global optimum. (3) A contradictory objectives diagnostic to measure a selection scheme's ability to locate and optimize conflicting objectives. (4) A multi-path exploration diagnostic to measure a selection scheme's ability to maintain and simultaneously exploit multiple gradients of differing fitness peaks. Ultimately, our diagnostics allow us to identify meaningful differences between the six chosen selection schemes.

6.2 Diagnostics

The "no free lunch" theorem states that no single optimization algorithm dominates all other optimization algorithms across all possible problem instances (Wolpert and Macready, 1997). Indeed, "no free lunch" holds in practice, as it is common to see an evolutionary algorithm excel in one problem domain but struggle in others. One key determinant of an evolutionary algorithm's success is the trade-off between exploitation and exploration it exhibits throughout an evolutionary search (Eiben and Schippers, 1998). The selection scheme heavily influences this trade-off, as it determines what regions of a search space to explore or continue evaluating. We constructed a set of handcrafted search spaces (diagnostics), each with calculated features; these diagnostics help us to disentangle how a selection scheme trade-offs between exploitation and exploration. Additionally, some diagnostics can be minor alterations of other ones; if a selection scheme performs differently on such diagnostics, that difference can be attributed to the specific alteration, isolating the effect. Ultimately, diagnostics help us develop a more comprehensive understanding of a selection scheme's strengths and weaknesses.

Determining the best selection scheme to use is one of the first challenges a practitioner encounters when trying to solve a new problem. Any given selection scheme uses a particular set of techniques and parameters that interact to determine a search strategy, and ultimately, its effectiveness at solving the given problem. For example, search strategies may vary in their trade-off between exploitation and exploration, both of which are crucial to finding high-quality solutions

(Eiben and Schippers, 1998; Črepinšek et al., 2013). A selection scheme may regulate this trade-off by choosing the highest quality solutions under consideration (for exploitation) or the most distinct solutions (for exploration). A selection scheme that is too exploitative will prematurely push the population toward the nearest optimum, missing out on higher peaks elsewhere. Conversely, a scheme that is too exploratory will spread the population across the search space, but might miss out on nearby optima. The ability to understand how each component and configuration of a selection scheme affects this trade-off is crucial, as the ideal trade-off will vary by search space, and even by the local characteristics within a region of search space.

We propose using our set of carefully constructed diagnostics to measure a selection scheme's exploitation and exploration capabilities. While benchmark suites provide valuable information, extracting precise details on low-level capabilities may be difficult because of problems with complex search space topologies. By handcrafting each diagnostic's search space, we avoid complex search space topologies in favor of intuitive and interpretable search spaces designed to challenge selection schemes with targeted problem characteristics. The problem characteristics of interest in this work include modality, deception, epistasis (interaction among genes), and dimensionality, all of which pose unique challenges (Malan and Engelbrecht, 2013; Sun et al., 2014; Weise et al., 2012). Our simplest diagnostic requires only the ability to climb a single, smooth hill, while others require a balance of exploitation and exploration to solve. Indeed, one of the diagnostics presented here is not even focused on problem-solving ability, but instead focuses solely on measuring the coverage of many contradictory objectives.

6.2.1 Diagnostic Design

Our diagnostics focus on isolating and measuring selection scheme characteristics that are critical for problem-solving success; however, many other design factors for evolutionary algorithms must be considered and ideally controlled for, including representation, variation operators, and population size. All of the diagnostics in this work assume a genome-based representation consisting of a sequence of floating-point values, each bound to the range of 0.0 to 100.0. In this work, we initialized populations near the lowest (least fit) point in the search space, genomes composed of random values ranging from 0.0 to 1.0. This constrained representation creates a well-defined search space that can be rigorously analyzed, yet intuitively understood. The difficulty of each diagnostic can be adjusted by changing the range and number of values (the "dimensionality") in each genome. We use 100 as the default dimensionality for this work.

Each diagnostic specifies a translation function of a candidate solution's genome into an evaluated numerical vector of the same dimensionality (its "phenotype"). We refer to each position in a candidate solution's genome as a "gene" and each position in a phenotype as a "trait". Selection schemes can either operate on traits independently (where each is treated as a single objective) or use the sum of all traits as a single fitness value. Many diagnostics define success as finding high-quality solutions, but some diagnostics focus on active diversity measures.

6.2.2 Exploitation Rate Diagnostic

Exploitation is a hill-climbing process that focuses on optimizing within a local neighborhood of a search space (Črepinšek et al., 2013). Indeed, in search spaces with a single, smooth peak (*e.g.,* sum of different powers functions (Molga and Smutnicki, 2005)), exploitation alone is sufficient to find the global optimum. Selection schemes that exploit effectively will steer populations toward nearby optima (Beyer, 1998). Exploitation can be especially important when evaluations require substantial resources (*e.g.,* compute time, memory, robotic hardware, *etc.*), and improvements to existing solutions need to be found using as few evaluations as possible. To measure a selection scheme's capacity for exploitation, we constructed the "exploitation rate" diagnostic, where a selection scheme must steer a population through a search space that is unimodal, non-deceptive, and has independent objectives.

In this diagnostic, a candidate solution's genome is directly interpreted as its phenotype (Figure 6.1). Each trait is maximized at the upper bound (100.0); a phenotype where all traits are maximized occupies the global optimum in the search space. Because there are no interactions among genes when computing a genome's phenotype, this diagnostic's search space can be viewed as comprising multiple smooth, non-deceptive gradients (one for each trait) that can each be optimized in parallel. Since we initialize the populations near the lowest point in the search space, the distance selection schemes must traverse to reach the global optimum is nearly maximized. By increasing this diagnostic's dimensionality, we can increase the number of independent gradients to be exploited, and by increasing the upper bound on gene values, we can tune the distance of each gradient.

While this search space may be trivial to solve, it does isolate a key problem-solving characteristic: exploitation rate. Ultimately, this diagnostic allows us to compare how well different selection

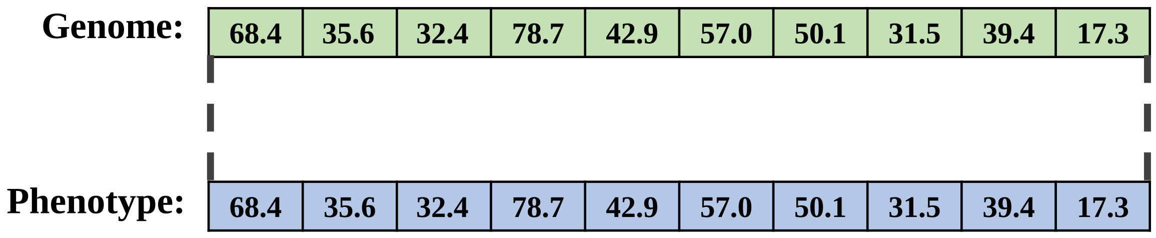

Figure 6.1: An example evaluation with the exploitation diagnostic. A candidate solution with a dimensionality of 10 is assessed. All genes are directly copied from the genome into the corresponding trait in the phenotype. The total fitness of the sequence is the sum of the traits in the phenotype or $68.4 + 35.6 + 32.4 + 78.7 + 42.9 + 57.0 + 50.1 + 31.5 + 39.4 + 17.3 = 453.3$.

schemes are able to exploit a smooth fitness gradient. As such, we expect exploitation-focused selection schemes (*e.g.*, truncation and tournament selection) to perform best on this diagnostic.

6.2.3 Ordered Exploitation Diagnostic

Many problems require sub-problems to be solved before an overall solution can be found. For example, to construct a multi-story building, the lower floors must be framed before starting higher floors. Framing one floor allows progress to be made on the next floor while still finishing the previous floor (*e.g.*, adding insulation, interior finishings, *etc*). In this example, the sub-problems—each floor–must be solved in order, and progress on lower floors must precede progress on higher floors. To measure selection schemes' capacity for such ordered optimization, we created the "ordered exploitation" diagnostic. This diagnostic extends the exploitation rate diagnostic, requiring that genes be optimized from start to finish, and sufficient progress must be made on previous genes before subsequent genes can be optimized.

In this diagnostic, genes are evaluated in order, starting from the beginning of a candidate solution's genome. The first gene is marked as "active", and each gene thereafter that is less than or equal to its predecessor is also marked as active. If a gene exceeds the value of its predecessor, that gene and all subsequent genes are marked as "inactive". We refer to the set of consecutive active genes as the "active region". All active genes are then directly interpreted as traits in the phenotype, and all inactive genes are interpreted as zero-valued traits in the phenotype (Figure 6.2). As in the exploitation rate diagnostic, each trait is maximized at the upper bound (100.0); a phenotype where all traits are maximized occupies the global optimum in the search space. Increasing the dimensionality increases the length of the narrow pathway to optimality, which allows us to expose

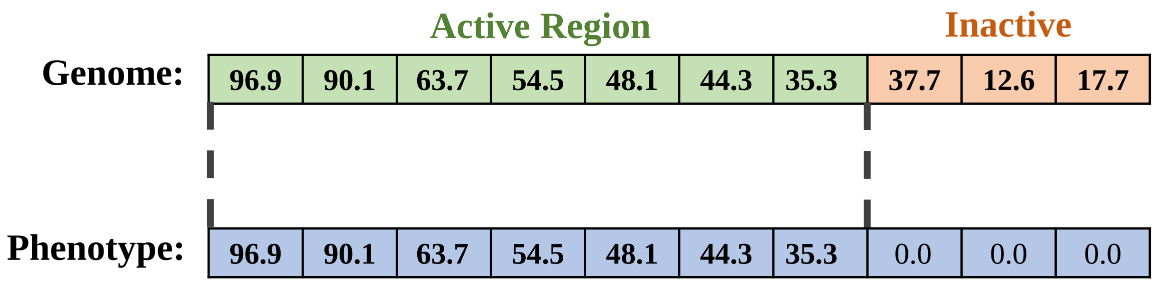

Figure 6.2: An example evaluation with the ordered optimization diagnostic. A candidate solution with a dimensionality of 10 is assessed. The first gene in its genome starts the active region. It and the next six gene values are all in a non-increasing sequence $(96.9, 90.1, 63.7, 54.5, 48.1, 44.3,$ and $35.3)$ and are thus all considered part of the active region. The first gene value after the active region (37.7) is greater than its predecessor, thus marked inactive, closing the active region. All genes marked as active are directly expressed as traits in the phenotype, and all remaining genes are interpreted as zero-valued traits. The total fitness of the sequence is the sum of the traits in the phenotype or $96.9 + 90.1 + 63.7 + 54.5 + 48.1 + 44.3 + 35.3 + 0.0 + 0.0 + 0.0 = 432.9$.

selection schemes to more extreme, yet similar, scenarios.

Intuitively, the ordered exploitation diagnostic requires selection schemes to guide populations through a search space with a single, narrow gradient toward the global optimum. This diagnostic extends the exploitation rate diagnostic by isolating a selection scheme's ability to perform ordered exploitation where genes must be optimized in a particular order. We hypothesize that selection schemes that focus on exploiting neighborhoods of high-performing solutions will excel in this diagnostic, as solutions are not rewarded for exploring outside of the narrow pathway to optimality. Additionally, we expect that selection schemes that perform well on the exploitation rate diagnostic will also perform well on this diagnostic.

6.2.4 Contradictory Objectives Diagnostic

The previous diagnostics each have a single global optimum, and for those diagnostics, we focus on a selection scheme's ability to steer populations to that optimum. For this diagnostic, however, we focus on *how many* global optima a selection scheme can find and maintain in a population (*i.e.*, "trait coverage"), providing insights on a scheme's ability to exhibit meaningful diversity. Such diversity maintenance is especially important for optimization problems with multiple contradictory objectives, as there is no single optimum for problems with this characteristic. Generating and maintaining a population with meaningful diversity can increase the chances of finding high-quality solutions by simultaneously exploring many distinct pathways through the search space and thus

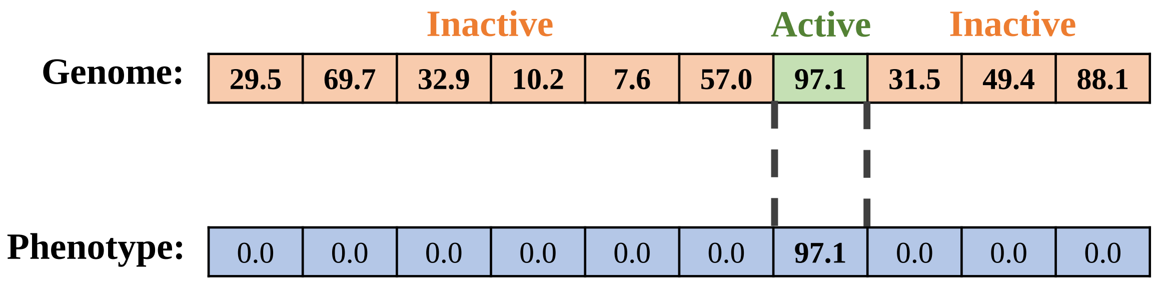

Figure 6.3: An example evaluation with the contradictory objectives diagnostic. A candidate solution with a dimensionality of 10 is assessed. The highest gene in its genome is identified as 97.1 and set as active, where the gene value is set in the phenotype. All other positions in the solution's genome are marked as inactive and expressed as 0.0 in the phenotype.

reducing premature convergence (Blickle and Thiele, 1995; Squillero and Tonda, 2016; Sudholt, 2020; Črepinšek et al., 2013).

To evaluate a genome, we identify the gene with the greatest value (ties are broken by choosing the gene closest to the beginning of the genome), and we mark that gene as active. All other genes are marked as inactive. The single active gene is directly interpreted as the associated trait in the phenotype, and all inactive genes are interpreted as zero-valued traits (Figure 6.3). A trait is maximized at the upper bound (100.0). There are many global optima in the search space—one for each trait in a candidate solution's phenotype (i.e., the diagnostic dimensionality); each optimum is associated with a single maximized trait.

Selection schemes must balance exploration to discover the many gradients in the search space and exploitation to follow gradients to their peak, while simultaneously preventing the population from collapsing onto a single gradient. Thus, we expect selection schemes that balance exploitation with diversity maintenance to maintain populations with many global optima.

6.2.5 Multi-path Exploration Diagnostic

The ideal trade-off between exploitation and exploration varies by optimization problem and even by local regions of a search space. This trade-off is especially true for problems with many local optima, each with a different peak fitness; exploration can help populations discover multiple gradients, and exploitation helps populations reach each of their peaks. In fact, simultaneously seeking multiple optima will often increase the chance of finding better-performing solutions. Given this common problem characteristic, we include a diagnostic that examines the ability of selection

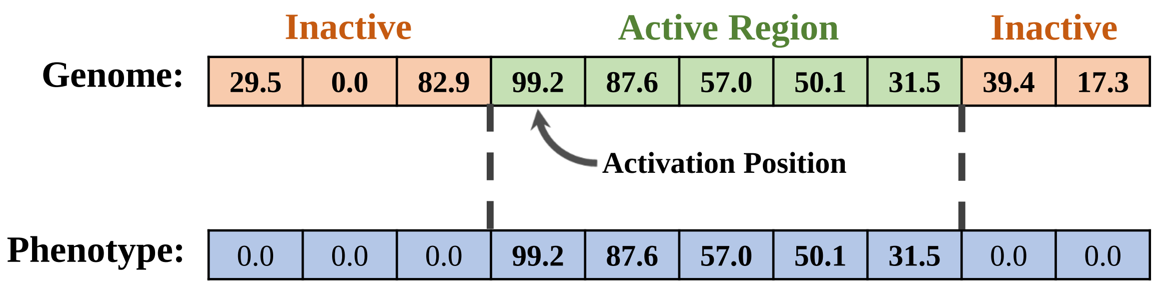

Figure 6.4: An example evaluation with the multi-path hill climbing diagnostic. A candidate solution with a dimensionality of 10 is assessed. The highest value in its vector is identified as 99.2, and its position is marked as the start of the active region. The next four values are all in a non-increasing sequence (87.6, 57.0, 50.1, and 31.5) and are thus all considered part of the active region. The value after the active region (39.4) is greater than its predecessor, thus marked inactive, closing the active region. All values not marked as active are expressed in the phenotype as 0.0, and all values in the active region are set in the phenotype. The total fitness of the sequence is the sum of the traits in the phenotype or $0.0 + 0.0 + 0.0 + 99.2 + 87.6 + 57.0 + 50.1 + 31.5 + 0.0 + 0.0 = 325.4$.

schemes to explore multiple avenues of a search space.

To evaluate a genome, we first mark the gene with the greatest value as the "activation position". Starting from this activation position, we mark all consecutive genes that are less than or equal to the previous gene as active, creating an active region, and move their values to the associated phenotypic traits (Figure 6.4). All genes outside of the active region are marked as inactive and are interpreted as zero-valued traits in the phenotype. Traits are maximized at the upper bound (100.0), and a phenotype where all traits are maximized occupies the global optimum in the search space.

Intuitively, the search space consists of multiple pathways (the number of which is determined by the dimensionality used) differing in path length and peak height but identical in slope. The fact that pathways are initially indistinguishable means that the potential of any given pathway can be determined only by traversing it to its end. Since all pathways terminate at the end of a genome, the activation position specifies which pathway it occupies in the search space. As such, the pathway beginning at the first position in the genome leads to the global optimum. This diagnostic measures how well a selection scheme can simultaneously explore multiple pathways (like the contradictory objectives diagnostic) and pursue narrow pathways (like the ordered exploitation diagnostic). Note pathways are synonymous with gradients.

The multi-path exploration diagnostic has already proven to be a valuable tool for analyzing selection schemes. In Chapter 5, we used this diagnostic to produce actionable recommendations on how to maximize the exploratory capacity of lexicase selection and several of its variants. The size of a population typically determines the number of evaluations that a selection scheme needs to perform each generation. As such, the computational budget of an evolutionary algorithm is typically proportional to the population size times the number of generations run. We demonstrated that the total computational budget available determined the best trade-off between population size and number of generations in order to maximize evolved solution quality. With a large computational budget, larger populations allowed for greater diversity maintenance and improved problem-solving potential; with a small computational budget, smaller populations outperformed larger populations, as small populations could be evolved for more generations to better exploit local regions of the search space. Additionally, this diagnostic was the first technique to reveal consequential differences between down-sampled and cohort lexicase selection, showing that cohort lexicase can better facilitate search space exploration. This diagnostic has also been used to demonstrate that measures of phylogenetic diversity can provide meaningfully different information about an evolving population than measures of phenotypic diversity (Hernandez et al., 2022a).

6.3 Methods

We conducted four sets of experiments, each experiment comparing how different selection schemes react to the search space characteristics embodied by one of our diagnostics. We compared the following commonly used selection schemes: truncation selection, tournament selection, fitness sharing, lexicase selection, nondominated sorting selection, and novelty search. For each experiment, we performed eight treatments: two associated with fitness sharing (one using genotypic similarity and another using phenotypic similarity), one treatment associated with each of the five remaining selection schemes, and a random control. Within each treatment, we performed 50 replicates; for each replicate, we evolved populations of 512 individuals for $50,000$ generations. In each case, we used the target selection scheme to identify 512 parents, fixing all other factors, such as starting conditions, variation operators, reproduction, *et cetera.*

We initialized all populations near the lowest point in each of the diagnostics' search space, genomes composed of random values ranging from 0.0 to 1.0. For each generation, we evaluated each candidate solution's genome according to the given diagnostic, and we used the resulting phenotypes to select individuals to serve as parents for the next generation. Selected candidate solutions reproduced asexually, and we applied mutations to offspring. We mutated individual genes at a per-gene rate of 0.7%, where the magnitude of each mutation is obtained from a normal distribution with a mean of 0.0 and a standard deviation of 1.0 ($\mathcal{N}(0.0, 1.0)$). If a mutation would cause a gene to drop below the lower bound (0.0), we rebound it to its absolute value (*i.e.*, a would-be gene value of -0.7 becomes 0.7). If a mutation would cause a gene to exceed the upper bound (100.0), we again rebound it by the amount it would have exceeded the limit (*i.e.*, a mutation to 100.7 instead becomes 99.3).

We intentionally limited variation operators to point mutations to prevent large-effect genetic changes (e.g., as a result of crossover). While it is common in practice to use crossover, more dramatic sources of variation could "jump" offspring to different regions of the search space than their parent. For the purposes of this study, we focus on a selection scheme's ability to iteratively traverse a search space. In future work, we will investigate the effect of crossover on how selection schemes steer populations through search spaces.

6.3.1 Selection Schemes

In this work, we diagnose the following six selection schemes. We selected these selection schemes because of their popularity and demonstrated effectiveness on different optimization problems.

Truncation Selection

Truncation selection uses the top performing ("elite") individuals in a population as parents to generate the next generation. This selection scheme is the simplest and most direct way to identify high-quality parents; it is the basis for most animal husbandry (Crow and Kimura, 1979) and is used widely within evolutionary computation (Beyer and Schwefel, 2002; Luke, 2013).

At the start of the selection step, all candidate solutions are assigned a single fitness value; for our diagnostics, we use the sum of all of the individual's traits. The population is then sorted by fitness (with ties settled randomly) and then truncated, leaving only the top t performers to be used as parents for the following generation. Selected parents produce equal numbers of offspring, so that the next generation contains the correct number of candidate solutions. In this work, we use $t = 8$, meaning that after truncation, each of the parents creates 64 offspring.

This selection scheme is heavily exploitive, as the population is truncated and only a designated number of top performers are used as parents. The level of truncation, t, dictates the strength of selection. As t approaches 1, the selection pressure is increased as only a small number of high-fitness solutions are used as parents. Conversely, increasing t reduces selection pressure, as a greater number of individuals with lower fitness are used as parents. We hypothesized that truncation selection (as configured here) would excel on diagnostics that focus on exploitation.

Tournament Selection

Tournament selection is one of the earliest and most commonly used selection schemes in evolutionary computing (Brindle, 1980; Goldberg and Deb, 1991; Luke, 2013). In tournament selection, each parent is chosen by first picking t random candidate solutions from the population. Each candidate solution in this "tournament" is assigned a single fitness value; for our diagnostics, we use the sum of all of the individual's traits. The individual with the highest fitness is selected (with ties settled randomly) as a parent. The number of tournaments determines the number of parents identified: thus, 512 (population size) tournaments are held given our setup.

The tournament size, t, determines the strength of selection. As the tournament size approaches the population size, only the most fit individuals in the population are able to win tournaments and be chosen as parents. Conversely, as the tournament size approaches 1, tournament selection behaves more like random selection, allowing lower-fitness individuals to be chosen as parents. For our work, we set the tournament size to an intermediate size of $t = 8$. We hypothesized that tournament selection under this configuration will perform well on diagnostics focused on exploitation, but struggle with diagnostics that require substantial exploration to find high-quality solutions.

Fitness Sharing

Fitness sharing incorporates an explicit mechanism for maintaining a diverse population in order to reduce the likelihood of premature convergence (Goldberg and Richardson, 1987). At the start of the selection step, all candidate solutions are assigned a single fitness value and a similarity to each other individual. For our diagnostics, we use the sum of all of the individual's traits to represent a candidate solution's fitness value. The similarity metric can be either genotypic (e.g., the hamming distance between genomes) or phenotypic (e.g., the number of traits two individuals have in common). Fitness sharing then modifies each individual's fitness value, decreasing it as a function of its similarity to the rest of the population; individuals occupying crowded regions of the search space have their fitness reduced more than those in less crowded regions.

Consider candidate solution x with f_x representing the sum of all of its trait values after being evaluated on a diagnostic. The shared fitness f'_x of solution x is given by

$$f'_x = \frac{f_x}{m_x}$$

where m_x quantifies a candidate solution x's fitness reduction due to its similarity to the rest of the population. In this work, we use two versions of fitness sharing, one that uses genotypic similarity and one that uses phenotypic similarity, which we refer to as genotypic fitness sharing and phenotypic fitness sharing. For each, we calculate m_x as

$$m_x = \sum_{y \in P} S(d_{xy})$$

where P is the current population, $S()$ is the sharing function, and d_{xy} is the euclidean distance between genotypic or phenotypic vectors for candidate solutions x and y. The sharing function $S()$ uses a candidate solution's distance from another solution to set the associated fitness penalty, if any:

$$S(d) = \begin{cases} 1 - (\frac{d}{\sigma})^{\alpha}, & \text{if } d < \sigma \\ 0, & \text{otherwise} \end{cases}$$

Two variables are required to configure the sharing function: α and σ. The parameter α regulates the shape of the sharing function and σ determines the threshold of dissimilarity beyond which no penalty should exist.

We use $\alpha = 1.0$ and $\sigma = 0.3$ for all replicates in this work. We selected this α value due to it being commonly used in the literature (Goldberg and Richardson, 1987; Sareni and Krahenbuhl, 1998a) and we empirically identified a generally effective σ value (Hernandez et al., 2022d). Once all solutions have their shared fitness assigned, the stochastic remainder selection with replacement described in Section 2 of (Haq et al., 2019) is used to identify parents, as it is recommended to pair fitness sharing with stochastic remainder selection (Goldberg and Richardson, 1987; Sareni and Krahenbuhl, 1998b).

We expect fitness sharing to perform better on those diagnostics that require more exploration, but poorly on diagnostics focused exclusively on exploitation (e.g., those where narrow pathways must be traversed). Even more so, the choice of genotypic versus phenotypic distance metrics will play a big role in problems where these values differ. If there are smooth phenotypic pathways to the global optimum, we expect that phenotypic distance metrics will perform better, while if a more exhaustive exploration of the fitness landscape is critical, genotypic distance should be preferred.

Lexicase Selection

Lexicase selection is a technique designed for genetic programming problems where solutions must perform well across multiple test cases (Helmuth and Abdelhady, 2020; Helmuth et al., 2015; Orzechowski et al., 2018). The previously described selection schemes focus on maximizing total trait values (truncation, tournament selection) or promoting rare trait values (fitness sharing). By contrast, lexicase selection selects for individuals that specialize on different *combinations* of high traits by iterating through shuffled sets of test cases, resulting in high levels of stable diversity (Dolson and Ofria, 2018; Helmuth et al., 2016a, 2020).

In lexicase selection, all candidate solutions are evaluated on a set of test cases, and their performance on each test case is recorded. In our diagnostics, we associate one test case with each possible trait in an individual's phenotype (resulting in 100 test cases). We use each trait value in an individual's phenotype as a direct measure of performance for the associated test case. In order to identify a parent for replication, lexicase selection shuffles the set of test cases and iterates through each test case in sequence. Starting from the full population, each test case (in shuffled order) is used to filter down the current set of candidate parents; only those solutions tied for best performance on a given test case are allowed to continue. This filtering process continues until a single candidate parent remains or all test cases have been used. The single remaining candidate becomes a parent or a random one is selected if multiple candidates remain.

We hypothesized that lexicase selection would excel across multiple diagnostics, as lexicase selection is able to balance both exploitation and exploration. Details about its ability to explore and exploit can now be easily compared to other selection schemes that focus on exploration and exploitation differently.

Nondominated Sorting

The nondominated sorting genetic algorithm (NSGA) (Srinivas and Deb, 1994) and its descendants (Deb et al., 2002; Yuan et al., 2014) are successful evolutionary multi-objective optimization techniques. Evolutionary multi-objective optimization methods aim to generate a set of solutions that represent the best possible trade-offs among multiple (often conflicting) objectives (Coello Coello et al., 2020). NSGA combines two procedures during selection: a ranking procedure that groups individuals into nondominated fronts and fitness sharing for diversity maintenance within each group. Given its proven success in multi-objective contexts, we included the selection procedure used in NSGA in our study.

We use the set of phenotypes produced by our diagnostics to identify whether or not each solution is dominated in a given population. Given two phenotypes x and y, we say that *x dominates y* if all of x's traits are greater than or equal to y's traits and at least one of x's trait is strictly greater than the corresponding trait in y. Note that these diagnostics assume maximization problems.

The first nondominated front is created by collecting all candidate solutions that are not dominated by any other solution in the population. Once the first front is constructed, all solutions in the first nondominated front are assigned a large fitness value (Hernandez et al., 2022d). Fitness sharing is then applied to the solutions within the first front; the same procedure used to calculate shared fitness is found in section 6.3.1 with phenotypic similarity. Each subsequent nondominated front is constructed by removing solutions in previous fronts and then finding the next set of nondominated solutions from the current population. As each front is constructed, a starting fitness value is selected that is lower than all shared fitness values from the previous front; fitness sharing is again applied within this new group and this cycle continues until all solutions in the population are placed into a front. Finally, once all solutions have their shared fitness assigned, the stochastic remainder selection paired with fitness sharing in section 6.3.1 is used to identify parents, as Srinivas and Deb (1994) use stochastic remainder selection.

Given that this selection scheme focuses on generating multiple Pareto-optimal solutions, we hypothesized that this scheme would perform well on diagnostics that possess multiple global optima and focus on exploration. Conversely, we expect this selection scheme will struggle with hill-climbing, since this scheme focuses on exploring across the entirety of the current Pareto front.

Novelty Search

Novelty search mitigates complications associated with objective functions (*e.g.,* deception and local optima) by abandoning traditional fitness-based objectives. Instead, it uses a novelty metric to quantify how behaviorally distinct solutions in the population are from one another. Novelty search then uses the resulting novelty score to preferentially select solutions with trait combinations distinct from those previously observed (Lehman et al., 2008), encouraging productive exploration even without an obvious path to optimality.

We use the phenotype returned by our diagnostics to represent the set of behaviors used for measuring novelty. Consider a phenotype x after being evaluated on some diagnostic. The novelty score of x is given by

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x, u_i)$$

where $\rho(x)$ is the novelty score of x and u_i is the $i-$th nearest neighbor of x with respect to phenotypes. All calculations of nearest neighbor phenotypes include both the current population and an archive of all novel phenotypes that were previously found. For this work, we used the euclidean distance between two phenotypes as the distance metric. We also set $k = 15$, as it recommended in (Lehman et al., 2008). Once the novelty scores are calculated for all candidate solutions, we used tournament selection with size two to identify parents for the following generation as in Lehman and Stanley (2010) and Jundt and Helmuth (2019).

Since novelty search is focused on finding phenotypes that were never previously encountered, maintaining an unbounded archive is important. We use a threshold *pmin* to determine whether a phenotype is sufficiently novel to be tracked by the archive. In this work, *pmin* is set to 10.0. Furthermore, approximately one phenotype is randomly saved to the archive every 200 generations. If more than 4 phenotypes enter the archive by being more novel than *pmin* in one generation, *pmin* is increased by 25%. If no new phenotypes are added to the archive for 500 generations, *pmin* is decreased by 5%. This configuration closely follows the novelty search used in (Lehman et al., 2008).

We expect that novelty search will perform poorly (compared to all other objective-based schemes) on diagnostics where a single gradient leads to high-quality solutions. The diagnostics that incorporate deception and multiple optima should be advantageous for the novelty search algorithm, but given its complexity, it is difficult to predict its performance.

6.3.2 Statistical Analysis

We performed a KruskalWallis test to determine if significant differences among selection schemes occurred. For comparisons where the Kruskal-Wallis test was significant (significance level of 0.05), we performed a post-hoc Wilcoxon rank-sum test between relevant schemes with a Bonferroni correction for multiple comparisons where appropriate. We note that because novelty search uses an archive to track novel behaviors, we also consider archive solutions when gathering data.

Data Tracking

The diagnostics for exploitation rate and ordered exploitation measure a selection scheme's ability to hill-climb; thus, we measure the quality of a search by the best-performing solution ever found. We report the *performance* of an individual as its average trait score, which is the sum of individual scores divided by the dimensionality (100), resulting in values between 0.0 and 100.0. Additionally, we record the generation a *satisfactory solution* is first discovered within the population. We define an individual trait to be satisfactory if it has a value greater than or equal to 99% of the target value (100.0); if all traits in an individual are satisfactory, we designate that individual as a satisfactory solution.

The contradictory objectives diagnostic measures the number of mutually exclusive global optima that a selection scheme can simultaneously maintain in a population. We track both *population-level satisfactory trait coverage*, which is the number of unique satisfactory traits found across all individuals in a population, and *population-level activation gene coverage*, which is the number of unique activation genes found across all individuals in a population, regardless of whether a satisfactory trait has been obtained. Given only one gene can be active within a genome for this diagnostic, we label that gene as the activation gene. Activation gene coverage measures a selection scheme's capacity to produce and maintain a diverse set of phenotypes within a population. Satisfactory trait coverage measures selection schemes' ability to simultaneously exploit mutually-exclusive traits. Note that both have values between 0 and 100 (dimensionality).

The multi-path exploration diagnostic focuses on a selection scheme's ability to explore multiple gradients, only one of which leads to the global optimum. We track the average trait score of the best-performing solution found each generation, as it tells us a selection scheme's ability to exploit a gradient. Additionally, we track population-level activation gene coverage, as it measures a selection scheme's capacity to pursue a diverse set of gradients.

6.3.3 Software Availability

We include supplemental material (Hernandez et al., 2022d) that is hosted on GitHub and contains the software, data analyses, and documentation for this work. Our experiments are implemented using the Empirical library (Ofria et al., 2020), and we used a combination of Python and R version 4 (R Core Team, 2020) for data processing and analysis. The following R packages are used for data wrangling, statistical analysis, graphing, and visualization: ggplot2 (Wickham et al., 2021), tidyverse (Wickham, 2019), cowplot (Wilke, 2020), reshape2 (Wickham, 2007) and dplyr (Wickham et al., 2020). Our experimental data is available on the Open Science Framework at https://osf.io/5nv86/.

6.4 Results and Discussion

6.4.1 Exploitation Rate

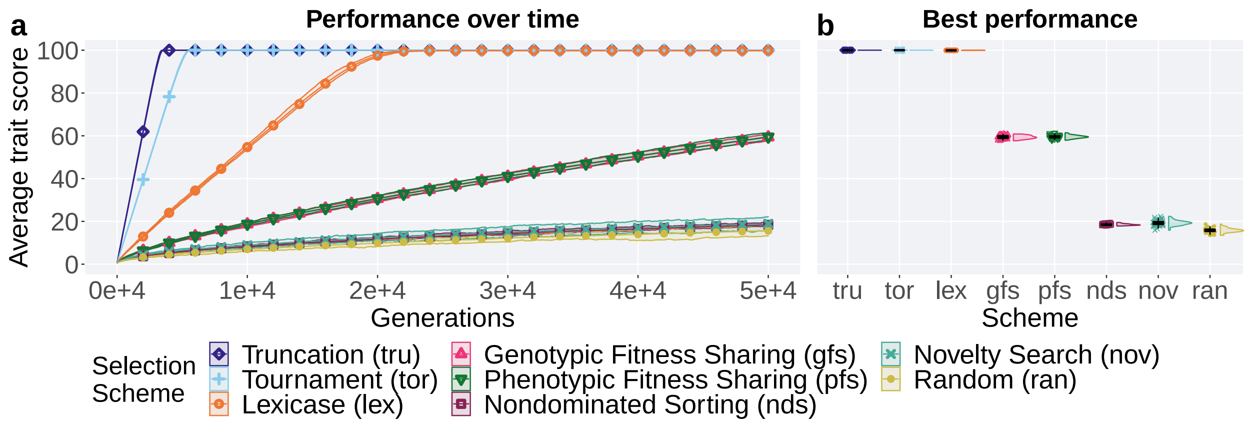

Figure 6.5: Results for selection schemes evaluated on the exploitation rate diagnostic. Best performance in the population (a) over time and (b) earned throughout 50, 000 generations. For panel (a), we plot the average across the 50 replicates, with shading between the maximum and minimum performance.

Using the exploitation rate diagnostic, we compared the relative ability to exploit a smooth fitness gradient among selection schemes. We found that all selection schemes improve performance over time. The two fitness sharing treatments produced identical results for this diagnostic; in all other cases, the rate of improvement differs between schemes (Figure 6.5a). Additionally, we found that all selection schemes outperformed the random control when comparing the best performance earned throughout an evolutionary run (Figure 6.5b; Wilcoxon rank-sum test: $p < 10^{-14}$).

Notably, truncation, tournament, and lexicase selection are the only schemes that found satisfactory solutions (phenotype with traits greater than or equal to 99.0), doing so in all replicates. For the configurations used here, truncation selection found satisfactory solutions in fewer generations than tournament selection (Wilcoxon rank-sum test: $p < 10^{-15}$). This result is interesting, as both selection schemes share the same asymptotic takeover time (Bäck, 1996; Goldberg and Deb, 1991), suggesting they exhibit similar selection pressure. Additionally, tournament selection found satisfactory solutions in fewer generations than lexicase selection (Wilcoxon rank-sum test:

$p < 10^{-15}$). Evidently, truncation and tournament selection find satisfactory solutions faster than lexicase selection due to maximizing an aggregate score, whereas lexicase selection pressures the population to be best at multiple test case combinations. Lexicase selection's diversity maintenance does not provide any explicit advantage to solutions with larger aggregate scores, slowing down the rate of exploitation.

Given the exploitation rate diagnostic directly translates a genotype into a phenotype, there is no procedural difference between genotypic or phenotypic fitness sharing. Indeed, no statistical difference is found between their best performances earned throughout an evolutionary run (Wilcoxon rank-sum test: $p > 0.05$). As populations can maximize traits in any order, fitness sharing penalizes the performance of individuals maximizing similar traits, lowering the chances of solutions with larger aggregate scores from being selected. This reduction in performance is exacerbated with higher aggregate scores, as genomes will become more similar as they approach the optimum. Nonetheless, both fitness sharing treatments outperform nondominated sorting and novelty search (Wilcoxon rank-sum test for best performance: $p < 10^{-15}$). The rate at which performance increases can be explained by the takeover time associated with stochastic remainder selection, as it is slower than truncation and tournament selection (Bäck, 1996; Goldberg and Deb, 1991).

Both nondominated sorting and novelty search perform poorly, as neither emphasizes exploitation in their search strategy. In fact, NSGA-II extends the nondominated sorting in NSGA by incorporating elitism (Deb et al., 2002), and other implementations of novelty search incorporate mechanisms to increase exploitation (Lehman and Stanley, 2011b). Interestingly, novelty search found better-performing solutions than nondominated sorting throughout an evolutionary search (Wilcoxon rank-sum test: $p < 10^{-3}$).

6.4.2 Ordered Exploitation

Using the ordered exploitation diagnostic, we compared the relative ability to pursue a single, narrow gradient toward the global optimum among selection schemes. We found that all selection schemes improve performance over time, but the rate of improvement differs between all schemes. Additionally, all selection schemes outperform the random control when comparing the best performance earned throughout an evolutionary run (Figure 6.6b; Wilcoxon rank-sum test: $p < 10^{-15}$).

Notably, truncation, tournament, and lexicase selection are the only schemes that found satis-

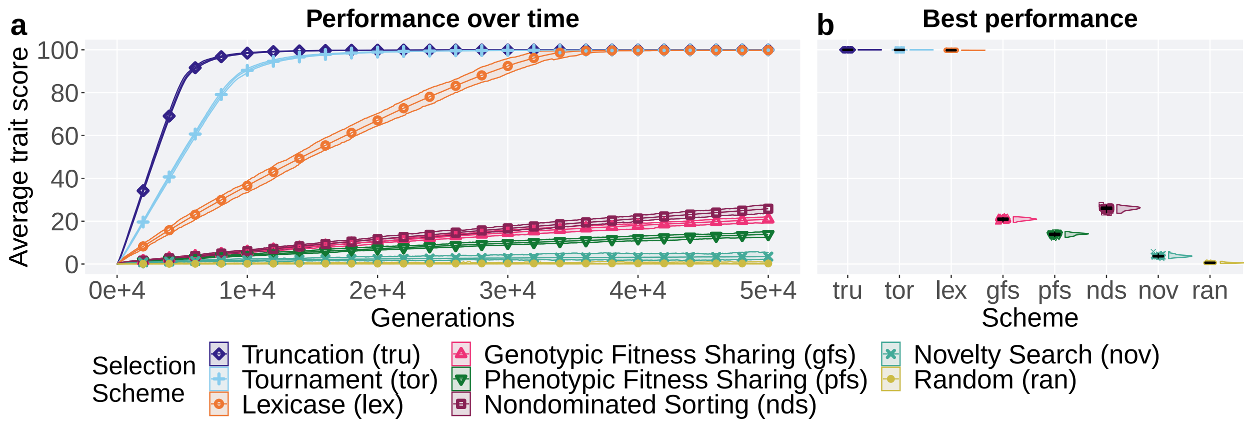

Figure 6.6: Results for selection schemes evaluated on the ordered exploitation rate diagnostic. Best performance in the population (a) over time and (b) earned throughout $50,000$ generations. For panel (a), we plot the average across the 50 replicates, with shading between the maximum and minimum performance.

factory solutions (phenotypes with traits greater than or equal to 99.9), doing so in all replicates. For the configurations used here, tournament selection found satisfactory solutions in fewer generations than lexicase selection, while truncation found satisfactory solutions in fewer generations than both selection schemes (Wilcoxon rank-sum test for both comparisons: $p < 10^{-15}$), similar to our results from the previous diagnostic. For this diagnostic, building off the best-performing solution's genome will lead to high-performing solutions much quicker, which coincides with truncation and tournament selection's search strategy. One could have easily guessed that the test case shuffling lexicase selection implements would complicate this diagnostic, yet lexicase selection still found satisfactory solutions.

Our results illustrate that fitness sharing is detrimental for exploitative search spaces. Indeed, we find additional evidence of this within the results for the previous diagnostic. Clearly, pressuring the population to explore sparse regions of the genotype or phenotype space was not beneficial for exploitative search spaces. Interestingly, genotypic fitness sharing found better-performing solutions than phenotypic fitness sharing (Figure 6.6b; Wilcoxon rank-sum test: $p < 10^{-15}$). We suspect this occurred because while early portions of a genome were being optimized, the genotype of the later, non-active regions could drift when genomes were being compared for fitness sharing, but would always be zero when phenotypic traits were compared. As such, genotypic fitness sharing's ability to minimize similarity allows it to outperform phenotypic fitness sharing.

Nondominated sorting found better-performing solutions than both fitness sharing configurations and novelty search (Figure 6.6b; Wilcoxon rank-sum test: $p < 10^{-15}$). The previous diagnostic provides evidence that nondominated sorting is able to climb a single gradient, even though it does so at a slower rate compared to the other schemes. We suspect nondominated sorting's performance can be explained by its focus on finding multiple Pareto-optimal solutions. Nondominated sorting pressures the solutions to be nondominated, as solutions in early nondominated fronts have higher fitness. This pressure favors unlocking a new active gene, thus the population is pressured to increase their streaks of active genes. Long streaks of active genes are found in high-performing solutions, as better-performing solutions can be reached, which helps explain its performance. Indeed, we find that nondominated sorting is able to find solutions with longer streaks of active genes than both fitness sharing configurations and novelty search (Wilcoxon rank-sum test: $p < 10^{-15}$; Hernandez et al. (2022d)).

Finally, as expected, novelty search performs poorly due to not emphasizing exploitation in its search strategy. Given enough time, however, we might expect novelty search to find solutions by exhaustively enumerating the search space.

6.4.3 Contradictory Objectives

The contradictory objectives diagnostic limits each individual's phenotype to specializing on a single trait, allowing us to compare the relative ability to locate and optimize conflicting objectives across selection schemes. Specifically, we compared population-level satisfactory trait coverage (*i.e.*, the number of distinct satisfied traits across the whole population) and population-level activation gene coverage (*i.e.*, the number of distinct activation genes maintained in the population).

All selection schemes, except novelty search and the random control, satisfied at least one trait after $50,000$ generations (Figure 6.7b). All populations evolved under truncation selection, tournament selection, and genotypic fitness sharing covered exactly one satisfactory trait; that is, these selection schemes never produced populations with more than one unique satisfactory trait. Phenotypic fitness sharing, lexicase selection, and nondominated sorting consistently produced populations with more than one unique satisfactory trait. For the configurations used here, nondominated sorting attained more satisfactory traits than lexicase selection, and lexicase selection attained more satisfactory traits than phenotypic fitness sharing (Wilcoxon rank-sum tests: $p < 10^{-15}$). We found that all selection schemes, except novelty search, attained more satisfactory traits than our random control (Wilcoxon rank-sum tests: $p < 10^{-15}$).

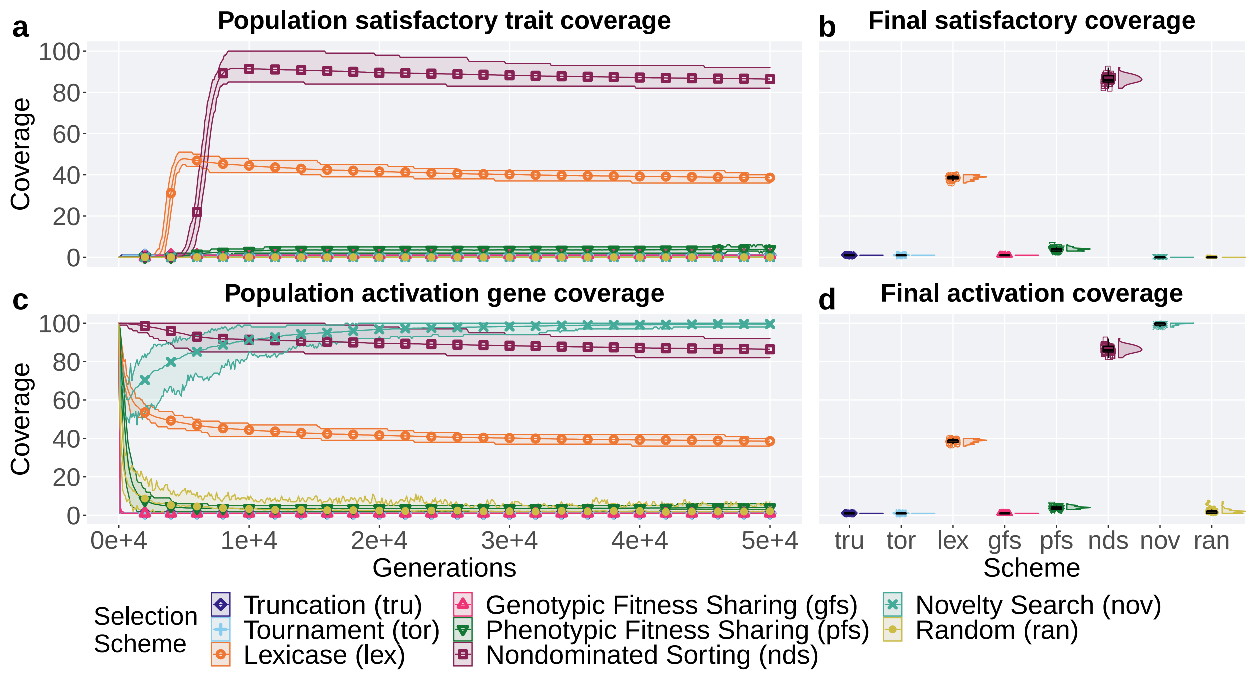

Figure 6.7: Results for selection schemes evaluated on contradictory objective diagnostic. Population-level unique satisfactory traits over (a) over time and (b) at 50,000 generations. Population-level activation gene coverage (c) over time and (d) as 50,000 generations. For panel (a) and (c), we plot the average across the 50 replicates, with shading between the maximum and minimum data.

Because starting populations are initialized with genomes consisting of random values between 0.0 and 1.0, each individual has a random activation gene. As such, initial populations have high activation gene coverage across all treatments (Figure 6.7c). However, activation gene coverage rapidly decreases for all selection schemes, except nondominated sorting. In fact, each replicate of tournament selection, truncation selection, and genotypic fitness sharing rapidly collapsed to a single activation gene; all other selection schemes, including our random control, maintained multiple activation genes in each population (Figure 6.7d). For the configuration used here, we found that novelty search maintained more activation genes than nondominated sorting, nondominated sorting maintained more activation genes than lexicase selection, and lexicase selection maintained more activation genes than phenotypic fitness sharing (Wilcoxon rank-sum tests: $p < 10^{-15}$).

In general, selection schemes capable of maintaining populations with diverse activation genes were also able to optimize those genes to satisfactory levels. Note that in such cases, the diverse activation genes were typically optimized in parallel.

We suspect novelty search did not obtain any satisfactory traits due to its preference for novel traits over incremental improvements to existing traits. Novelty search's results on the exploitation rate diagnostic demonstrate it can slowly pursue a single gradient and consistently performs poorly, but better than random search. Clearly, pressure for novel traits does not facilitate reaching a satisfactory trait within the allotted time; yet this pressure did allow novelty search to maintain higher activation gene coverage than all other selection schemes (Figure 6.7d; Wilcoxon rank-sum tests: $p < 10^{-15}$). We suspect this result was due to a combination of the emphasis of finding novel behaviors and the implicit diversity enhancements provided by the archive. By using phenotypic similarity, the population is pressured to optimize different traits, as doing so increases novelty scores. Early in the evolutionary search, activation gene coverage drops, yet it reaches perfect coverage by the end (Figure 6.7c). This drop in coverage occurred due to the population requiring time to diversify, as the starting novelty threshold is too high for solutions to be added to the archive. Note these results include data found by solutions in both the current population and the archive.

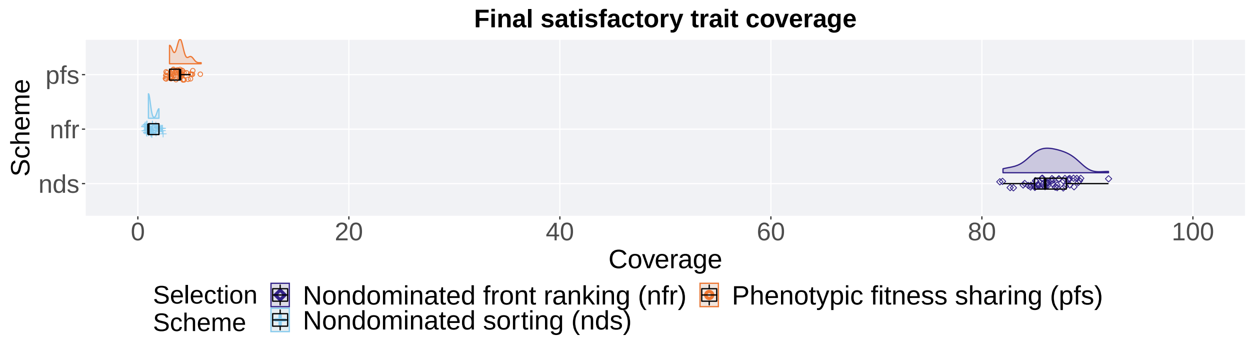

Figure 6.8: Results for nondominated sorting, phenotypic fitness sharing, and nondominated front ranking (nondominated sorting with $\sigma = 0.0$) evaluated on the contradictory objective diagnostic. We present the unique satisfactory traits found at 50,000 generations.

Interestingly, nondominated sorting surpassed all other selection schemes at producing populations with high satisfactory trait coverage (Wilcoxon rank-sum tests: $p < 10^{-15}$). This result is expected, as the contradictory objectives diagnostic generates the ideal search space for nondominated sorting, containing one equidistant Pareto-optimal solution per trait (100). Nondominated sorting's performance appears to be due to its two diversity maintenance mechanisms: nondominated front ranking and phenotypic fitness sharing within fronts. To illuminate the relative importance of both components of nondominated sorting, we applied nondominated front ranking and phenotypic fitness sharing to the contradictory objectives diagnostic (Figure 6.8). There is a significant drop in the final satisfactory trait coverage between using both nondominated front ranking and fitness sharing, as compared to using just one (Wilcoxon rank-sum tests: $p < 10^{-15}$). Clearly, nondominated sorting can simultaneously optimize multiple gradients within a population.

Aside from nondominated sorting, lexicase selection was the only other selection scheme to produce populations with high coverage of satisfactory traits. Lexicase selection's success is consistent with previous theoretical and experimental findings that demonstrate its ability to produce populations with meaningful diversity without impeding simultaneous exploitation (Dolson and Ofria, 2018; Helmuth et al., 2016a, 2020). Lexicase selection's emphasis on selecting specialists (Helmuth et al., 2020) is particularly valuable for performing well on this diagnostic, as a population that finds satisfactory solutions for all traits *must be* a population of specialists; given this, why does lexicase selection result in substantially lower satisfactory trait coverage than nondominated sorting? Previous theoretical and experimental work has shown that lexicase selection's capacity to maintain a given specialist is related to the probability that its associated test cases appear first in the shuffles during selection (Chapter 5). That is, lexicase selection is sensitive to the ratio between population size and the number of test cases. As such, we expect that increasing population size or decreasing diagnostic dimensionality would reduce the performance gap between nondominated sorting and lexicase selection on the contradictory objectives diagnostic.

Neither phenotypic nor genotypic fitness sharing produced populations with high satisfactory trait coverage. However, of these two methods of fitness sharing, phenotypic fitness sharing surpassed genotypic fitness sharing (Figure 6.7b and 6.7d; Wilcoxon rank-sum test: $p < 10^{-15}$). We suspect that this difference in outcome is driven by the information captured by each similarity metric. Phenotypic similarity is more likely to penalize individuals that optimize the same trait, which results in greater selection pressure to optimize different traits. This pressure is masked with genotypic similarity, as the similarity between solutions optimizing the same trait can be decreased by inactive genes drifting. Thus, when comparing two solutions, genotypic fitness sharing does not focus only on the traits that those two solutions are optimizing, but this is exactly what happens with phenotypic fitness sharing. Previous theoretical and experimental work has shown the threshold of dissimilarity and population size affect fitness sharing's ability to fill multiple niches (Della Cioppa et al., 2004); indeed, we find evidence of this, as increasing the threshold value leads to higher activation gene coverage and satisfactory trait coverage (Hernandez et al. (2022d); Wilcoxon rank-sum tests: $p < 10^{-12}$).

Truncation and tournament selection performed poorly on this diagnostic, as both schemes do not maintain or generate diverse populations, and exhibit strong selection pressure (Bäck, 1996; Goldberg and Deb, 1991; Helmuth et al., 2016a). Tournament selection increases the number of unique parents identified through tournaments, yet only one satisfactory trait and activation gene is reached at the end of 50,000 generations for all replicates. Each selection scheme's takeover time suggests that early high-performing solutions will reduce the number of unique traits being optimized in the population, as expected (Bäck, 1996; Goldberg and Deb, 1991). Additionally, aggregating traits makes it impossible to differentiate what trait is being optimized.

6.4.4 Multi-path Exploration

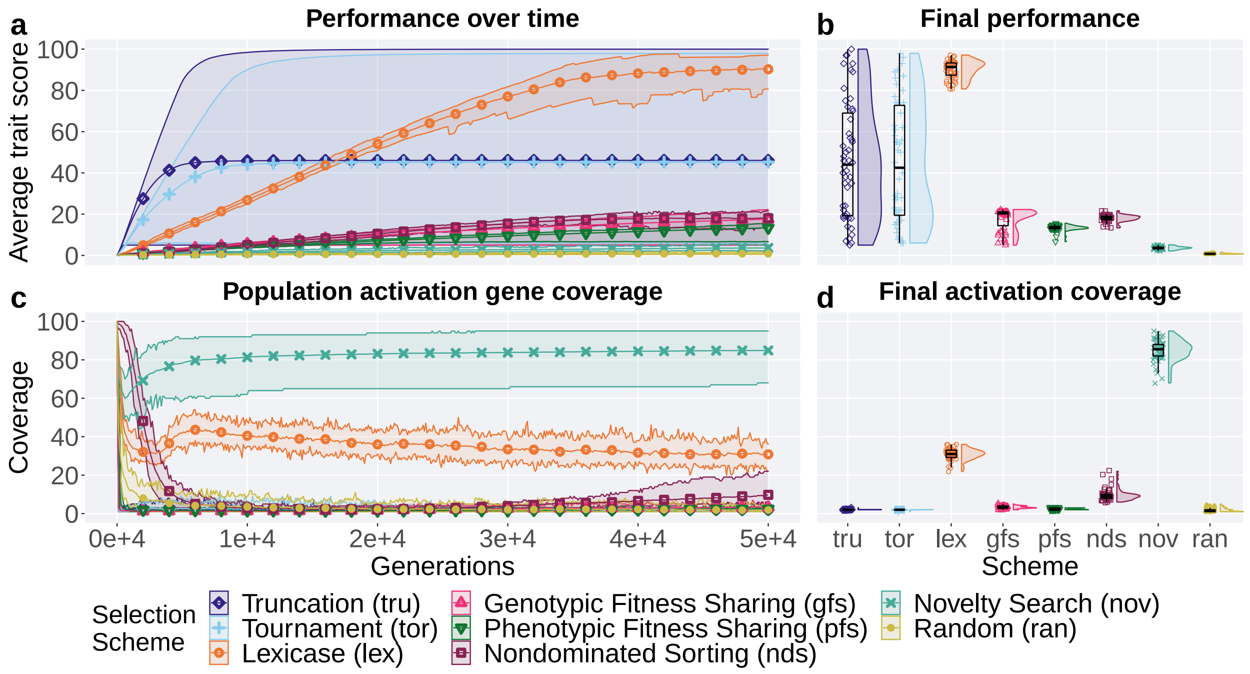

Figure 6.9: Results for selection schemes evaluated on the multi-path exploration diagnostic. Best performance in the population (a) over time and (b) at 50,000 generations. Population-level activation gene coverage (c) over time and (d) at 50,000 generations. For panels (a) and (c), we plot the average across the 50 replicates, with shading between the maximum and minimum data.

The multi-path exploration diagnostic generates a search space with multiple gradients, equal in slope but differing in length, and thus final peak fitness. This search space allows us to compare the relative ability for selection schemes to maintain and simultaneously exploit different gradients, with the goal of fully traversing the gradient that leads to the global optimum. Specifically, we compared performance and population-level activation gene coverage.

We found that all selection schemes improve performance over time, but the rates and levels of improvement differ among schemes (Figure 6.9a). All selection schemes found better-performing solutions in the final populations than the random control (Figure 6.9b; Wilcoxon rank-sum tests: $p < 10^{-15}$). This diagnostic proved to be challenging for all selection schemes, as none were able to consistently evolve a satisfactory solution in the allotted time; yet, tournament, truncation, and lexicase selection each produced high-performing solutions. While tournament and truncation selection were able to find high-performing solutions, only a few replicates were able to do so. Selection schemes that are unable to maintain any exploration should have approximately a 1% chance of stumbling on the optimal trajectory; the results for tournament and truncation were consistent with this expectation. Lexicase selection, by contrast, consistently found high-performing solutions, outperforming all other selection schemes (Wilcoxon rank-sum tests: $p < 10^{-10}$).

Initial populations have high activation gene coverage across all treatments, as starting populations are randomly generated; however, coverage rapidly decreases for all selection schemes (Figure 6.9c). We found no difference between activation gene coverage in the final populations for truncation selection, tournament selection, and the random control (Wilcoxon rank-sum tests: $p > 0.05$); while all other selection schemes maintain more activation genes than the random control (genotypic fitness sharing, lexicase selection, novelty search, nondominated sorting: $p < 10^{-7}$; phenotypic fitness sharing: $p < 10^{-3}$). As in the previous diagnostic results, we found that novelty search, lexicase selection, and nondominated sorting maintained higher activation gene coverage than all other selection schemes (Wilcoxon rank-sum tests: $p < 10^{-15}$). For the configuration used here, lexicase selection maintained more activation genes than nondominated sorting, while novelty search maintained more than both selection schemes (Wilcoxon rank-sum tests: $p < 10^{-15}$). The remaining selection schemes maintained low levels of activation gene coverage.

We found lexicase selection was the only selection scheme to continuously reach better-performing solutions throughout an evolutionary search (Figure 6.9a), while also maintaining high activation gene coverage (Figure 6.9c). These results are expected, as the previous diagnostics allow us to estimate a selection scheme's potential on this diagnostic. Indeed, lexicase selection's performance on the ordered exploitation diagnostic demonstrates its ability to exploit gradients similar to those found in this diagnostic's search space, while its performance on the contradictory objectives diagnostic demonstrates its ability to maintain a diverse set of activation genes. Ultimately, lexicase selection is the only selection scheme to consistently simultaneously achieve the levels of exploration and exploitation to reach high-performing solutions in the final populations. Lexicase selection's strong performance on the exploration diagnostic is consistent with previous work investigating the exploratory capacity of different variants of lexicase selection (Chapter 5).

Success on the contradictory objectives diagnostic predicts success on this diagnostic, as the ability to maintain a diversity of activation genes increases a selection scheme's chance of exploring multiple pathways in the search space. While truncation and tournament selection are apt at exploiting gradients, neither selection scheme can maintain high activation gene coverage, limiting their ability to explore more than one gradient in the search space. Indeed, only replicates where the population (by chance) converges to a high-potential gradient produce high-fitness solutions.

Exploitation is also crucial for success on this diagnostic, as the only way to reach the global optimum is by effectively exploiting the optimum's gradient. While fitness sharing, nondominated sorting, and novelty search maintained multiple activation genes, they failed to fully exploit the associated gradients, which is consistent with their performances on the ordered exploitation diagnostic.

6.5 Conclusion

In this work, we introduce four diagnostics—exploitation rate, ordered exploitation, contradictory objectives, and multi-path exploration—that can be used to measure the relative exploitation and exploration capabilities of selection schemes. We use our diagnostics to compare six popular categories of selection schemes: truncation selection, tournament selection, fitness sharing, lexicase selection, nondominated sorting, and novelty search.

In general, our results are consistent with previous work. Truncation and tournament selection were heavily exploitative with poor capacities for exploration, and novelty search was purely exploratory with no mechanism for exploitation. Nondominated sorting excelled at managing multiple, contradictory objectives, but did not exploit gradients well. Fitness sharing consistently performed poorly across diagnostics, neither exploiting nor exploring particularly well. Lexicase selection effectively balanced exploration with exploitation, performing reasonably well across all diagnostics. Because the results for each diagnostic are heavily dependent on the configurations used for each selection scheme, we included additional replicates for each selection scheme with different parameter configurations (Hernandez et al., 2022d). Overall, our results emphasize the importance of choosing the appropriate selection scheme for a given problem, as each of the selection algorithms that we investigated exhibited distinct trade-offs between different problem-solving characteristics.

Here, we investigated the relative exploration and exploitation abilities of basic versions of only six selection methods. Future work will expand our analyses to more selection schemes, including more complex versions of those investigated here. In particular, we plan to use our diagnostics to help us to disentangle the relative importance of different components of complex selection algorithms (*e.g.*, NSGA-II) by isolating the scheme's constituent components and evaluating them on each diagnostic. We also plan to use our diagnostics to investigate how other factors, such as population size, influence a selection scheme's ability to exploit or explore. Ultimately, our diagnostics allow selection schemes to be investigated with more control than previously possible with standard benchmarking approaches.

Chapter 7
Diagnosing Island Structures Within Selection Schemes

Authors: Jose Guadalupe Hernandez, Alexander Lalejini, and Charles Ofria

This chapter uses the selection schemes framework from Chapter 2 to integrate an island structure within tournament selection, truncation selection, and lexicase selection. We use the same evolutionary algorithm configuration from Chapter 6, with only the previously mentioned selection schemes. Within the selection scheme, we integrate an island structure with a ring topology, where the population is partitioned into four groups (islands). Additionally, migrations occur after a certain amount of generations, where solutions are swapped between each island pairing, and no solution can return to its original island. We find that island structures reduce the exploitation abilities of all selection schemes. Conversely, island structures increase the exploration abilities of both truncation and tournament selection, but reduce exploration abilities for lexicase selection.

7.1 Introduction

"Everything should be made as simple as possible, but not simpler" – Albert Einstein

All evolutionary algorithms (EAs) possess a population structure that defines how solutions compete and interact with one another. Typically, EAs use a well-mixed population where all solutions in the population can interact (Alba and Tomassini, 2002; Sprave, 1999; Tomassini, 2005). Well-mixed populations may be the standard, but previous work has shown that different population structures can increase problem-solving success (Hornby, 2006; Punch, 1998; Skolicki and De Jong, 2004; Whitley et al., 1999). In fact, this improvement can be seen in evolutionary programming, evolutionary strategies, and genetic algorithms, as they all use a well-mixed population by default, but population structures can help them increase their problem-solving abilities (Cohoon et al., 1987; Duncan, 1993; Rudolph, 1991). A balance must be considered when adding additional features to any EA, as each "upgrade" may improve efficiency in some circumstances, but also introduces new parameters to tune and new interactions to understand. Population structures, in particular, limit interactions from acting across all organisms to just local neighborhoods, which will have a dramatic effect on their dynamics and influence on the evolutionary process.

Due to the limited computational resources in the 1960s when EAs were first conceived, they used well-mixed populations and were implemented to run sequentially on a single processor (Alba and Tomassini, 2002). In order to increase the computational power and resources directed at an EA, new population structures were formulated that could partition a population across independent CPUs to be executed in parallel (Sprave, 1999). Given this shift to parallelization, well-mixed populations were now actively slower and harder to implement than those that used structured populations. These new structures limited interactions among candidate solutions to reduce interprocess communication and maximize the benefits of parallelized EA implementations. Indeed, this need to partition a population across different CPUs led to variations of EAs that make use of structured populations: distributed EAs (Gong et al., 2015), parallel EAs (Sudholt, 2015), and coevolutionary EAs (Miguel Antonio and Coello Coello, 2018). Fortunately, structured populations also had the potential to improve optimization success. Many types of population structures have been explored, but the two most used in practice are island models and cellular models (Gong et al., 2015; Skolicki and De Jong, 2004; Tomassini, 2005). In this work, we focus on diagnosing the impact that island models have on a selection scheme's problem-solving success.

Island models work by dividing an entire population into sub-populations (islands) and periodically migrating solutions across those sub-populations. The concept of partitioning a population into independent subpopulations was inspired by nature (Bäck et al., 1997; Cohoon et al., 1987; Rozenberg et al., 2012; Tomassini, 2005), as a population of organisms can be initially well-mixed but be abruptly divided by external factors. For example, allopatric speciation can occur when a population becomes separated by external factors, such as a rare event that moves members of a species from one island to another (MacArthur and Wilson, 1967). As a result, the subpopulations are unable to exchange genetic material and the organisms on each island accumulate mutations that differentiate them from their recent kin on other islands. Both subpopulations may continue to thrive in similar environments, yet, each group may evolve to have different traits and behaviors. Evolutionary algorithms can benefit from this scenario, where the subpopulations can diverge to simultaneously explore different regions of the search space. Indeed, allopatric speciation inspired an early variation of island models with EAs in Cohoon et al. (1987), where the island model EA found better-performing solutions for an optimal linear arrangement problem with less total 'work' than a well-mixed EA.

Population structures are used throughout all phases within an EA, as structures may dictate how solutions are evaluated (Chapter 3), which solutions compete with one another (Hornby, 2006), and where offspring are placed (Mouret and Clune, 2015). As such, it can be difficult to pinpoint exactly how the structure is affecting overall problem-solving success. This difficulty is exacerbated by the same issues found with the standard approach for analyzing EAs through benchmark suites (Section 1.3). Currently, there is no consensus on how to determine the set of parameters for an island model and more work is needed to understand the kinds of problems a structure and its configuration is best suited for (Fernández et al., 2000, 2003; Punch, 1998; Skolicki, 2007; Tomassini, 2005). Fortunately, our diagnostics give us a controlled environment to measure how island structures influence a population traversing the handcrafted search spaces. Ultimately, this work gives practitioners a better understanding of the search space characteristics that an island model may be useful for.

We focus our analysis on homogeneous island models where all subpopulations remain in perfect synchronization and each island is configured similarly. Specifically, we use a population structure within a single EA that partitions a population into separate islands and periodically migrates solutions between islands. However, this structuring of the population can lead to two different interpretations of how a selection scheme is being used. For example, one can interpret that each individual island uses a unique instance of the same selection scheme, or one can interpret that all islands use a single selection scheme. The selection scheme framework (Section 2) fits the latter interpretation, as the population structure is a component of the selection scheme and uses it to identify parents accordingly.

In this work, we measure the effects that an island structure has when integrated within truncation selection, tournament selection, and lexicase selection. We compare these results to selection schemes with a well-mixed population structure and an island structure with no migration. We find that the exploitation abilities of these selection schemes are negatively impacted by any island structures when evaluated on the exploitation rate diagnostic and ordered exploitation diagnostic. The magnitude of the impact is influenced by the migration interval. Conversely, we find that island structures increase the exploration abilities of both truncation and tournament selection, but negatively affect the exploration abilities of lexicase selection.

7.2 Island models

While it is widely accepted that diversity plays a role in an EA's ability to solve problems, an EA's parameters must be tuned to properly promote diversity for a given problem (Burke et al., 2004; McPhee and Hopper, 1999; Sudholt, 2020; Črepinšek et al., 2013). For example, low diversity can lead to premature convergence and the inability to escape local optima, but can be beneficial in finding the global optimum in a unimodal search space. Conversely, high diversity can reduce the exploitation of promising regions of the search space, but can help decrease the likelihood of premature convergence in a multimodal search space. As a result, numerous techniques have been used to promote diversity, such as penalizing for similarity (Goldberg and Richardson, 1987), selecting for novel behaviors (Lehman et al., 2008), and injecting randomly generated solutions into the population (Grefenstette, 1992). Island models promote diversity by partitioning a population into subpopulations that alternate between evolving independently and exchanging solutions. Islands can explore distinct regions of the search space during independent evolution and periodically share solutions through migration, often providing bursts of exploitation (Skolicki and De Jong, 2005; Tomassini, 2005; Whitley et al., 1999).

Island models are characterized by three key categories of configurations (Skolicki, 2007): island subpopulation parameters, migration parameters, and island topology parameters. Typically all of these parameters remain constant throughout an evolutionary run (Skolicki, 2007), but there are techniques where parameters change throughout a run (Fernández et al., 2000; Lässig and Sudholt, 2011; Wineberg and Chen, 2004). While island models may increase problem-solving success, finding the best configuration for each parameter is not well-understood (Fernández et al., 2000; Skolicki, 2007; Tomassini, 2005). Understanding the strengths and weaknesses of a simple EA is already difficult, and island models exacerbate this issue by integrating an extra layer of complexity (Sprave, 1999). Additionally, all three categories of island model parameters interact with one another and affect the evolutionary trajectory of individual islands and problem-solving success. Typically, the configuration of an island model is set by trial-and-error or by arbitrary choice (Cantú-Paz and Goldberg, 2000; Gong and Fukunaga, 2011).

7.2.1 Individual island configurations

Each individual island can be described by the EA rules used to evolve its constituent subpopulation (Skolicki, 2007), including its mutation rate, selection scheme, solution representation, subpopulation size, etc. Simple island models are *homogeneous.* That is, each island uses the same configuration to direct evolution. Because all islands follow the same search strategy, simple island models rely on stochastic differences from one island to another to diversify the evolutionary search at the global level.

Island models with at least one EA parameter differing between islands are labeled as *heterogeneous*. Heterogeneous island models can potentially generate additional diversity, as different EAs implement different balances of exploration and exploitation. Indeed, different heterogeneous island models have proven useful by varying solution representation per island (Skolicki and De Jong, 2004), dynamically changing the subpopulation sizes (Wineberg and Chen, 2004), evolving island parameters (Clune et al., 2005), and randomly setting configurations for islands (Gong and Fukunaga, 2011). Any island model inherently adds a layer of complexity to analyzing evolutionary dynamics, with heterogeneous island models being especially challenging for theoretical analysis.

An island's size plays a significant role in determining how much of the search space its subpopulation can cover, and thus the island model's overall exploitation and exploration abilities (Skolicki and De Jong, 2005; Tomassini, 2005). Smaller islands are limited in exploitation because fewer points can be covered in a promising region of the search space, while larger islands can use more solutions to thoroughly sample the promising region. Larger subpopulations allow for greater diversity to exist within a single island, but their higher exploitation increases the probability that all of the islands will explore the same region of the search space. The reduced exploitation in smaller islands (combined with the greater number of islands given a fixed total population size) results in them having a greater diversity *between* islands and more distinct regions of the search space being simultaneously explored. As such, the balance between the number and size of islands may need to vary by problem, as each problem requires a specific combination of exploitation and exploration.

7.2.2 Migration configurations

Migration events can be defined by migration size, migration interval, and emigration and immigration policies (Skolicki, 2007). The migration size determines the total number of solutions that migrate when a migration event occurs, and the migration interval determines the number of generations that pass between migration events. Both play an important factor in an island model's problem-solving success, but there is evidence to suggest that the migration interval plays a bigger role (Skolicki and De Jong, 2005). Emigration policies specify the rules for choosing solutions to migrate, and whether they are copied or moved (*i.e.*, do chosen migrants also remain on their source island). Immigration policies specify how incoming solutions are integrated into the receiving island.

Given an overall population size of N and k subpopulations with sizes of n_1 through n_k (where $\sum_{i=1}^{k} n_i = N$), a migration size of zero means no migration between islands occurs and is equivalent to a set of independent runs of an EA with population sizes n_1 through n_k (Cantú-Paz and Goldberg, 2003; Fuchs, 1999). In this scenario, each individual island may cover a unique region of the search space for exploration, but will be limited to local solutions for exploiting a region.

A large migration size can disrupt the evolutionary trajectory for both the sending and receiving islands, as a large number of solutions may need to be replaced in both islands. In the extreme case, entire islands may be swapped, which provides no benefit to the overall evolutionary search. Conversely, a small migration size may not have any effect on the evolutionary trajectory of an island that receives migrants, as the small number of migrants may not survive due to stochasticity or may not introduce sufficient variation to incorporate within the receiving subpopulation.

Migration size must be paired with migration interval to determine the total number of solutions that migrate throughout an evolutionary search. A migration interval that is too large mimics independent runs of an EA, similar to setting the migration size to zero. Conversely, a small migration interval will lead to frequent migration events and may become indistinguishable from one large population. Certain combinations of migration size and migration intervals can have issues as well. For example, a small migration interval paired with a large migration size will not allow islands to generate meaningful diversity before they are fully intermixed again. The best combination of migration size and migration interval must be tuned for the problem at hand, where there is evidence suggesting that it is best to use moderate migration intervals with small migration sizes (Skolicki and De Jong, 2005).

The emigration and immigration policies directly influence an island model's exploitation and exploration abilities (Cantú-Paz, 2001; Sprave, 1999). Typically, emigration policies select either the best or random solutions to migrate, and immigration policies replace either the worst or random solutions within the receiving islands. In Cantú-Paz (2001), the takeover times are formulated for different migration policies, where migrants replace solutions within the receiving island until the best solution completely sweeps the subpopulation. From fastest to slowest takeover time, the following rankings for times were found: (1) good migrants replacing bad solutions, (2) good migrants replacing random solutions, (3) random migrants replacing bad solutions, and (4) random migrants replacing random solutions. This last migration policy had takeover times that were indistinguishable from one large population rather than an island structure.

7.2.3 Island topology configurations

Island topology dictates the number of islands and which islands can exchange solutions within the island model. A trade-off must be made between the sizes of individual islands and the number of islands. The topology of those islands is then formally defined by a graph (Tomassini, 2005), where each node represents an individual island and each edge represents islands that can exchange solutions. For a simple island model, the topology (nodes and edges) remains constant throughout an evolutionary search, where only the solutions on each island differ after a migration event. In practice, common topologies include rings, lattices, stars, and hypercubes (Tomassini, 2005), but more sophisticated techniques exist that randomly generate topologies (Fernández et al., 2000; Tang et al., 2004) and alter a topology throughout an evolutionary run (Lin et al., 1994). Indeed, the topology used for an island model directly influences its exploitation and exploration abilities (Cantú-Paz, 2001; Giacobini et al., 2005; Rudolph, 2001).

The topology of islands influences evolution on each island. For example, consider an island model with a fully-connected topology, where all islands can exchange solutions with one another. In this scenario, each island may send most of its subpopulation as migrants to other islands and replenish its subpopulation with migrants from all other islands. This type of topology can result in the island model behaving like a standard EA with a well-mixed population. Conversely, consider an island model with a ring topology, where each island is arranged in a circular ring and connected to its two neighboring islands. In this scenario, each individual island exchanges migrants only with its two neighboring islands, which slows the spread of solutions across islands relative to a fully connected topology.

7.3 Methods

Here, we used the four diagnostics in the DOSSIER suite to examine the impact that different island structures have on the effectiveness of three common selection schemes: truncation selection, tournament selection, and lexicase selection. For our initial analysis, we conducted twelve sets of experiments, one for each diagnostic and selection scheme combination. Each experiment had three treatments (of 100 runs each), one for each focal population structure: a well-mixed structure, a standard island structure (*i.e.*, an island model with migration), and an island structure with no migration (effectively a set of parallel runs, each with a smaller population). We then compared how different population structures affect each selection scheme's ability to traverse the search space. To keep these experiments comparable with previous work, we followed a similar configuration to the EA in Section 6.3 with an additional step that checked if migration events should be triggered. We used a moderate migration interval (Skolicki and De Jong, 2005) of 500 generations, but in follow-up experiments, we also examined shorter (50) and longer (5000) intervals.

7.3.1 Evolutionary algorithm

At the start of each evolutionary run, the EA initialized the population with 512 genotypes from the lowest region in the search space, where genes varied between 0.0 and 1.0 (Step 1 in Algorithm 7.1). The EA evolved each population for $50,000$ generations (Step 2 in Algorithm 7.1). During each generation, the EA evaluated all candidate solution genomes according to the treatment's diagnostic (Step a in Algorithm 7.1). Once all candidate solutions were assigned a phenotype, the selection scheme identified 512 parents (Step b in Algorithm 7.1). Then, each identified parent asexually produced an offspring with mutations potentially applied to it (Step c in Algorithm 7.1). The EA used this constructed set of offspring to form the next generation of solutions, following the rules of the given population structure. For the standard island structure, a migration event occurred every 500 generations (Step d in Algorithm 7.1).

```
1. Initialize population of solutions and population structure.
2. Repeat for 50,000 generations:
    (a) Evaluate each solution on diagnostic and assign a phenotype.
    (b) Select solutions from the population via phenotype and population
        structure.
    (c) Reproduce offspring asexually with mutations applied.
    (d) If migration:
        i. Migrate solutions between islands.
```

Algorithm 7.1: Pseudocode for evolutionary algorithm in this work.

Population structures and migrations

The well-mixed structure mimics a standard EA, whereas both island structures evenly partition the population into four homogeneous islands. We chose this island count to allow each subpopulation to maintain a reasonable size (128 solutions) for a selection scheme to work with. We used a ring topology within the standard island structure due to its use in practice (Tomassini, 2005). The four islands were arranged into a circular ring, and each island was connected to two neighboring islands.

Isolated island structures did not have migration events. As such, the isolated island structure was identical to four parallel instances of the standard EA, each with a smaller population size. For the standard island structure, migration events occurred at intervals of 500 generations, and 8 solutions migrated between island pairs. We determined this default migration interval and size pairing based on Skolicki and De Jong (2005), which recommended the use of moderate migration intervals and small migration sizes. To perform a single migration, a solution was randomly chosen from each island and the pair were swapped between islands. Migrations occurred between pairs of islands in an order that guaranteed that no solution could return to its starting island, though it was technically possible for an individual solution to be migrated twice. After a migration event, $\approx 12\%$ of each subpopulation comprises migrants. Tomassini (2005) recommends that 10% of a subpopulation should be sent to another island, although this value may be too high for certain problems (Skolicki and De Jong, 2005).

DOSSIER diagnostics

For each DOSSIER diagnostic (see Chapter 6), a solution's genome consists of a numerical vector of dimensionality 100. A diagnostic specifies a transformation from the genotype to a phenotype that is of the same type and dimensionality. Each diagnostic focuses on different aspects of exploitation and exploration:

- The **exploitation rate diagnostic** measures the ability of a selection scheme to exploit a single, smooth fitness gradient.
- The **ordered exploitation diagnostic** measures the ability of a selection scheme to exploit a single, narrow fitness gradient.
- The **contradictory objectives diagnostic** measures the ability of a selection scheme to simultaneously maintain and exploit conflicting objectives.
- The **multi-path exploration diagnostic** measures the ability of a selection scheme to simultaneously explore multiple pathways and pursue narrow pathways.

The population structure used by a selection scheme has no direct effect on how a solution is evaluated by a diagnostic.

Selection

Given a well-mixed population structure, a selection scheme can identify parents from anywhere in the population. The island structures restrict how candidate solutions interact with one another, thus, limiting which solutions compete to become a parent. Specifically, island structures limit competitions to solutions within the same island, where the number of parents identified per island is the size of that island (128 for this work).

A more detailed description of the selection schemes used in this work can be found in Section 6.3.1, where each scheme may combine trait values differently. For example, tournament selection and truncation selection both use an aggregate performance value, whereas lexicase selection uses individual traits to narrow a set of candidate parents. We parameterized the selection schemes used in this work in the same way as in Chapter 6. Specifically, tournament selection uses a tournament size of 8, and truncation selection uses a truncation size of 8. Note that the same configuration of a selection scheme is used regardless of the population structure.

For island model conditions, each selection scheme operates as follows:

- **Truncation selection:** Sort the island's subpopulation by performance and truncate the 8 top-performing candidate solutions. Each of the top performers is identified as a parent 16 times, such that a total of 128 parents are identified for a given island.

- **Tournament selection:** To identify a single parent, randomly select 8 solutions from the island's subpopulation and choose the top-performing candidate solution as a parent. A total of 128 tournaments must be held to identify 128 parents for a given island.

- **Lexicase selection:** The island's subpopulation serves as the starting set of candidate parents, and the set of traits is considered in random order. As each trait is processed only those parents that have the highest value on that trait are kept. Once all traits are processed, if more than one solution is left, a random solution is selected from those remaining. This process is repeated 128 times to identify 128 parents for a given island.

Reproduction

The mutations applied to offspring in this work are similar to the mutations in Chapter 6. The variation applied to an offspring's genotype is limited to point mutations potentially applied to individual genes, where the chance of a mutation occurring is 0.7%. The magnitude of each mutation is obtained from a normal distribution with a mean of 0.0 and a standard deviation of 1.0 ($\mathcal{N}(0.0, 1.0)$). If a mutation leads to a gene going below the lower bound (0.0) or going over the upper bound (100.0), the gene value is rebounded. Point mutations are the only variation operators used in this work so that the impact that different population structures have within the iterative search process can be measured for a selection scheme.

The placement of an offspring differs by the population structure being used. For a well-mixed structure, the offspring is placed back into the single population. For island structures, the offspring is placed on the same island that its parents belonged to.

7.3.2 Hypotheses

Given that island models are meant to increase diversity, we expect that incorporating island structures within any selection scheme will hinder exploitation abilities, but benefit exploration abilities. Of course, we expect that the magnitude of impact an island structure has will vary by selection scheme, as each scheme exhibits is own unique balance of exploitation and exploration. The results for these island structure configurations may generalize to other configuration settings, but more extensive experimentation is needed to verify this claim.

Both truncation selection and tournament selection excel at exploiting gradients, but island structures reduce the opportunity for the best-performing solution to become a parent. As such, we hypothesize this reduction caused by island structures will limit both selection scheme's ability to exploit, especially for isolated islands because there is no mechanism for the best-performing solution or its descendants to migrate to other islands. However, we hypothesize the same reduction should benefit exploration for both selection schemes, as islands can focus on different regions of the search space. Indeed, the increase in exploration abilities for standard island structures will be dictated by how a selection scheme incorporates and maintains new migrants throughout the evolutionary run, as new migrants can be washed away by the existing solutions on the island over time.

Lexicase selection performs relatively well across all diagnostics, meaning it has a good balance of exploitation and exploration. We hypothesize that lexicase selection's ability to exploit will also be negatively impacted, as it already selects a diverse set of solutions to act as parents and the islands further reduce the likelihood of the top-performing solution from being identified. In terms of exploration, we hypothesize that lexicase selection's capacity for exploration will be increased, as lexicase selection already explores well, and now each island can focus on a specific portion of the search space. Additionally, we expect lexicase selection's exploration abilities will not be affected by migrations, as lexicase already selects a diverse set of parents and the region a migrant is in may already be covered within the new island.

7.3.3 Data tracking and analysis

We record the same data in this work similar to the work in Chapter 6. For both the exploitation rate and ordered exploitation diagnostics, we report the best performance found in the population each generation and the generation a satisfactory solution is found. For the contradictory objectives diagnostic, we report both the activation gene and satisfactory trait coverage. For the multi-path exploration diagnostics, we report both the activation gene coverage and the best performance found in the population at each generation.

Statistical analysis

We perform a Kruskal-Wallis test to determine if significant differences among population structures within a selection scheme occurred. If significant differences were observed for a Kruskal-Wallis test (significance level of 0.05), we performed a post-hoc Wilcoxon rank-sum test between population structures with a Bonferroni correction for multiple comparisons. For comparisons of performance, we use the aggregate of the phenotype. For comparisons of coverage, we use the raw coverage value. Additionally, we repeated this process to determine if significant differences among migration intervals occurred within a standard island structure for a given selection scheme.

7.3.4 Software availability

We include supplementary material in (Hernandez et al., 2023) that is hosted on GitHub and contains all the software, data analysis, and documentation for this work. The experiments in this work are implemented using the Empirical Library (Ofria et al., 2020). The data processing, analysis, and visualizations are implemented with a combination of Python3 and R version 4 (R Core Team, 2020). The following R packages were used in this project: ggplot2 (Wickham, 2016b), cowplot (Wilke, 2020), dplyr (Wickham et al., 2020), and PupillometryR (Forbes, 2020). All the data used for visualizations and statistical analysis in this work is available on the Open Science Framework at https://osf.io/vbk8d/.

7.4 Results and Discussion

For each diagnostic, we present the results for truncation and tournament selection together, and lexicase selection separately. Note that the results for the standard island model assume a migration interval of 500 unless otherwise specified. We include additional figures and statistical analyses in our supplemental material in Hernandez et al. (2023).

7.4.1 Exploitation rate diagnostics

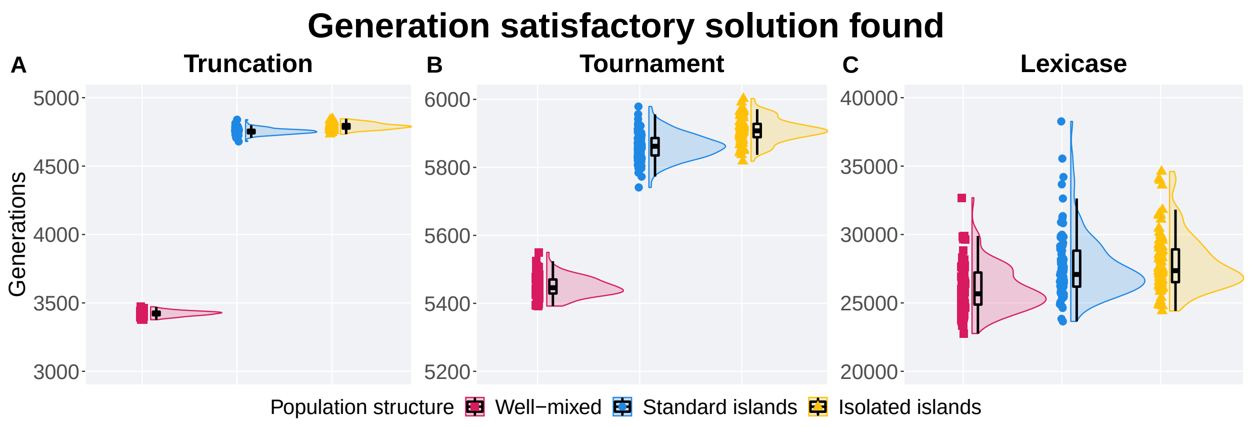

Figure 7.1: Results for the exploitation rate diagnostic. The generation that a satisfactory solution is found for (A) truncation selection, (B) tournament selection, and (C) lexicase selection. Within each panel, each column follows the ordering found in the legend. Note that each panel has a different range of generations.

We used the exploitation rate diagnostic to measure the effect of different population structures on a selection scheme's ability to exploit a single, smooth gradient. Regardless of population structure, all selection schemes improved performance over time, with all 100 replicates finding satisfactory solutions within $50,000$ generations for every pairing of selection scheme and population structure. Consistent with our hypothesis, however, island structures increased the number of generations needed to find satisfactory solutions compared to the well-mixed structure for all selection schemes (Figure 7.1; Wilcoxon rank-sum test: $p < 10^{-3}$). This effect is less obvious with lexicase selection due to wider distributions of solve times and greater overlap, but the mean increase in time to finding a solution is still substantial, with the standard island model taking an average of $\approx 6.18\%$ longer.

In this work, island structures limit the ability to search promising regions of the search space (*i.e.*, exploitation rate) by reducing the number of solutions that can cover a newly discovered region and constraining the number of times high-performing solutions can become parents. The initial islands are populated with random solutions from the bottom of the search space, where selection schemes identify parents within each subpopulation to construct the following generation. Selection schemes may collapse diversity in a subpopulation by favoring the high-performing solutions within an island, which dictates the path taken to reach the optimum. Thus, each island may be covering distinct regions of the search space. A well-mixed structure, however, lets a selection scheme identify parents from the entire population; thus the highest-performing solutions are selected as parents more often in well-mixed than in island models.

Island structures decrease raw exploitation for truncation and tournament

Both truncation and tournament selection improved performance regardless of population structure, but the use of island structures decreased the rate of improvement. Interestingly, island structures increased the number of generations required to find satisfactory solutions when compared to the well-mixed structure for both selection schemes (Figure 7.1 A and B; Wilcoxon rank-sum test: $p < 10^{-3}$). This result can be explained by how island structures change the manner in which a selection scheme can identify parents (Bäck, 1994; Blickle and Thiele, 1995; Goldberg and Deb, 1991). Specifically, both selection schemes had fewer opportunities to select the top performers as parents. Truncation selection identified 8 unique parents for each island, meaning that a total of 32 unique parents were used across islands. In the runs using a well-mixed structure, however, the top 8 unique parents overall were used. Similarly, tournament selection is limited to picking parents on a given island, even when one island is strictly inferior to another. As such, more offspring are potentially descendants of lower-performing solutions in both selection schemes. Indeed, island structures help identify a more diverse set of parents, which helps exploration, but limits high-performing parents from producing offspring, which limits exploitation.

The standard island structure found satisfactory solutions in fewer generations than the isolated island structure for both tournament and truncation selection (Wilcoxon rank-sum test: $p < 10^{-3}$). This result must be caused by migrations, as it is the only difference between island structures. Specifically, at least 9 migration events occurred for truncation selection and at least 11 migration events occurred for tournament selection before all satisfactory solutions were found. Migrations give high-performing solutions more opportunities to become parents by landing on new islands. Both selection schemes likely flood islands with local high-performing solutions, as they naturally exhibit high selection pressure. Migrants that are better than local solutions are likely to become parents and their lineage may eventually dominate the new island. This dynamic provides bursts of exploitation that reduce the number of generations needed to find satisfactory solutions (Cantú-Paz, 2001). Indeed, each island may initially pursue the optimum from different regions of the search space, yet migrations help islands pursue the optimum from a better position in the search space that other islands may reside in.

Increasing the migration interval to 5000 led to fewer migration events before a satisfactory solution was found, as truncation selection found all satisfactory solutions before a migration event could occur and in all cases only one migration event occurred for tournament selection. Conversely, reducing the migration interval to 50 led to over 90 migration events for truncation selection and over 110 migration events for tournament selection before satisfactory solutions were found. For both selection schemes, the migration interval of 50 found satisfactory solutions in fewer generations than the migration intervals of 500 and 5000, and the migration interval of 500 required fewer generations than the interval of 5000 (Figure 7.2 A and B; Wilcoxon rank-sum test: $p < 10^{-3}$). However, the well-mixed structure found satisfactory solutions in fewer generations than the standard island structure with any migration interval (Hernandez et al. (2023); Wilcoxon rank-sum test: $p < 10^{-3}$). The reduction in generations needed to find satisfactory solutions for a migration interval of 50 was expected, as small migration intervals more closely mimic the well-mixed structure and all solutions encounter one another over time (Skolicki and De Jong, 2005).

Island structures decrease raw exploitation for lexicase

Lexicase selection improved performance over time with all population structures, yet island structures reduced the rate of improvement. Specifically, island structures required more generations to find satisfactory solutions than the well-mixed structure (Figure 7.1 C; Wilcoxon rank-sum test: $p < 10^{-3}$). Indeed, favoring solutions that are specialists on a subset of traits hinders lexicase selection's performance because a better total performance is always closer to the optimal peak. This issue is compounded by the island structures, as selection schemes must identify parents within individual islands and there are even fewer opportunities for high-performing solutions to become parents. The well-mixed structure, however, provides more opportunities for solutions

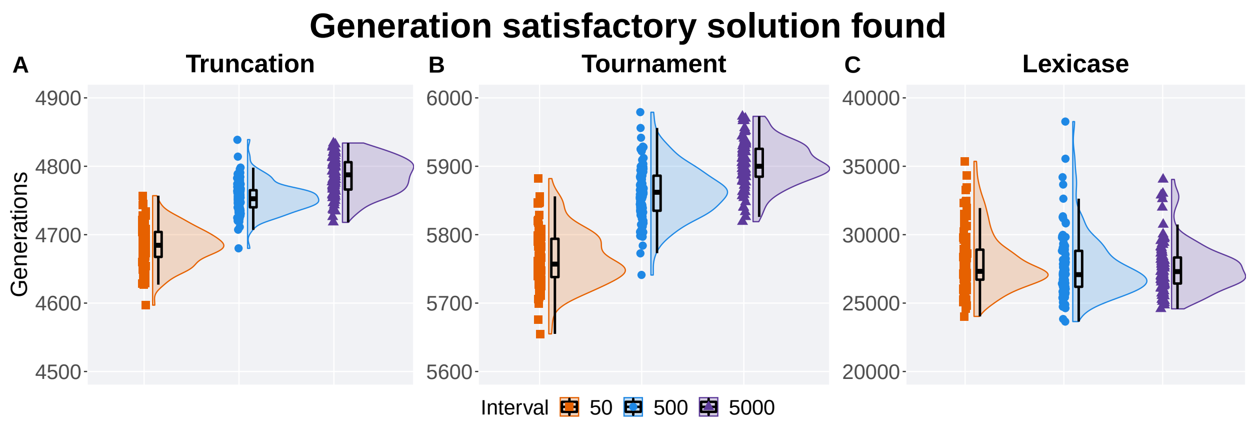

Figure 7.2: Results for the exploitation rate diagnostic for standard island structures with small, moderate, and large intervals. The generation that a satisfactory solution is found for (A) truncation selection, (B) tournament selection, and (C) lexicase selection. Within each panel, each column follows the ordering found in the legend. Note that each panel has a different range of generations.

that specialize on multiple traits, where these solutions are farther up the gradient than solutions that specialize on fewer traits.

No difference was detected in the number of generations needed to reach a satisfactory solution between both island structures (Wilcoxon rank-sum test: $0.05 < p$). Clearly, migration did not help with finding satisfactory solutions, where at least 46 migration events occurred before satisfactory solutions were found. Lexicase selection will generate and maintain islands with specialists on a subset of traits, where the likelihood of islands sharing specialists is low due to its diversity maintenance. When migrants that possess a unique set of specialized traits are sent to new islands, lexicase selection will typically favor them within the new island due to test cases being shuffled each time a parent is identified. The isolated island structure, however, will continue to optimize specialists within individual islands until new traits give rise to new specialists. Indeed, both of these approaches mimic the same exploitation abilities for this diagnostic, as no difference is found between them.

The well-mixed structure found satisfactory solutions in fewer generations than the standard island structure with any migration interval (Wilcoxon rank-sum test: $p < 10^{-3}$). Interestingly, there was no difference in the number of generations needed to reach satisfactory solutions between the three migration intervals (Figure 7.2 C; Wilcoxon rank-sum test: $0.05 < p$). This result was surprising, as all solutions will eventually encounter one another over time with small migration intervals (Skolicki and De Jong, 2005), but this has no detectable effect on lexicase selection.

7.4.2 Ordered exploitation diagnostic

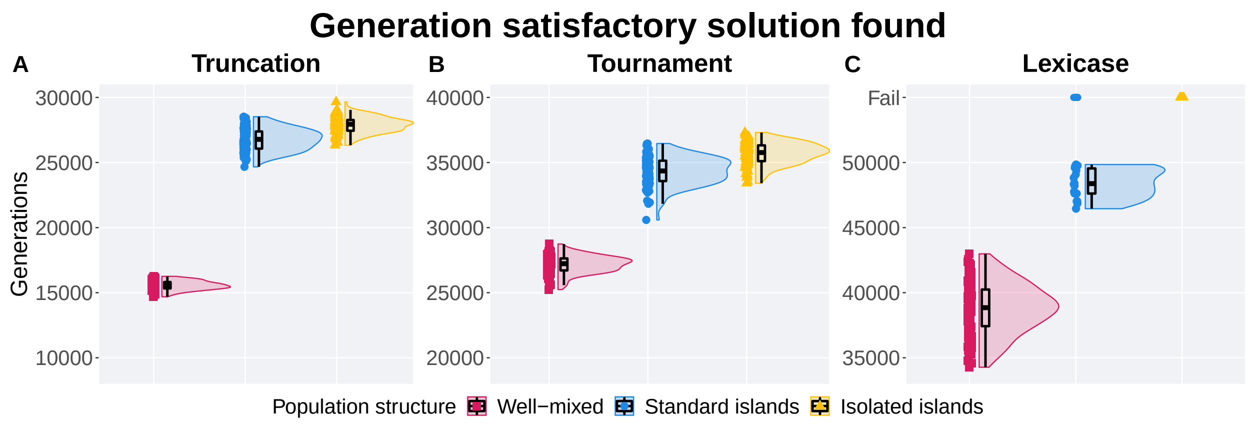

Figure 7.3: Results for the ordered exploitation diagnostic. The generation that a satisfactory solution is found for (A) truncation selection, (B) tournament selection, and (C) lexicase selection. For panel (C), Fail indicates that a satisfactory solution was not found by the end of the evolutionary search. Note that each panel has a different range of generations.

We used the ordered exploitation diagnostic to measure the effect of different population structures on a selection scheme's ability to exploit a single, narrow gradient. Each selection scheme and population structure pairing displayed improved performance over time, but island structures reduced the rate at which performance increased (Hernandez et al., 2023). All replicates of tournament and truncation selection still found satisfactory solutions; lexicase selection, however, found satisfactory solutions in only 18 out of 100 replicates with migration and in none of the replicates without migration. Given that this diagnostic penalizes solutions for diverging away from the narrow gradient, selection schemes that strictly favor those solutions that are further up the gradient will reduce the number of generations needed to find satisfactory solutions and thus increase the likelihood of success. These results provide additional evidence that island structures reduce the exploitation abilities of the selection schemes used in this work, but the magnitude of the impact varies by selection scheme.

Island structures reduce *ordered* exploitation for truncation and tournament

For both tournament and truncation selection, the island structures increased the number of generations needed to find satisfactory solutions when compared to the well-mixed structure (Figure 7.3 A and B; Wilcoxon rank-sum test: $p < 10^{-3}$). This result can be explained by how island structures reduce the opportunity for the top-performing solutions to be selected as parents, similar to the exploitation rate diagnostic. In this case, there are typically fewer beneficial mutations that will allow a solution to follow the gradient. Therefore, there is a greater advantage for those rare higher performing individuals to be selected. As with the previous diagnostic, population structures that facilitate the selection of the top individuals from the entire population are likely to be most effective.

The isolated island structure needs more generations than both the standard island and well-mixed structures to reach satisfactory solutions (Wilcoxon rank-sum test: $p < 10^{-3}$). Yet, the standard island model reached satisfactory solutions in fewer generations than the isolated island model for both selection schemes (Wilcoxon rank-sum test: $p < 10^{-3}$). This result must be due to the migration of high-performing solutions into new islands, where truncation selection underwent at least 48 migration events and tournament selection underwent at least 60 migration events to find satisfactory solutions. High-performing migrants are especially helpful for this diagnostic, as these migrants can have other islands improve the solutions further up the gradient than the local solutions. The isolated island structure takes an average of $\approx 4.2\%$ longer than the standard island model for truncation selection, and $\approx 3.9\%$ for tournament selection.

For both selection schemes, the migration interval of 50 found satisfactory solutions in fewer generations than the migration intervals of 500 and 5000 (Wilcoxon rank-sum test: $p < 10^{-3}$), and the migration interval of 500 required fewer generations than the interval of 5000 (Figure 7.4 A and B; Wilcoxon rank-sum test: $p < 10^{-3}$). However, the well-mixed structure found satisfactory solutions in fewer generations than the standard island structure with any migration interval (Wilcoxon rank-sum test: $p < 10^{-3}$). As above, this result was expected, as increasing the frequency of migration events will place high-performing migrants on new islands and give them more opportunities to be improved.

Island structures reduce the number of satisfactory solutions found with lexicase

Island structures reduced the rate of progress compared to the well-mixed structure for lexicase selection, but all population structures reached high-quality solutions (Hernandez et al., 2023). Specifically, for the well-mixed structure all of the replicates reached satisfactory solutions; for the standard island model 18 out of 100 replicates reached satisfactory solutions; and for the isolated island model, none of the replicates reached satisfactory solutions. These results indicate that the size of the local population affects lexicase selection's ability to reach a satisfactory solution.

While island structures reduced the number of replicates that reached a satisfactory solution,

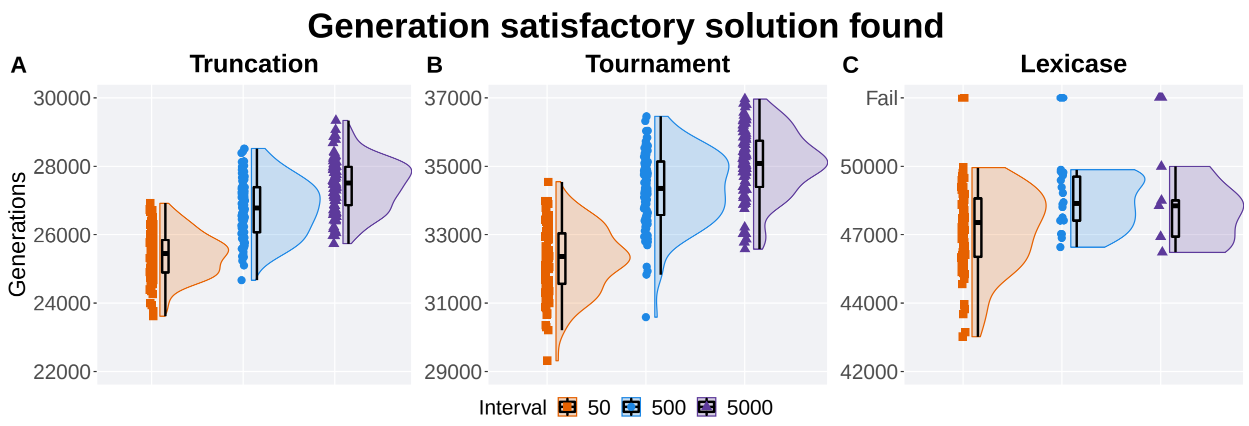

Figure 7.4: Results for the ordered exploitation diagnostic for standard island structures with small, moderate, and large intervals. The generation that a satisfactory solution is found for (A) truncation selection, (B) tournament selection, and (C) lexicase selection. Within each panel, each column follows the ordering found in the legend. Note that each panel has a different range of generations.

the standard island structure reached better-performing solutions than the isolated island structures (Figure 7.3 C; Wilcoxon rank-sum test: $p < 10^{-3}$). This result must be due to migrations, which allow high-performing solutions to propagate onto new islands. Lexicase is driven by diversity and performs best when it has more options to choose from. The isolated island structures, however, do not possess any mechanisms to increase island diversity or to improve the opportunities for near-optimal solutions to be chosen by selection.

Decreasing the migration interval to 50 nearly quadrupled the number of replicates to reach satisfactory solutions (from 18 to 70 out of 100). Conversely, increasing the migration interval to 5000 dropped the number of satisfactory solutions to only 5 out of the 100 replicates. These results are illustrated in Figure 7.4 C. As would be expected, the migration interval of 50 reached better-performing solutions than the migration intervals of 500 and 5000, and the migration interval of 500 found better-performing solutions than the interval of 5000 (Hernandez et al. (2023); Wilcoxon rank-sum test: $p < 10^{-3}$).

7.4.3 Contradictory objectives diagnostic

We used the contradictory objectives diagnostic to measure the effect of different population structures on a selection scheme's ability to simultaneously locate, maintain, and optimize conflicting objectives. All evolutionary runs began with nearly full coverage of activation genes albeit with very low fitness for each gene. This effect is because starting populations are initialized with random solutions, and as such each gene is equally likely to be the activation gene in each individ-

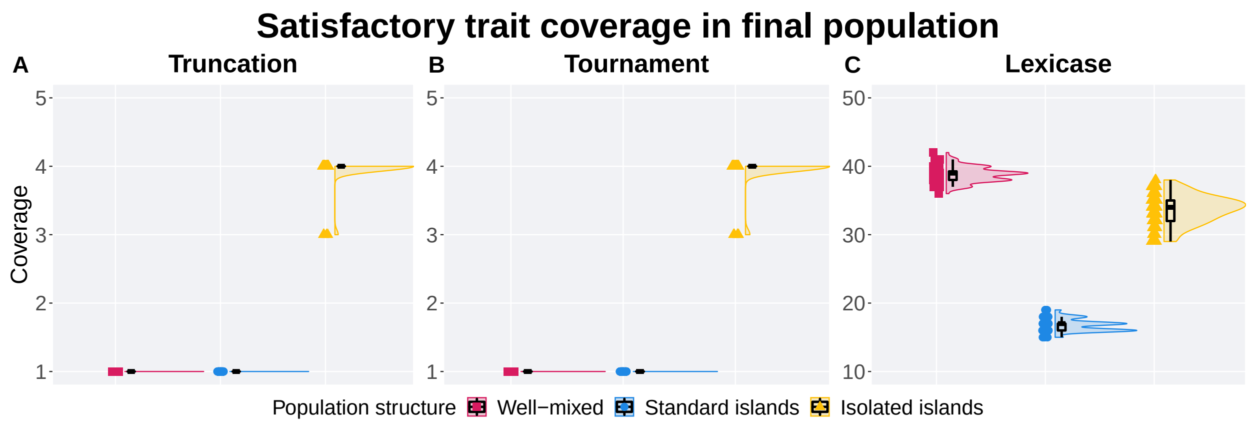

Figure 7.5: Results for the contradictory objectives diagnostic. The satisfactory trait coverage in the final population for (A) truncation selection, (B) tournament selection, and (C) lexicase selection. Note that panel (C) has a greater range of coverage compared to panels (A) and (B).

ual solution; with 512 solutions the probability of *all* 100 genes being active is $> 55\%$. Activation gene coverage, however, decreased over time and the rate at which coverage decreased varies by selection scheme and population structure pairing. As such, the coverage converges to different values under different conditions. Island structures increased satisfactory trait coverage in truncation and tournament selection early in the run. However, in the standard island model, satisfactory trait coverage eventually collapsed to a single trait. These results concurred with our hypothesis that island structures would improve exploration for both truncation and tournament selection. Contrary to our hypothesis, lexicase selection was negatively impacted by island structures, where the island structures decreased satisfactory trait coverage.

Island structures can increase exploration for tournament and truncation

Activation gene coverage rapidly decreased for truncation and tournament selection with all population structures, as both selection schemes naturally exhibit high selection pressure (Chapter 6). For both selection schemes, island structures consistently maintained between 3 and 4 unique activation genes across the entire population at the 100 generation mark. Conversely, the well-mixed structure always collapsed to 1 unique activation gene by the 100 generation mark. The isolated island structure maintained higher activation gene coverage in the final generation than both the standard island and well-mixed structures (Wilcoxon rank-sum test: $p < 10^{-3}$). There was no difference in activation gene coverage in the final generation between the standard island and well-mixed structures, as all populations had exactly one unique activation gene.

Each island under truncation or tournament selection was able to maintain at most one unique activation gene. As such, populations with four islands (and prior to migration, if any) would end up with four randomly chosen activation genes, one per island. These activation genes were usually distinct ($> 90\%$ of the time) but did occasionally overlap by chance. For standard island models, migrations quickly collapsed the activation gene coverage. Indeed, by four migration events, all standard island model populations had reduced to only one unique activation gene.

Beyond merely looking at the number of unique active genes in a population, we are also interested in whether selection was able to optimize those genes. In all cases, the number of active genes at the end of evolution was identical to the number of satisfactory genes. Looking best coverage found across the entire run, both island structures achieved greater satisfactory trait coverage than the well-mixed structure (Wilcoxon rank-sum test: $p < 10^{-3}$), and no difference was detectable between both island structures (Wilcoxon rank-sum test: $0.05 < p$). The success of the standard island model, however, was despite migrations, not because of them as the diverse satisfactory traits were predominantly found prior to the first migration event.

In the standard island model, reducing the migration interval to 50 led to a satisfactory trait coverage of one throughout the evolutionary run, as 50 generations was insufficient to optimize the traits before the first migration collapsed the diversity of the activated genes. Increasing the migration interval to 5000, however, led to qualitatively identical results to the interval of 500 (Hernandez et al., 2023). The only obvious difference with the longer interval between migrations was a delay in the coverage collapsing to one. Indeed, the use of frequent migrations is detrimental to this diagnostic, as islands quickly become dominated by solutions with only one satisfactory trait.

Island structures reduce exploration for lexicase selection

Activation gene coverage decreased at a slower rate for lexicase selection, although the specific rate at which coverage decreases varies by population structure. Specifically, coverage decreased faster for island structures than the well-mixed structure, and coverage decreased faster for the standard island structure than the isolated island structure. By the final generation, the well-mixed structure maintained the most activation gene coverage (36 to 42 unique activation genes), followed by the isolated island structure (29 to 38), with the standard island structure reaching the least coverage (15 to 19) (Hernandez et al. (2023); Wilcoxon rank-sum test: $p < 10^{-3}$). In a standard EA, lexicase selection alone is effective at maintaining diversity, and this effect is stronger as population

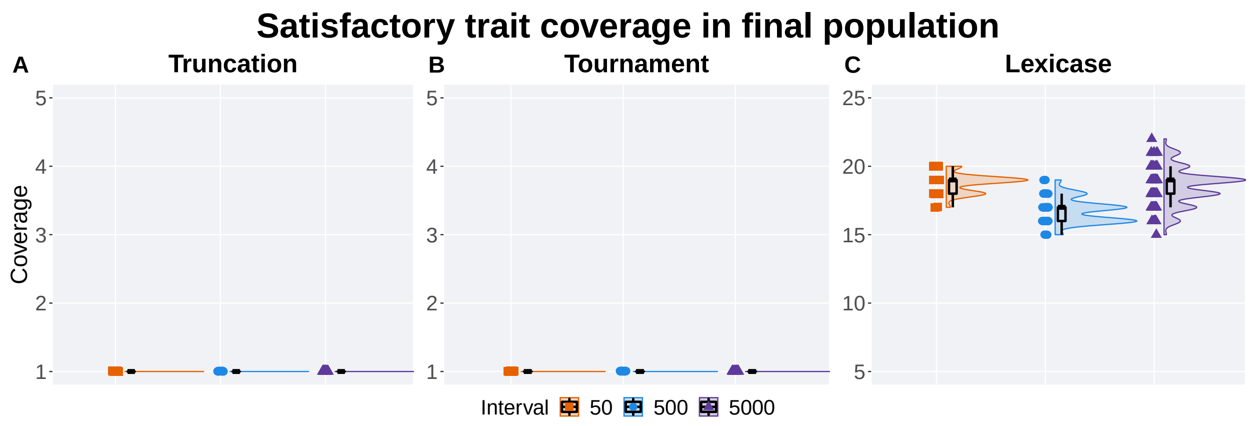

Figure 7.6: Results for the contradictory objectives diagnostic for standard island structures with small, moderate, and large intervals. The satisfactory trait coverage in the final population for (A) truncation selection, (B) tournament selection, and (C) lexicase selection. Note that panel (C) has a greater range of coverage compared to panels (A) and (B).

size increases (Chapter 5). Ignoring migration, each island would be able to maintain substantially lower diversity because of its smaller size (Dolson and Ofria, 2018), and the diversity between islands would be independent, therefore allowing overlap. While migrations may superficially seem like they should help with overall diversity, they will actually cause more overlap between islands. The number of unique traits maintained per island would remain the same in the presence of migration, but the overall pool of traits would become more homogenized. Indeed, across all replicates of the standard island model, we found at most one non-overlapping activation gene among all islands by the end of the run. Since lexicase selection acts on only one island at a time, it would not be able to rebalance based on inter-island duplications.

As would be expected, all activated genes that were maintained over time, were eventually satisfactory. Some active genes were lost due to random chance; examining the entire run, the number of satisfactory traits that ever existed remains consistent with the above: the well-mixed structure ranged between 45 and 51 unique satisfactory traits, the isolated island structure ranged between 35 and 45, and the standard island structure ranged between 21 and 29.

The best satisfactory trait coverage achieved varied by migration interval, where more frequent migration intervals collapsed diversity more rapidly. The migration interval of 5000 ranged between 34 and 45 unique satisfactory traits, the migration interval of 500 ranged between 21 and 29, and the migration interval of 50 ranged between 21 and 26. Indeed, the migration interval of 5000 achieved greater coverage than both migration intervals of 500 and 50 (Wilcoxon rank-sum test:

$p < 10^{-3}$), and no difference was detectable between migration intervals of 500 and 50 (Figure 7.6; Wilcoxon rank-sum test: $0.05 < p$). Yet, all migration intervals achieved less coverage than the well-mixed model (Hernandez et al. (2023); Wilcoxon rank-sum test: $p < 10^{-3}$).

7.4.4 Multi-path exploration diagnostic

We used the multi-path exploration diagnostic to measure the effect of different population structures on a selection scheme's ability to simultaneously maintain and exploit multiple gradients. All selection schemes begin with nearly perfect activation gene coverage due to random start conditions, but, as with the previous diagnostic, coverage decreased over time and the amount of coverage maintained differs by population structure. Both the standard island structure with a migration interval of 5000 and the isolated island structure improved the quality of the solutions found for truncation and tournament selection. The shorter migration intervals did not show obvious improvement. For lexicase selection, however, any island structure negatively affected the quality of solutions reached. Indeed, these results provide additional evidence that an island structure must be tuned for the selection scheme it is combined with and the problem at hand.

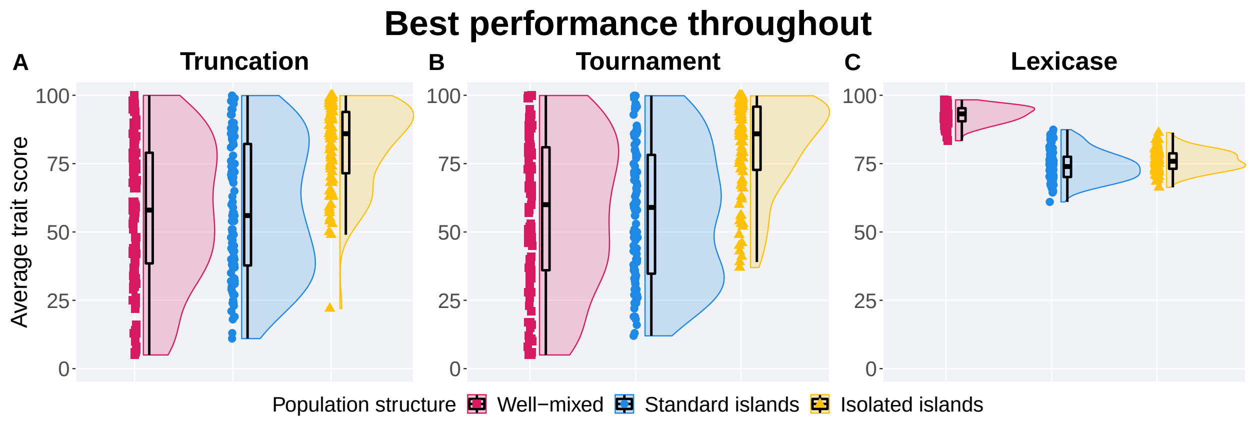

Figure 7.7: Results for the multi-path exploration diagnostic. Distribution of the best performance reached throughout evolutionary search per replicate for (A) truncation selection, (B) tournament selection, and (C) lexicase selection.

Island structures increase exploration for tournament and truncation

Early exploration is critical for this diagnostic so that many paths can be tried, and the best paths can be settled upon. Isolated island structures maintained the greatest activation gene coverage, allowing them to explore more pathways in the search space than either the well-mixed or standard island structure. Both the well-mixed and standard island structures failed to explore many pathways in the search space, as they could not maintain high activation gene coverage. The collapse in diversity observed is consistent with those from the contradictory objective diagnostic (Section 7.4.3).

The quality of the best solutions reached in the entire run varies across selection schemes and population structure pairings. The isolated island structure found better-performing solutions than the well-mixed and standard island structures (Wilcoxon rank-sum test: $p < 10^{-3}$), and we were unable to detect any difference between the well-mixed and standard island structures (Figure 7.7 A and B; Wilcoxon rank-sum test: $0.05 < p$). The increase in the quality of solutions appears to be explained by the independent exploration on the isolated islands. The ordered exploitation results demonstrate that both selection schemes can reach the end of a single, narrow gradient regardless of population structure given enough time. The well-mixed and standard island structures eventually collapsed to low activation gene converge, which prevented exploration of more than one primary pathway (though deleterious mutations could produce shorter pathways that were degraded from the original). Conversely, the isolated island structure maintained an average activation gene coverage approximately three times greater than the other two population structures, which increases the chances of finding better-performing solutions.

For both selection schemes with the standard island model, the migration interval of 5000 reached better-performing solutions than the migration intervals of 50 or 500 (Figure 7.8 A and B; Wilcoxon rank-sum test: $p < 10^{-3}$), and we were unable to detect a difference between the migration intervals of 50 and 500 (Wilcoxon rank-sum test: $0.05 < p$). Interestingly, the migration interval of 5000 was the only interval that reached better-performing solutions than the well-mixed structure (Wilcoxon rank-sum test: $p < 10^{-3}$), and we were unable to detect a difference between the migration interval of 5000 and the isolated island structure (Hernandez et al. (2023); Wilcoxon rank-sum test: $0.05 < p$). The increase in quality of solutions reached with the migration interval of 5000 can be explained by islands having enough time to reach the end of a gradient prior to a migration event, as both selection schemes find satisfactory solutions before 5000 generations on the ordered exploitation diagnostic (Section 7.4.2). Conversely, migrations that happen early on in the evolutionary run will send migrants to new islands that can potentially out-compete solutions pursuing better gradients.

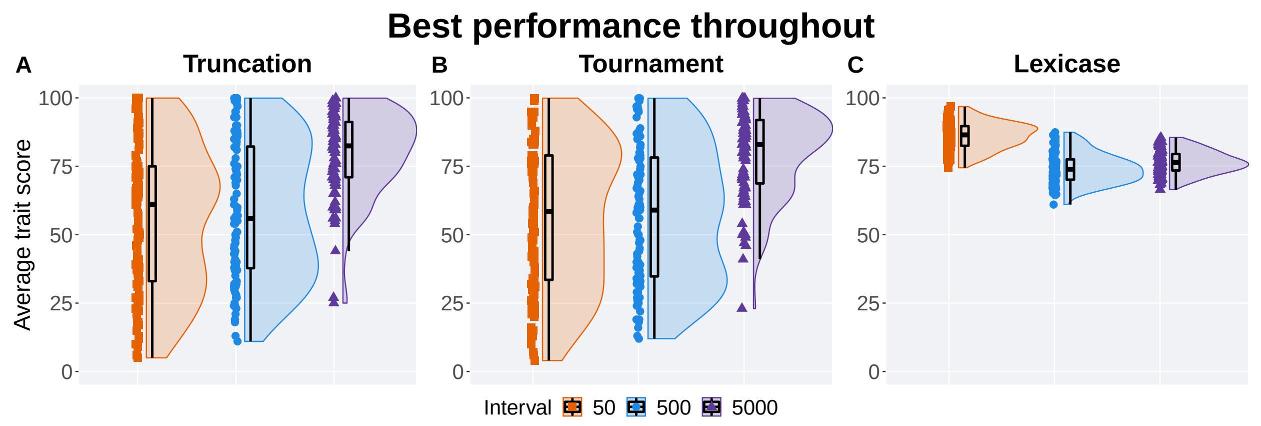

Figure 7.8: Results for the multi-path exploration diagnostic for standard island structures with small, moderate, and large intervals. Distribution of the best performance reached throughout evolutionary search per replicate for (A) truncation selection, (B) tournament selection, and (C) lexicase selection.

Island structures reduce lexicase's ability to simultaneously explore multiple pathways

Maintaining multiple gradients is essential for success with this diagnostic, as it enhances the chances of following the gradient that leads to the optimum. Consistent with the contradictory objectives diagnostic, both island structures reduced lexicase selection's performance compared to a well-mixed population (Figure 7.7 C; Wilcoxon rank-sum test: $p < 10^{-3}$). However, the isolated island structure reached better solutions than the standard island structure (Wilcoxon rank-sum test: $p = 0.0032$).

Islands with migration create a shared pool of candidates across all islands, but the efforts of lexicase selection are merely duplicated from one island to another rather than being able to balance across all available individuals. For islands with no migration, lexicase selection appears to maintain similar activation gene coverage as the well-mixed structure (Hernandez et al., 2023), but this was likely due to each individual island focusing on a unique subset of activation genes. In both cases, the lower coverage in each individual island is not conducive to success on this diagnostic. In fact, previous work illustrates that lexicase selection's performance on this diagnostic is influenced by the ratio between the population size and the number of gradients to explore 5.

The quality of the best solutions reached throughout an evolutionary run varies by the migration interval. Specifically, the migration interval of 50 reached better solutions than migration intervals of 500 and 5000 (Wilcoxon rank-sum test: $p < 10^{-3}$), and the migration interval of 5000 reached better solutions than the interval of 500 (Figure 7.8 C; Wilcoxon rank-sum test:

$p = 0.0013$). This result is interesting, as the standard island model with the migration interval of 500 reached worse performances than either longer or shorter migration intervals, but moderate migration intervals are typically recommended (Skolicki and De Jong, 2005). Yet, the improvements in performances reached by the migration intervals of 50 and 5000 can be explained by how they mimic the well-mixed and isolated island structures, respectively. Frequent migrations closely mimic well-mixed structures and all solutions eventually encounter one another, which benefits lexicase selection on this diagnostic. Conversely, rare migrations closely mimic isolated island structures, where no difference was detected between the standard island structure with a migration interval of 5000 and the isolated island structure when comparing the best solutions reached throughout an evolutionary run (Hernandez et al. (2023); Wilcoxon rank-sum test: $0.05 < p$). Clearly, the pairing of lexicase selection and island structure with the migration interval of 500 is not conducive to success on this diagnostic.

7.5 Conclusion

In this work, we used the DOSSIER suite to measure the effect different island structures have on the exploitation and exploration abilities of three prominent selection schemes: truncation selection, tournament selection, and lexicase selection. The selection schemes tested in this work were paired with either a well-mixed structure, a standard island structure (*i.e.*, an island model with migration), and an isolated island structure (*i.e.*, an island model with no migration). Island structures are known to help promote diversity (Skolicki and De Jong, 2005; Tomassini, 2005), which is a key factor in problem-solving and avoiding premature convergence. We found that the raw exploitation abilities of all three selection schemes were negatively impacted by the island structures, requiring more generations to reach satisfactory solutions. Additionally, island structures improved search space exploration when combined with truncation or tournament selection, as demonstrated by the contradictory objectives and multi-path exploration diagnostics. Contrary to our expectations, however, we found that island structures negatively impacted lexicase selection's exploration abilities.

Indeed, we found that the migration interval for the standard island structure affected performance differently across each diagnostic. For exploitation-focused diagnostics, shorter migration intervals reduced the number of generations needed to find satisfactory solutions for tournament and truncation selection. Yet, for exploration-focused diagnostics, longer migration intervals improved the quality of solutions found for tournament and truncation selection. This provides evidence that the migration interval must be adjusted for a given problem.

Island structures introduce new dynamics that interact with selection schemes. Predicting these combined dynamics may be counterintuitive. For example, our hypotheses on how the island structures would affect both truncation and tournament selection were supported, as exploitation was negatively impacted and exploration was positively impacted. Making predictions for both selection schemes was simple, as they both focus purely on exploitation. However, our simple intuition for how island structures would affect lexicase selection's exploration abilities was not supported. Indeed, lexicase selection's diversity maintenance is sensitive to the ratio between population size and the number of test cases, and island models alter this dynamic. As such, practitioners must consider how a particular population structure should be implemented and how the structure will interact with the selection scheme used.

Here, we focus on the theory behind island models, ignoring the specific implementation details (*e.g.*, whether it is serial or parallel). This abstraction facilitates our ability to describe the model, understand how it operates, and identify general properties. Specifically, we use island structures that mimic homogeneous island models, where all subpopulations remain in perfect synchronization and each island is configured identically. Indeed our diagnostics help illuminate the impact these island structures have on three prominent selection schemes. In future work, we can evaluate more complex island structures, such as heterogeneous island models that remain in perfect synchronization, each island is configured differently (*e.g.*, different population size, selection scheme, variation operators, *etc.*).

In examining the role of island models, the diagnostics were unable to identify conditions under which island models with migration outperformed both alternative population structures. The island models were, however, clearly on the Pareto front, in that they were better than the well-mixed structure on some combinations of diagnostics and selection schemes, and better than the isolated island structure on others. That said, these results identify a potential gap in our diagnostic testing suite.

The four diagnostics described in this dissertation are intended as an initial starting point for the DOSSIER suite. We have already identified the need for additional diagnostics that examine a selection scheme's capacity for valley crossing, though we do not expect those diagnostics to identify a situation where the island models with migration outperform the other contenders. For this situation, we believe that the real advantage of island models does not lie in a pure improvement in either exploration or exploitation; instead, it may be the ability to alternate between the two capacities. While islands are separated they will have an easier time exploring in different directions, thus increasing exploration overall. At each migration event, an island model shifts into much higher exploitation as newly migrated solutions compete with existing options on each island. Selecting a migration interval is likely most important for determining the balance between periods of exploration and bursts of exploitation.

We have multiple possible directions to still go in order to investigate these ideas about exploitation and exploration in island models. In the literature, one common method to illustrate the value of islands is to introduce recombination (Whitley et al., 1999), a factor we ignored in order to first focus on the fundamentals of how island models work. Recombination is valuable because it allows for the best parts of solutions found on different islands to be combined into a single solution. Unfortunately, we also do not expect our existing diagnostics to benefit greatly from this form of additional search. In order for the additional exploration provided by islands to be helpful under recombination, it has to produce building blocks on individual islands that would not have been as easily produced in a well-mixed population. It is not clear that any of the current diagnostics have this property.

In examining diagnostics to add to DOSSIER to capture this important aspect of fitness landscapes, we have multiple options. One possibility is to structure our valley-crossing diagnostic such that there are multiple independent valleys to cross (perhaps one per trait). In such as scenario, if different islands cross different valleys between migration events, recombination would be able to produce a single solution where all valleys had been traversed. Alternatively, we could also produce a diagnostic that does not rely on recombination, but instead creates a simple landscape where populations must be able to alternate between exploration and exploitation many times in order to find a satisfactory solution. The standard island structure should exhibit such an alternation, and we would be able to focus experiments on tuning the migration interval to match the needed exploration periods for optimal evolutionary rate.

Chapter 8
Conclusions

Evolutionary algorithms (EAs) provide an effective set of tools for solving different kinds of problems. Yet, engineering these algorithms to maximize problem-solving success is not an intuitive process. Two issues arise when a new EA is being developed and tested: (1) describing the EA such that it can be intuitively understood and (2) understanding why the EA performs as well as it does. EAs typically consist of multiple integrated components that can make describing them difficult. Additionally, benchmark suites used for testing may contain problems with complex search spaces that do not provide an intuitive understanding of how an EA traverses them. Both issues are challenging, but resolving them will allow researchers to use principled approaches for developing better EAs.

8.1 Contributions

In this dissertation, I developed a theoretical framework that formally defines the selection scheme used within a generational EA into three components, which helps to describe an EA more precisely. The framework played a crucial role in the development of both cohort lexicase and down-sampled lexicase, where down-sampled lexicase selection is one of the more promising lexicase variants to date (Helmuth and Abdelhady, 2020). Indeed, the modifications to lexicase selection illustrated how even small alterations to a selection scheme can lead to different problem-solving dynamics and capabilities. I have also demonstrated the value of developing *diagnostic* problems that facilitate a more intuitive understanding of the strengths and weaknesses of a selection scheme, and how a scheme traverses the search spaces of each diagnostic. Indeed, I was able to use the diagnostics to establish key differences between commonly used selection schemes.

In summary, this dissertation makes the following contributions:

- In **Chapter 2**, I introduced my selection scheme framework that formally defines a selection scheme through three components: population structures, trait processing, and selectors. By representing selection schemes with this framework, I can easily modify, analyze, and extend different selection scheme configurations and combine concepts across otherwise distinct selection schemes.

- In **Chapter 3**, I introduced two new variations of lexicase selection designed to reduce the number of per-generation evaluations: down-sampled lexicase and cohort lexicase. I used the two new lexicase variants to evolve populations of linear genetic programs to solve five different program synthesis problems. This work demonstrated that the random subsampling of test cases can the improve problem-solving success of lexicase selection, and both down-sampled and cohort lexicase variants are successful for a variety of problems.

- In **Chapter 4**, I attempted to develop a deeper understanding of why subsampling test cases could improve problem-solving success for lexicase selection. This work made three key findings, where the subsampling variants of lexicase (1) did not outperform standard lexicase selection given a fixed number of generations, but (2) required fewer total evaluations than standard lexicase selection to evolve solutions on four program synthesis problems, and yet (3) struggled with specialist maintenance. However, the program synthesis benchmark problems used had complex search spaces, making it challenging to fully disentangle the effects of subsampling on how lexicase traversed the search space.

- In **Chapter 5**, I introduced the *exploration diagnostic* as an intuitive tool to measure the exploration abilities of lexicase selection and several of its variants. The exploration diagnostic creates a search space with multiple pathways that differ in path length and peak height, where selection schemes are challenged with steering populations to the correct pathway that leads to the optimum. I made two key findings for standard lexicase selection on this diagnostic: (1) lexicase selection facilitates better search space exploration than tournament selection and (2) lexicase selection's exploration abilities are sensitive to the ratio between the population size and the number of test cases. Additionally, I found that epsilon lexicase outperforms standard lexicase selection on this diagnostic, while the remaining variants degrade the exploration abilities of standard lexicase selection.

- In **Chapter 6**, I introduced the DOSSIER suite that holds the set of diagnostics used in this work. In this initial version of the suite, there are a total of four diagnostics that measure different aspects of exploitation and exploration. I used the DOSSIER suite to diagnose a variety of commonly used selection schemes for their exploitation and exploration abilities. I found that truncation and tournament selection excel at exploitation, but struggle with exploration. Novelty search excelled at exploration, but struggled with exploitation.

Fitness sharing performed poorly across all diagnostics, which implies that there is some aspect of problem-solving that the current set of diagnostics is missing. Lexicase selection performed reasonably well across all diagnostics. Nondominated sorting excelled at managing contradictory objectives, but struggled with exploitation.

- In **Chapter 7**, I used the DOSSIER suite to measure the effect different population structures have on the exploitation and exploration abilities of three selection schemes: truncation selection, tournament selection, and lexicase selection. Three population structures were used in this work: a well-mixed structure, a standard island structure with migration, and an isolated island structure with no migration. I found that island structures reduce the exploitation abilities for the three selection schemes tested. Additionally, I found that island structures increase the exploration abilities for both truncation selection and tournament selection, but negatively impact the exploration abilities of lexicase selection.

Overall, this dissertation represents my initial attempt at moving research forward into more fundamental analyses of how EAs function, how we can disentangle their dynamics more intuitively, and how we can use these principles to design more effective evolutionary problem-solving techniques. I believe that I have clearly demonstrated the power and potential of this approach.

The DOSSIER suite is still new, and I have started it with four diagnostics that I believe are all essential to understanding how an EA operates. That said, there are still a huge number of other critical dynamics and characteristics of EAs that I do not yet have diagnostics to identify.

One of my key ideas for the DOSSIER suite is that it should always be able to highlight factors that make one EA more effective than another on a subset of problems. Any time an EA is identified that is able to show superior performance on a real-world problem, but where this advantage is not yet reflected in DOSSIER results, it indicates a need to add an additional diagnostic that can account for this disparity. The procedure for adding such a diagnostic is to (1) hypothesize about the core mechanism that the EA under investigation is using, (2) simplify the idea as far as possible (without losing its importance), (3) design a proposed diagnostic that directly targets this simplified mechanism, and (4) demonstrate that this new diagnostic provides the needed novel information in an intuitive form.

For example, in our results above, we were unable to demonstrate any advantage to using

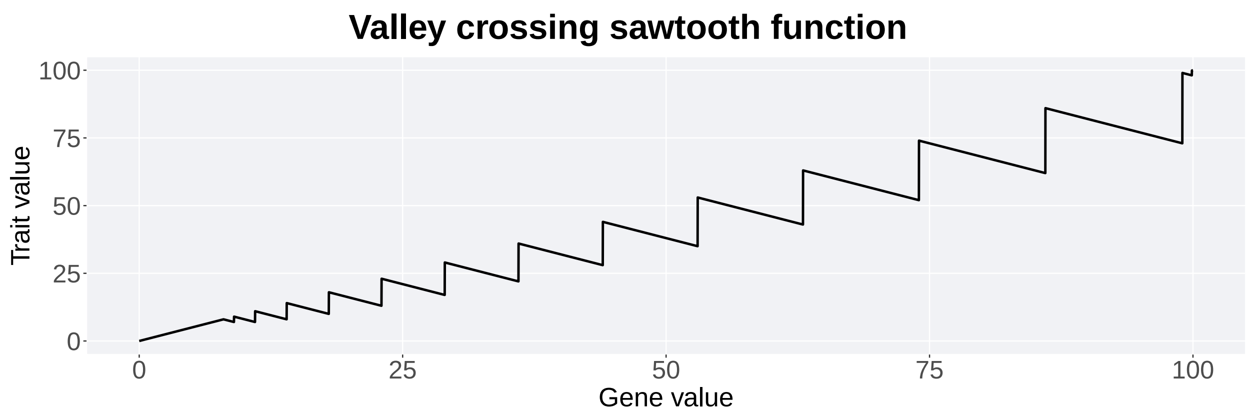

Figure 8.1: Example of how sawtooth valleys can be applied in the conversion of a gene to a trait.

fitness sharing, yet fitness sharing has been frequently used in real-world works to positive results. In preliminary tests, however, I have been able to show that fitness sharing is more effective at valley crossing than any of the other selection schemes that I have been examining. I am still finalizing the exact valley-crossing diagnostics that would be valuable to include in DOSSIER, but it will likely be a sawtooth function for the conversion of each gene to a trait, such as in Figure 8.1.

When running the set of selection schemes from Chapter 6 on this new candidate diagnostic, the results were intriguing (shown in Figure 8.2).

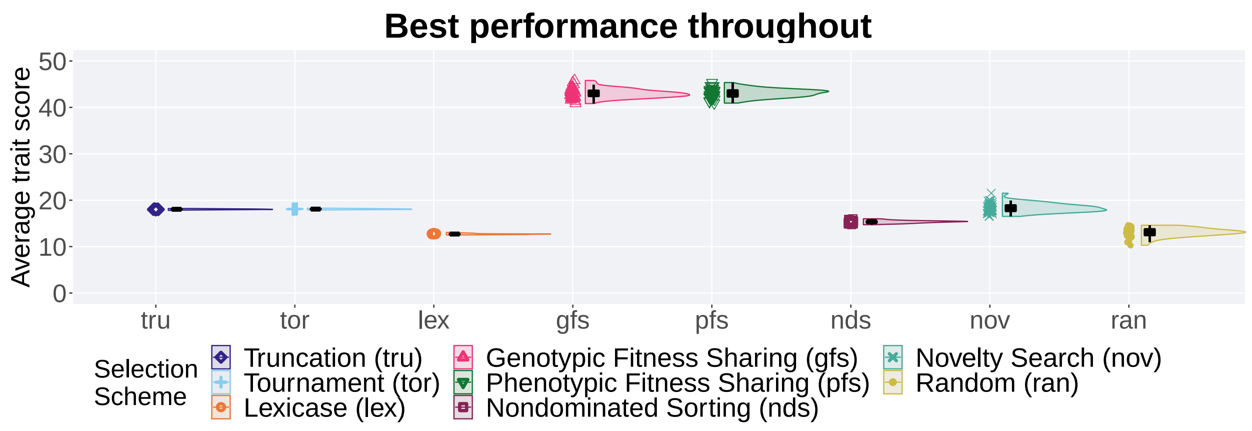

Figure 8.2: Results for the preliminary valley-crossing diagnostic. This diagnostic sums the values of all traits after the valley-crossing transformation from Figure 8.1 is applied. This graph shows the distribution of the best performance reached throughout the evolutionary search for the same selection schemes and parameters evaluated in Chapter 6.

Not only does this diagnostic illustrate the advantage of using fitness sharing, but it also identifies a clear shortcoming of lexicase selection. In retrospect, the result with lexicase is not surprising. Lexicase selection maintains progress on multiple upward pathways by shuffling the order of test cases, but it always picks the very best individuals on each test case as they are applied. As such, lexicase selection is unable to take any downward steps, excluding it from performing valley crossings that cannot occur with a single mutational step. The above extension is a project that I am currently engaging in, but I have many other thoughts on how this work should continue.

8.2 Future directions

Below, I highlight two planned directions: diagnosing EAs with sophisticated selection schemes and additional extensions to the DOSSIER suite.

8.2.1 Diagnosing EAs with sophisticated selection schemes

In this dissertation, I varied components in a simple EA that repeats three key phases: evaluation, selection, and reproduction. This model provides a good starting point when engineering and configuring an EA for the problem at hand, but often needs to be extended for more challenging problems. Typically, only one selection scheme is used within an EA, however, this approach locks the EA into the single search strategy implemented by the selection scheme. Instead, I envision using the DOSSIER suite to identify effective combinations of different selection schemes or other approaches that use more than one scheme to improve overall problem-solving success.

Combining selection schemes

As I have repeatedly shown, all selection schemes implement their own unique search strategy that strikes a balance of exploitation and exploration. Each selection scheme can be broken down into three fundamental components with my selection scheme framework: population structure, trait processing, and selectors. Decomposing a selection scheme into these three components makes it easier to view different selection schemes as interchangeable parts, if compatible. For example, the framework makes it intuitive to combine tournament selection and fitness sharing. The three components of this new selection scheme would consist of a population structure that is well-mixed, traits that are processed so that fitness is shared among similar solutions, and parent identification in tournaments that use this processed fitness value. While this may be a simple example, the same methodology works for more complex selection scheme combinations. For example, lexicase selection and novelty search are combined in Jundt and Helmuth (2019), where the framework would consist of a population structure that is well mixed, trait processing that uses independent test cases and novelty scores, and parent identification using lexicase filtering.

I am most excited about developing new selection schemes that harvest components from existing schemes, and evaluating their effectiveness on both real-world problems and the DOSSIER suite. My hope is to find new selection scheme variants that increase the problem-solving success over the selection schemes from which they derive due to more synergistic interactions. Additionally, the idea of viewing these newly combined selection schemes as offspring from their "parent" selection schemes hints at the idea of using evolution to evolve a new set of promising selection schemes, where a Pareto set can be uncovered.

Multi-selection scheme approach

Using one selection scheme serves as a good starting point when constructing an EA for a given problem, yet this approach limits the EA to one search strategy, regardless of the structure of the local search space. The use of a single search strategy may reduce an EA's problem-solving success, as some regions of a search space may require more exploitation, while others may require more exploration. Using multiple selection schemes may help mitigate this issue, but the use of more than one selection scheme makes it even more difficult to understand the evolutionary dynamics occurring during the search.

Typically, if one selection scheme is used, it identifies the complete set of parents to construct the next generation of solutions. However, if more than one selection scheme is used, each selection scheme must split the number of parents identified. For example, let us assume that both tournament selection and novelty search are going to be used to identify parents. The former selection scheme excels at exploitation and the latter excels at exploration. As such, altering the balance of search space exploitation and exploration is rather simple, where allowing tournament selection to identify more parents increases exploitation, and allowing novelty search to identify more parents increases exploration. This tuning of search space exploitation and exploration will be useful, as there is more flexibility with this approach than using a single selection scheme. Additionally, dynamically adjusting the proportion of identified parents by selection scheme and alternating between the selection scheme being used throughout an evolutionary run may be beneficial to avoid premature convergence and increase the exploitation of promising regions of the search space (Ragusa and Bohm, 2022). Indeed, finding useful combinations of selection schemes that increase problem-solving on real-world problems is preferred, and the DOSSIER suite allows us to understand how the multiple selection schemes affect exploitation and exploration abilities.

Island models provide an additional approach for using multiple selection schemes to guide an evolutionary search. In Chapter 7, I integrated a homogeneous island structure within a single EA, where the same selection scheme was used within each individual island. Partitioning the population into separate islands allowed for each subpopulation to focus on a distinct region of a search space, and migrations allowed for different islands to encounter solutions potentially residing in new regions and differing in performance. However, the selection scheme used can collapse the diversity each individual island maintains if the scheme overly emphasizes high-performing solutions. The use of a heterogeneous island structure may help reduce this issue, as different selection schemes may be used within each island. For example, let us assume that each island is assigned either tournament selection or novelty search to identify parents. The distribution of selection schemes within islands will impact the overall exploitation and exploration, as islands with tournament selection will focus on exploitation, and islands with novelty search will focus on exploration. As such, exploitation and exploration can be adjusted by tuning the number of islands paired with a given selection scheme. Of course, the island structure configuration plays an important role in the overall exploitation and exploration, which can make it difficult to understand how everything is affecting the evolutionary search. Fortunately, the DOSSIER suite allows us to test how each individual component may affect exploitation and exploration abilities.

8.2.2 Additional extensions to the DOSSIER suite

The DOSSIER suite used in this dissertation consisted of four diagnostics (Chapter 6 and 7): the exploitation rate diagnostic, the ordered exploitation diagnostic, the contradictory objectives diagnostic, and the multi-path exploration diagnostic. Each diagnostic generates a unique search space that requires different degrees of exploitation and exploration to reach optima. While the current set of diagnostics revealed key differences among selection schemes, there are additional characteristics to consider for exploitation and exploration.

Both the contradictory objective and multi-path exploration diagnostic encompassed search space exploration from the perspective of populations exploring multiple gradients residing within a search space. Indeed, this flavor of exploration is a problem-solving characteristic that selection schemes encounter in many problems, but a different kind of exploration may be needed to cross fitness valleys in a search space, as discussed above. While the preliminary valley crossing diagnostic effectively adds valley crossing to the exploitation diagnostic, this transformation could be applied to all four existing diagnostics, allowing a more nuanced study of how valleys interact with exploitation and other aspects of exploration.

All of the existing diagnostics specify a unique transformation from a genotype to a phenotype, with no stochasticity in this process. Future work could apply small amounts of noise when a transformation occurs to test how well a selection scheme can traverse a noisy search space. For example, noise can be applied to each trait in the phenotype, where the magnitude of the noise is taken from a normal distribution with a mean of 0.0 and a standard deviation of 1.0 ($\mathcal{N}(0.0, 1.0)$). Stochasticity is found in many real-world problems that EAs encounter, thus, studying how selection schemes react to a noisy environment is important (Beyer, 2000; Neumann et al., 2020). Indeed, the extension of the DOSSIER suite with noise will contribute to this research and generate a deeper understanding of how noise affects selection schemes.

Another axis that could extend the DOSSIER suite is considering the relationship between genes and traits. Currently, each individual gene can be interpreted as a single specific test case and the corresponding trait can be interpreted as a result of the given test case. It is, however, possible to create duplicates of an existing trait that introduce redundancy. In fact, redundancy is seen within real-world problems, as it is common for multiple test cases to focus on the same exact functionality required in the solution. To account for this aspect, we could generate redundancy by randomly determining the number of times the trait associated with each specific gene is generated. The addition of redundant test cases would have interesting results across diagnostics. Specifically, the multi-path exploration diagnostic with redundancy will generate multiple instances of each pathway. This redundancy in pathways could impact selection schemes in different ways, as some may be better than others at handling redundant test cases.

8.3 Closing remarks

My passion for mathematics and computer science naturally guided me to fields related to artificial intelligence (AI) and machine learning (ML). Mathematics allowed me to understand, speak, and write the language used for algorithms within AI and ML. Computer science allowed me to bring these algorithms to life and apply them to real problems. While the application and power of both AI and ML initially captured my interest, it was when I was presented with an alternative process to solve difficult real-world problems – evolution – that I was truly inspired. The first time evolution caught my attention was during CSE 431 Algorithm Engineering, taught by none other than Dr. Charles Ofria and Dr. Alexander Lalejini. Both Dr. Ofria and Dr. Lalejini demonstrated the power and potential of using evolution as a unique optimization procedure to generate solutions that may be unintuitive to a human engineer. Indeed, evolution is a creative optimizer when considering all of the complex organisms in the world today living in harsh environments.

I believe this dissertation advances the ongoing research for developing a deeper understanding of EAs. Specifically, this dissertation adds to this discussion through two key contributions: I engineered tools that increase our understanding of selection scheme abilities and I developed a theoretical framework to describe a selection scheme. The diagnostics provide a new set of problems that allow practitioners to develop a better intuition of the strengths and weaknesses of a selection scheme, which is crucial for understanding why problem-solving success may occur. The selection scheme framework provides a formal definition to describe selection schemes, which can potentially lead to a common language. Clearly, both contributions provide additional knowledge for developing a deeper understanding of EAs.

Throughout the completion of this dissertation, I have gained a tremendous amount of knowledge and experience working with EAs. Chapters 3 and 4 focus on evaluating EAs with problems from benchmark suites, which provided me with experiences on how to use EAs to solve real-world problems. Later chapters focused on the theoretical problem-solving characteristics of the selection scheme within an EA. Following graduation, I will continue to hone my research skills as a post-doctoral scholar with Dr. Jason Moore's Lab at Cedars-Sinai. I am excited to start this postdoc, as it will give me the opportunity to apply the knowledge I gained from this dissertation to new problem domains, including biomedical engineering, and state-of-the-art EAs such as TPOT (Olson and Moore, 2019).

BIBLIOGRAPHY

Adorio, E. and Dilman, U. (2005). Mvf-multivariate test function library in c for unconstrained global optimization methods. *online] http://www. geocities. ws/eadorio/mvf. pdf (accessed 20 January 2013).*

Aenugu, S. and Spector, L. (2019). Lexicase selection in learning classifier systems. In *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '19*, pages 356–364, Prague, Czech Republic. ACM Press.

Ahire, S., Greenwood, G., Gupta, A., and Terwilliger, M. (2000). Workforce-constrained preventive maintenance scheduling using evolution strategies. *Decision Sciences*, 31(4):833–859.

Ahlmann-Eltze, C. and Patil, I. (2021). *ggsignif: Significance Brackets for ggplot2*. R package version 0.6.2.

Alba, E. and Tomassini, M. (2002). Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462.

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2020). *rmarkdown: Dynamic Documents for R*. R package version 2.6.

Andrei, N. (2008). An unconstrained optimization test functions collection. *Advanced Modeling and Optimization,*, 10(1):147–161.

Atkinson, T., Plump, D., and Stepney, S. (2018). Evolving graphs by graph programming. In Castelli, M., Sekanina, L., Zhang, M., Cagnoni, S., and García-Sánchez, P., editors, *Genetic Programming*, pages 35–51, Cham. Springer International Publishing.

Averick, B. M., Carter, R. G., Xue, G.-L., and Moré, J. (1992). The minpack-2 test problem collection. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States).

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

Bäck, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press.

Bäck, T., Fogel, D. B., and Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*. IOP Publishing Ltd., GBR, 1st edition.

Bäck, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23.

Barros, R. C., Basgalupp, M. P., de Carvalho, A. C. P. L. F., and Freitas, A. A. (2012). A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3):291–312.

Bartz-Beielstein, T., Doerr, C., Berg, D. v. d., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., La Cava, W., Lopez-Ibanez, M., Malan, K. M., Moore, J. H., Naujoks, B., Orzechowski, P., Volz, V., Wagner, M., and Weise, T. (2020). Benchmarking in optimization: Best practice and open issues.

Belaqziz, S., Mangiarotti, S., Le Page, M., Khabba, S., Er-Raki, S., Agouti, T., Drapeau, L., Kharrou, M., El Adnani, M., and Jarlan, L. (2014). Irrigation scheduling of a classical gravity network based on the covariance matrix adaptation evolutionary strategy algorithm. *Computers and Electronics in Agriculture*, 102:64–72.

Beyer, H.-G. (1998). On the explorative power of es/ep-like algorithms. In *International Conference on Evolutionary Programming*, pages 323–334. Springer.

Beyer, H.-G. (2000). Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2):239–267.

Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52.

Bhanu, B., Lee, S., and Ming, J. (1995). Adaptive image segmentation using a genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(12):1543–1567.

Blickle, T. and Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms. Technical report, Gloriastrasse 35, CH-8092 Zurich: Swiss Federal Institute of Technology (ETH) Zurich, Computer Engineering and Communications Networks Lab (TIK.

Brameier, M., Banzhaf, W., and Banzhaf, W. (2007). *Linear genetic programming*, volume 1. Springer.

Brindle, A. (1980). *Genetic algorithms for function optimization.* PhD dissertation, University of Alberta.

Burke, E., Gustafson, S., and Kendall, G. (2004). Diversity in genetic programming: an analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62.

Cantú-Paz, E. (2001). Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of Heuristics*, 7(4):311–334.

Cantú-Paz, E. and Goldberg, D. E. (2003). Are multiple runs of genetic algorithms better than one? In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, L. D., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K. A., Jonoska, N., and Miller, J., editors, *Genetic and Evolutionary Computation — GECCO 2003*, pages 801–812, Berlin, Heidelberg. Springer Berlin Heidelberg.

Cantú-Paz, E. and Goldberg, D. E. (2000). Efficient parallel genetic algorithms: theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2):221–238.

Clune, J., Goings, S., Punch, B., and Goodman, E. (2005). Investigations in meta-gas: Panaceas or pipe dreams? In *Proceedings of the 7th Annual Workshop on Genetic and Evolutionary Computation*, GECCO '05, pages 235–241, New York, NY, USA. Association for Computing Machinery.

Coello Coello, C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11):1245–1287.

Coello Coello, C. A., González Brambila, S., Figueroa Gamboa, J., Castillo Tapia, M. G., and Hernández Gómez, R. (2020). Evolutionary multiobjective optimization: open research areas and some challenges lying ahead. *Complex & Intelligent Systems*, 6(2):221–236.

Cohoon, J. P., Hegde, S. U., Martin, W. N., and Richards, D. (1987). Punctuated equilibria: A parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, page 148154, USA. L. Erlbaum Associates Inc.

Crow, J. F. and Kimura, M. (1979). Efficiency of truncation selection. *Proceedings of the National Academy of Sciences*, 76(1):396–399.

Curry, R. and Heywood, M. (2004). Towards efficient training on large datasets for genetic programming. In Tawfik, A. Y. and Goodwin, S. D., editors, *Advances in Artificial Intelligence*, pages 161–174, Berlin, Heidelberg. Springer Berlin Heidelberg.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection.* Murray, London. or the Preservation of Favored Races in the Struggle for Life.

De Jong, K. (1988). Learning with genetic algorithms: An overview. *Machine Learning*, 3(2):121–138.

De Melo, V. V., Vargas, D. V., and Banzhaf, W. (2019). Batch tournament selection for genetic programming: The quality of lexicase, the speed of tournament. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 994–1002, New York, NY, USA. Association for Computing Machinery.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Della Cioppa, A., De Stefano, C., and Marcelli, A. (2004). On the role of population size and niche radius in fitness sharing. *IEEE Transactions on Evolutionary Computation*, 8(6):580–592.

Devaraj, D. and Yegnanarayana, B. (2005). Genetic-algorithm-based optimal power flow for security enhancement. *IEE Proceedings - Generation, Transmission and Distribution*, 152:899–905(6).

Ding, S., Su, C., and Yu, J. (2011). An optimizing bp neural network algorithm based on genetic algorithm. *Artificial intelligence review*, 36:153–162.

Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O. M., and Bäck, T. (2019). Benchmarking discrete optimization heuristics with iohprofiler. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pages 1798–1806, New York, NY, USA. Association for Computing Machinery.

Dolson, E., Lalejini, A., Jorgensen, S., and Ofria, C. (2018). Quantifying the tape of life: Ancestry-based metrics provide insights and intuition about evolutionary dynamics. In Ikegami, T., Virgo, N., Witkowski, O., Oka, M., Suzuki, R., and Iizuka, H., editors, *The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*, pages 75–82, Cambridge, MA. MIT Press.

Dolson, E., Lalejini, A., Jorgensen, S., and Ofria, C. (2020). Interpreting the Tape of Life: Ancestry-Based Analyses Provide Insights and Intuition about Evolutionary Dynamics. *Artificial Life*, 26(1):58–79.

Dolson, E. and Ofria, C. (2018). Ecological theory provides insights about evolutionary computation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 105–106.

Duncan, B. S. (1993). Parallel evolutionary programming. In Fogel, D. B. and Atmar, W., editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 202–208, Evolutionary Programming Society, San Diego, CA.

Eiben, A. E. and Schippers, C. A. (1998). On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50.

Espejo, P. G., Ventura, S., and Herrera, F. (2010). A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(2):121–144.

Ferguson, A. (2020). FergusonAJ/gptp-2019-subsampled-lexicase: GPTP Chapter Companion. https://github.com/FergusonAJ/gptp-2019-subsampled-lexicase.

Ferguson, A. J., Hernandez, J. G., Junghans, D., Lalejini, A., Dolson, E., and Ofria, C. (2020). Characterizing the effects of random subsampling on lexicase selection. *Genetic Programming Theory and Practice XVII*, pages 1–23.

Fernández, F., Tomassini, M., Punch, W. F., and Sánchez, J. M. (2000). Experimental study of multipopulation parallel genetic programming. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P., and Fogarty, T. C., editors, *Genetic Programming*, pages 283–293, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fernández, F., Tomassini, M., and Vanneschi, L. (2003). An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 4(1):21–51.

Floudas, C. A., Pardalos, P. M., Adjiman, C. S., Esposito, W. R., Gümüş, Z. H., Harding, S. T., Klepeis, J. L., Meyer, C. A., and Schweiger, C. A. (1999). *Handbook of Test Problems in Local and Global Optimization*. Springer US, Boston, MA.

Fogel, D., Fogel, L., and Atmar, J. (1991). Meta-evolutionary programming. In *Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems &; Computers*, pages 540–545, Los Alamitos, CA, USA. IEEE Computer Society.

Fogel, D. B. (1988). An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 60(2):139–144.

Fogel, D. B. (1993). Applying evolutionary programming to selected traveling salesman problems. *Cybernetics and Systems*, 24(1):27–36.

Fogel, D. B. and Beyer, H.-G. (1995). A note on the empirical evaluation of intermediate recombination. *Evolutionary Computation*, 3(4):491–495.

Fogel, D. B., Wasson, E. C., and Boughton, E. M. (1995a). Evolving neural networks for detecting breast cancer. *Cancer Letters*, 96(1):49–53.

Fogel, D. B., Wasson, E. C., Boughton, E. M., and Porto, V. W. (1997). A step toward computer-assisted mammography using evolutionary programming and neural networks. *Cancer Letters*, 119(1):93–97.

Fogel, L. J., Angeline, P. J., and Fogel, D. B. (1995b). An evolutionary programming approach to self-adaptation on finite state machines. In *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 355–365. Mit Press.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial intelligence through simulated evolution.* John Wiley & Sons, Oxford, England.

Fong, K., Hanby, V., and Chow, T. (2006). Hvac system optimization for energy management by evolutionary programming. *Energy and Buildings*, 38(3):220–231.

Forbes, S. H. (2020). Pupillometryr: An r package for preparing and analysing pupillometry data. *Journal of Open Source Software*, 5(50):2285.

Forstenlechner, S., Fagan, D., Nicolau, M., and O'Neill, M. (2018). Towards understanding and refining the general program synthesis benchmark suite with genetic programming. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–6.

Forsyth, R. (1981). BEAGLE - A Darwinian Approach to Pattern Recognition. *Kybernetes*, 10(3):159–166.

Freitas, A. A. (2003). A survey of evolutionary algorithms for data mining and knowledge discovery. In *Advances in Evolutionary Computing: Theory and Applications*, pages 819–845, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fuchs, M. (1999). Large populations are not always the best choice in genetic programming. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, pages 1033–1038. Citeseer.

Garden, R. W. and Engelbrecht, A. P. (2014). Analysis and classification of optimisation benchmark functions and benchmark suites. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1641–1649.

Garnier, S. (2018). *viridis: Default Color Maps from matplotlib.* R package version 0.5.1.

Gathercole, C. and Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature — PPSN III*, pages 312–321, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ghosh, S. and Bhattacharya, S. (2020). A data-driven understanding of covid-19 dynamics using sequential genetic algorithm based probabilistic cellular automata. *Applied Soft Computing*, 96:106692.

Giacobini, M., Tomassini, M., and Tettamanzi, A. (2005). Takeover time curves in random and small-world structured populations. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 1333–1340, New York, NY, USA. Association for Computing Machinery.

Giacobini, M., Tomassini, M., and Vanneschi, L. (2002). Limiting the number of fitness cases in genetic programming using statistics. In Guervós, J. J. M., Adamidis, P., Beyer, H.-G., Schwefel, H.-P., and Fernández-Villacañas, J.-L., editors, *Parallel Problem Solving from Nature — PPSN VII*, pages 371–380, Berlin, Heidelberg. Springer Berlin Heidelberg.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, volume 1 of *Foundations of Genetic Algorithms*, pages 69–93. Elsevier.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, page 4149, USA. L. Erlbaum Associates Inc.

Gonçalves, I., Silva, S., Melo, J. B., and Carreiras, J. M. B. (2012). Random sampling technique for overfitting control in genetic programming. In Moraglio, A., Silva, S., Krawiec, K., Machado, P., and Cotta, C., editors, *Genetic Programming*, pages 218–229, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gong, Y. and Fukunaga, A. (2011). Distributed island-model genetic algorithms using heterogeneous parameter settings. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 820–827.

Gong, Y.-J., Chen, W.-N., Zhan, Z.-H., Zhang, J., Li, Y., Zhang, Q., and Li, J.-J. (2015). Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, 34:286–300.

Greenwood, G., Gupta, A., and McSweeney, K. (1994). Scheduling tasks in multiprocessor systems using evolutionary strategies. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 345–349 vol.1.

Grefenstette, J. J. (1992). Genetic algorithms for changing environments. In *Parallel Problem Solving from Nature*.

Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., and Brockhoff, D. (2021). Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144.

Hansen, N., Finck, S., Ros, R., and Auger, A. (2009). Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

Haq, E.-u., Ahmad, I., Hussain, A., and Almanjahie, I. M. (2019). A novel selection approach for genetic algorithms for global optimization of multimodal continuous functions. *Computational intelligence and neuroscience*, 2019.

Harrell, Jr., F. E. (2020). *Hmisc: Harrell Miscellaneous*. R package version 4.4-2.

Hashemi, S., Kiani, S., Noroozi, N., and Moghaddam, M. E. (2010). An image contrast enhancement method based on genetic algorithm. *Pattern Recognition Letters*, 31(13):1816–1824. Metaheuristic Intelligence Based Image Processing.

Hatanaka, T., Uosaki, K., Tanaka, H., and Yamada, Y. (1996). System parameter estimation by evolutionary strategy. In *Proceedings of the 35th SICE Annual Conference. International Session Papers*, pages 1045–1048.

Helmuth, T. and Abdelhady, A. (2020). Benchmarking parent selection for program synthesis by genetic programming. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 237–238.

Helmuth, T. and Kelly, P. (2021). PSB2: The second program synthesis benchmark suite. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, pages 785–794, New York, NY, USA. Association for Computing Machinery.

Helmuth, T., McPhee, N. F., and Spector, L. (2016a). Effects of lexicase and tournament selection on diversity recovery and maintenance. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 983–990.

Helmuth, T., McPhee, N. F., and Spector, L. (2016b). Lexicase Selection for Program Synthesis: A Diversity Analysis. In Riolo, R., Worzel, W., Kotanchek, M., and Kordon, A., editors, *Genetic Programming Theory and Practice XIII*, pages 151–167. Springer International Publishing, Cham. Series Title: Genetic and Evolutionary Computation.

Helmuth, T., Pantridge, E., and Spector, L. (2019). Lexicase selection of specialists. In *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '19*, pages 1030–1038, Prague, Czech Republic. ACM Press.

Helmuth, T., Pantridge, E., and Spector, L. (2020). On the importance of specialists for lexicase selection. *Genetic Programming and Evolvable Machines*, 21(3):349–373.

Helmuth, T. and Spector, L. (2015). General program synthesis benchmark suite. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1039–1046.

Helmuth, T. and Spector, L. (2020). Explaining and Exploiting the Advantages of Down-sampled Lexicase Selection. In *The 2020 Conference on Artificial Life*, pages 341–349, Online. MIT Press.

Helmuth, T. and Spector, L. (2021). Problem-solving benefits of down-sampled lexicase selection. *arXiv:2106.06085 [cs]*. arXiv: 2106.06085. To be published in Artificial Life Journal.

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

Hernandez, J. G., Lalejini, A., and Dolson, E. (2022a). What can phylogenetic metrics tell us about useful diversity in evolutionary algorithms? In *Genetic Programming Theory and Practice XVIII*, pages 63–82. Springer.

Hernandez, J. G., Lalejini, A., Dolson, E., and Ofria, C. (2019). Random subsampling improves performance in lexicase selection. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pages 2028–2031, New York, NY, USA. Association for Computing Machinery.

Hernandez, J. G., Lalejini, A., and Ofria, C. (2021). Supplemental Material GitHub Repository. doi: 10.5281/zenodo.5020769. url: https://doi.org/10.5281/zenodo.5020769.

Hernandez, J. G., Lalejini, A., and Ofria, C. (2022b). An exploration of exploration: Measuring the ability of lexicase selection to find obscure pathways to optimality. *Genetic Programming Theory and Practice XVIII*, pages 83–107.

Hernandez, J. G., Lalejini, A., and Ofria, C. (2022c). A suite of diagnostic metrics for characterizing selection schemes. *arXiv preprint arXiv:2204.13839*.

Hernandez, J. G., Lalejini, A., and Ofria, C. (2022d). Supplemental material for "A suite of diagnostic metrics for characterizing selection schemes". doi: 10.5281/zenodo.6499353. url: https://github.com/jgh9094/ECJ-2022-suite-of-diagnostics-for-selection-schemes.

Hernandez, J. G., Lalejini, A., and Ofria, C. (2023). Supplemental material for "Diagnosing Island Structures Within Selection Schemes". doi: 10.5281/zenodo.7807496. url: https://github.com/jgh9094/Diagnosing-Island-Structures.

Hmida, H., Hamida, S. B., Borgi, A., and Rukoz, M. (2017). Sampling methods in genetic programming learners from large datasets: A comparative study. In *Advances in Big Data*, pages 50–60, Cham. Springer International Publishing.

Holland, J. H. (1962). Outline for a logical theory of adaptive systems. *J. ACM*, 9(3):297–314.

Holland, J. H. (1967). Nonlinear environments permitting efficient adaptation. *Computer and Information Sciences-II*.

Holland, J. H. (1975). Adaptation in natural and artificial systems. *The University of Michigan Press*.

Hooker, J. N. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42.

Hoorfar, A. (2007). Evolutionary programming in electromagnetic optimization: A review. *IEEE Transactions on Antennas and Propagation*, 55(3):523–537.

Hornby, G. S. (2006). Alps: the age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 815–822.

Hou, E., Ansari, N., and Ren, H. (1994). A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120.

Hruschka, E. R., Campello, R. J. G. B., Freitas, A. A., and Ponce Leon F. de Carvalho, A. C. (2009). A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(2):133–155.

Hu, J., Goodman, E., Seo, K., Fan, Z., and Rosenberg, R. (2005). The Hierarchical Fair Competition (HFC) Framework for Sustainable Evolutionary Algorithms. *Evolutionary Computation*, 13(2):241–277.

Hussain, K., Salleh, M. N. M., Cheng, S., and Naseem, R. (2017). Common benchmark functions for metaheuristic evaluation: A review. *JOIV: International Journal on Informatics Visualization*, 1(4-2):218–223.

Jamil, M. and Yang, X.-S. (2013). A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194. PMID: 55204.

Jian, F. and Yugeng, X. (1997). Neural network design based on evolutionary programming. *Artificial Intelligence in Engineering*, 11(2):155–161.

Jong, K. D. (1993). Editorial introduction. *Evolutionary Computation*, 1(1):iii–v.

Jundt, L. and Helmuth, T. (2019). Comparing and combining lexicase selection and novelty search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 1047–1055, New York, NY, USA. Association for Computing Machinery.

Karabulut, K., Öztop, H., Kandiller, L., and Tasgetiren, M. F. (2021). Modeling and optimization of multiple traveling salesmen problems: An evolution strategy approach. *Computers & Operations Research*, 129:105192.

Karafotias, G., Hoogendoorn, M., and Eiben, A. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187.

Kassambara, A. (2021). *rstatix: Pipe-Friendly Framework for Basic Statistical Tests.* R package version 0.7.0.

Katoch, S., Chauhan, S. S., and Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80:8091–8126.

Kauffman, S. and Levin, S. (1987). Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1):11–45.

Kaur, M. and Kumar, V. (2018). Beta chaotic map based image encryption using genetic algorithm. *International Journal of Bifurcation and Chaos*, 28(11):1850132.

Khan, A., Qureshi, A. S., Wahab, N., Hussain, M., and Hamza, M. Y. (2021). A recent survey on the applications of genetic programming in image processing. *Computational Intelligence*, 37(4):1745–1778.

Kim, J.-H. and Myung, H. (1997). Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 1(2):129–140.

Kinnear, K. (1993). Evolving a sort: lessons in genetic programming. In *IEEE International Conference on Neural Networks*, pages 881–888 vol.2.

Koza, J. R. (1989). Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, pages 768–774, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Koza, J. R. (1990a). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Stanford University, Stanford, CA, USA.

Koza, J. R. (1990b). Non-linear genetic algorithms for solving problems. United States Patent 4935877. filed may 20, 1988, issued june 19, 1990, 4,935,877. Australian patent 611,350 issued september 21, 1991. Canadian patent 1,311,561 issued december 15, 1992.

Koza, J. R. and Poli, R. (2005). Genetic programming. In Burke, E. K. and Kendall, G., editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 127–164. Springer US, Boston, MA.

Kriegman, S., Blackiston, D., Levin, M., and Bongard, J. (2020). A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 117(4):1853–1859.

La Cava, W., Helmuth, T., Spector, L., and Moore, J. H. (2018). A Probabilistic and Multi-Objective Analysis of Lexicase Selection and -Lexicase Selection. *Evolutionary Computation*, pages 1–26.

La Cava, W., Spector, L., and Danai, K. (2016). Epsilon-Lexicase Selection for Regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 741–748, New York, NY, USA. ACM. event-place: Denver, Colorado, USA.

Lacroix, B. and McCall, J. (2019). Limitations of benchmark sets and landscape features for algorithm selection and performance prediction. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, page 261262, New York, NY, USA. Association for Computing Machinery.

Lalejini, A. and Hernandez, J. (2019). GECCO-2019-cohort-lexicase GitHub Repository. DOI: 10.5281/zenodo.2603050.

Lalejini, A. and Ofria, C. (2018). Evolving event-driven programs with SignalGP. In *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '18*, pages 1135–1142, Kyoto, Japan. ACM Press.

Lalejini, A. and Ofria, C. (2019). Tag-accessed memory for genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '19*, pages 346–347, Prague, Czech Republic. ACM Press.

Lalejini, A., Wiser, M. J., and Ofria, C. (2017). Gene duplications drive the evolution of complex traits and regulation. In *Artificial Life Conference Proceedings 14*, pages 257–264. MIT Press.

Lalejini, A. M. and Hernandez, J. G. (2021). Experiment data. doi: 10.17605/OSF.IO/XPJFT. url: osf.io/xpjft.

Lang, R. D. and Engelbrecht, A. P. (2021). An exploratory landscape analysis-based benchmark suite. *Algorithms*, 14(3).

Langdon, W. B., Poli, R., McPhee, N. F., and Koza, J. R. (2008). Genetic programming: An introduction and tutorial, with a survey of techniques and applications. In *Computational Intelligence: A Compendium*, pages 927–1028, Berlin, Heidelberg. Springer Berlin Heidelberg.

Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., and Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial intelligence review*, 13:129–170.

Lässig, J. and Sudholt, D. (2011). Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms*, FOGA '11, page 181192, New York, NY, USA. Association for Computing Machinery.

Le, T. T., Fu, W., and Moore, J. H. (2019). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, 36(1):250–256.

Lee, C.-Y. and Yao, X. (2004). Evolutionary programming using mutations based on the levy probability distribution. *IEEE Transactions on Evolutionary Computation*, 8(1):1–13.

Lehman, J. and Stanley, K. O. (2010). Efficiently evolving programs through the search for novelty. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 837–844.

Lehman, J. and Stanley, K. O. (2011a). Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation*, 19(2):189–223.

Lehman, J. and Stanley, K. O. (2011b). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218.

Lehman, J., Stanley, K. O., et al. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336. Citeseer.

Leung, F., Lam, H., Ling, S., and Tam, P. (2003). Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural Networks*, 14(1):79–88.

Li, R., Emmerich, M. T., Eggermont, J., and Bovenkamp, E. G. (2006). Mixed-integer optimization of coronary vessel image analysis using evolution strategies. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 1645–1652, New York, NY, USA. Association for Computing Machinery.

Li, R., Emmerich, M. T., Eggermont, J., Bäck, T., Schütz, M., Dijkstra, J., and Reiber, J. (2013a). Mixed integer evolution strategies for parameter optimization. *Evolutionary Computation*, 21(1):29–64.

Li, X., Engelbrecht, A., and Epitropakis, M. G. (2013b). Benchmark functions for cec2013 special session and competition on niching methods for multimodal function optimization. *RMIT University, Evolutionary Computation and Machine Learning Group, Australia, Tech. Rep.*

Li, X., Tang, K., Omidvar, M. N., Yang, Z., Qin, K., and China, H. (2013c). Benchmark functions for the cec 2013 special session and competition on large-scale global optimization. *gene*, 7(33):8.

Lin, S.-C., Punch, W., and Goodman, E. (1994). Coarse-grain parallel genetic algorithms: categorization and new approach. In *Proceedings of 1994 6th IEEE Symposium on Parallel and Distributed Processing*, pages 28–37.

López-Ibáñez, M., Branke, J., and Paquete, L. (2021). Reproducibility in evolutionary computation. *ACM Trans. Evol. Learn. Optim.*, 1(4).

Louchet, J. (2000). Stereo analysis using individual evolution strategy. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 1, pages 908–911 vol.1.

Luke, S. (2013). *Essentials of Metaheuristics*. Lulu, second edition. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

MacArthur, R. H. and Wilson, E. O. (1967). *Theory of island biogeography. (MPB-1), volume 1*. Monographs in Population Biology. Princeton University Press, Princeton, NJ.

Malan, K. M. and Engelbrecht, A. P. (2013). A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163.

Martínez, Y., Naredo, E., Trujillo, L., Legrand, P., and López, U. (2017). A comparison of fitness-case sampling methods for genetic programming. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(6):1203–1224.

McPhee, N. F. and Hopper, N. J. (1999). Analysis of genetic diversity through population history. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99, pages 1112–1120, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Metevier, B., Saini, A. K., and Spector, L. (2019). Lexicase selection beyond genetic programming. In Banzhaf, W., Spector, L., and Sheneman, L., editors, *Genetic Programming Theory and Practice XVI*, pages 123–136, Cham. Springer International Publishing.

Miguel Antonio, L. and Coello Coello, C. A. (2018). Coevolutionary multiobjective evolutionary algorithms: Survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 22(6):851–865.

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Miller, J. F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99, pages 1135–1142, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Mitchell, M., Holland, J., and Forrest, S. (1991). The royal road for genetic algorithms: Fitness landscapes and ga performance. Technical report, Los Alamos National Lab., NM (United States).

Molga, M. and Smutnicki, C. (2005). Test functions for optimization needs. *Test functions for optimization needs*, 101:48.

Moore, J. M. and McKinley, P. K. (2016). A Comparison of Multiobjective Algorithms in Evolving Quadrupedal Gaits. In Tuci, E., Giagkos, A., Wilson, M., and Hallam, J., editors, *From Animals to Animats 14*, volume 9825, pages 157–169. Springer International Publishing, Cham. Series Title: Lecture Notes in Computer Science.

Moore, J. M. and Stanton, A. (2017). Lexicase selection outperforms previous strategies for incremental evolution of virtual creature controllers. In *Proceedings of the 14th European Conference on Artificial Life ECAL 2017*, pages 290–297, Lyon, France. MIT Press.

Moore, J. M. and Stanton, A. (2018). Tiebreaks and Diversity: Isolating Effects in Lexicase Selection. In *The 2018 Conference on Artificial Life*, pages 590–597, Tokyo, Japan. MIT Press.

Moore, J. M. and Stanton, A. (2019). The Limits of Lexicase Selection in an Evolutionary Robotics Task. In *The 2019 Conference on Artificial Life*, pages 551–558, Newcastle, United Kingdom. MIT Press.

Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites.

Neumann, F., Pourhassan, M., and Roostapour, V. (2020). Analysis of evolutionary algorithms in dynamic and stochastic environments. In Doerr, B. and Neumann, F., editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 323–357, Cham. Springer International Publishing.

Neuwirth, E. (2014). *RColorBrewer: ColorBrewer Palettes*. R package version 1.1-2.

Nguyen, S., Mei, Y., and Zhang, M. (2017). Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems*, 3(1):41–66.

Ofria, C., Moreno, M. A., Dolson, E., Lalejini, A., Rodriguez-Papa, S., Fenton, J., Perry, K., Jorgensen, S., Hoffman, R., Miller, R., Edwards, O. B., Stredwick, J., G, N. C., Clemons, R., Vostinar, A., Moreno, R., Schossau, J., Zaman, L., and Rainbow, D. (2020). Empirical: A scientific software library for research, education, and public engagement. doi: 10.5281/zenodo.4141943.

Ohira, R. and Islam, M. S. (2020). Gpu accelerated genetic algorithm with sequence-based clustering for ordered problems. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8.

Olson, R. S. and Moore, J. H. (2019). Tpot: A tree-based pipeline optimization tool for automating machine learning. In Hutter, F., Kotthoff, L., and Vanschoren, J., editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 151–160, Cham. Springer International Publishing.

O'Neill, M. and Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358.

Orzechowski, P., La Cava, W., and Moore, J. H. (2018). Where are we now? a large benchmark study of recent symbolic regression methods. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1183–1190.

Ostertag, M., Nock, E., and Kiencke, U. (1995). Optimization of airbag release algorithms using evolutionary strategies. In *Proceedings of International Conference on Control Applications*, pages 275–280.

Perkis, T. (1994). Stack-based genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 148–153 vol.1.

Pezzella, F., Morganti, G., and Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212. Part Special Issue: Search-based Software Engineering.

Poli, R. (2001). Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines*, 2(2):123–163.

Poli, R. and McPhee, N. F. (2003a). General schema theory for genetic programming with subtree-swapping crossover: Part i. *Evolutionary Computation*, 11(1):53–66.

Poli, R. and McPhee, N. F. (2003b). General schema theory for genetic programming with subtree-swapping crossover: Part ii. *Evolutionary Computation*, 11(2):169–206.

Punch, B., Zongker, D., and Goodman, E. (1996). The royal tree problem, a benchmark for single and multiple population genetic programming. In *Advances in Genetic Programming: Volume 2*, page 299316, Cambridge, MA, USA. MIT Press.

Punch, W. F. (1998). How effective are multiple populations in genetic programming. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 308–313, University of Wisconsin, Madison, Wisconsin, USA. Morgan Kaufmann.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ragusa, V. and Bohm, C. (2022). Augmenting evolution with bio-inspired "super explorers". In *ALIFE 2022: The 2022 Conference on Artificial Life*. MIT Press. 56.

Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment Library Translation*, 1122.

Rechenberg, I. (1973). Evolutionsstrategie : Optimierung technischer systeme nach prinzipien der biologischen evolution.

Rodionova, A., Antonov, K., Buzdalova, A., and Doerr, C. (2019). Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 855863, New York, NY, USA. Association for Computing Machinery.

Ross, B. J. (2000). The effects of randomly sampled training data on program evolution. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, GECCO'00, pages 443–450, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Rozenberg, G., Bäck, T., and Kok, J. N. (2012). *Handbook of natural computing*. Springer, Berlin, Heidelberg.

Rudolph, G. (1991). Global optimization by means of distributed evolution strategies. In Schwefel, H.-P. and Männer, R., editors, *Parallel Problem Solving from Nature*, pages 209–213, Berlin, Heidelberg. Springer Berlin Heidelberg.

Rudolph, G. (2001). On takeover times in spatially structured populations: Array and ring. *Proceedings of the SecondAsia-Pacific Conference on Genetic Algorithms and Applications (APGA '00)*, pages 144–151.

Sareni, B. and Krahenbuhl, L. (1998a). Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97–106.

Sareni, B. and Krahenbuhl, L. (1998b). Fitness sharing and niching methods revisited. *IEEE transactions on Evolutionary Computation*, 2(3):97–106.

Sauter, J. A., Matthews, R., Van Dyke Parunak, H., and Brueckner, S. (2002). Evolving adaptive pheromone path planning mechanisms. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, AAMAS '02, pages 434–440, New York, NY, USA. Association for Computing Machinery.

Schwefel, H.-P. (1965). Kybernetische evolution als strategie der experimentellen forschung in der stromungsmechanik. *Master's thesis, Technische Universitat Berlin, Hermann Fottinger Institut fuer Hydrodynamik*.

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhäuser Basel.

Shailti Swamp, K. and Natarajan, A. (2005). Constrained optimization using evolutionary programming for dynamic economic dispatch. In *Proceedings of 2005 International Conference on Intelligent Sensing and Information Processing, 2005.*, pages 314–319.

Sharma, S. and Kumar, V. (2022). Application of genetic algorithms in healthcare: A review. In Tripathy, B. K., Lingras, P., Kar, A. K., and Chowdhary, C. L., editors, *Next Generation Healthcare Informatics*, pages 75–86, Singapore. Springer Nature Singapore.

Skolicki, Z. and De Jong, K. (2004). Improving evolutionary algorithms with multi-representation island models. In Yao, X., Burke, E. K., Lozano, J. A., Smith, J., Merelo-Guervós, J. J., Bullinaria, J. A., Rowe, J. E., Tiňo, P., Kabán, A., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN VIII*, pages 420–429, Berlin, Heidelberg. Springer Berlin Heidelberg.

Skolicki, Z. and De Jong, K. (2005). The influence of migration sizes and intervals on island models. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1295–1302.

Skolicki, Z. M. (2007). *An analysis of island models in evolutionary computation*. PhD thesis, George Mason University. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2023-02-22.

Slowik, A. and Kwasnicka, H. (2020). Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, 32:12363–12379.

Spector, L. (2012). Assessment of problem modality by differential performance of lexicase selection in genetic programming: A preliminary report. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, pages 401–408, New York, NY, USA. Association for Computing Machinery.

Spector, L., Cava, W. L., Shanabrook, S., Helmuth, T., and Pantridge, E. (2018). Relaxations of lexicase parent selection. In Banzhaf, W., Olson, R. S., Tozier, W., and Riolo, R., editors, *Genetic Programming Theory and Practice XV*, pages 105–120, Cham. Springer International Publishing.

Spector, L., Martin, B., Harrington, K., and Helmuth, T. (2011). Tag-based modules in genetic programming. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11*, pages 14–19, Dublin, Ireland. ACM Press.

Sprave, J. (1999). A unified model of non-panmictic population structures in evolutionary algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1384–1391 Vol. 2.

Squillero, G. and Tonda, A. (2016). Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. *Information Sciences*, 329:782–799.

Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248.

Stanley, K. and Miikkulainen, R. (2002). Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1757–1762 vol.2.

Sudholt, D. (2015). Parallel evolutionary algorithms. In Kacprzyk, J. and Pedrycz, W., editors, *Springer Handbook of Computational Intelligence*, pages 929–959, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sudholt, D. (2020). The benefits of population diversity in evolutionary algorithms: A survey of rigorous runtime analyses. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 359–404, Cham. Springer International Publishing.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-P., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. *KanGAL report*, 2005005(2005):2005.

Sun, Y., Halgamuge, S. K., Kirley, M., and Munoz, M. A. (2014). On the selection of fitness landscape analysis metrics for continuous optimization problems. In *7th International Conference on Information and Automation for Sustainability*, pages 1–6. IEEE.

Swain, A. and Morris, A. (2000). A novel hybrid evolutionary programming method for function optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, volume 1, pages 699–705 vol.1.

Tang, J., Lim, M., Ong, Y., and Er, M. (2004). Study of migration topology in island model parallel hybrid-ga for large scale quadratic assignment problems. In *ICARCV 2004 8th Control, Automation, Robotics and Vision Conference, 2004.*, volume 3, pages 2286–2291 Vol. 3.

Tayarani-N., M.-H., Yao, X., and Xu, H. (2015). Meta-heuristic algorithms in car engine design: A literature survey. *IEEE Transactions on Evolutionary Computation*, 19(5):609–629.

Tomassini, M. (2005). *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series).* Springer-Verlag, Berlin, Heidelberg.

Črepinšek, M., Liu, S.-H., and Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.*, 45(3).

Vermetten, D., Wang, H., López-Ibañez, M., Doerr, C., and Bäck, T. (2022). Analyzing the impact of undersampling on the benchmarking and configuration of evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 867875, New York, NY, USA. Association for Computing Machinery.

Vikhar, P. A. (2016). Evolutionary algorithms: A critical review and its future prospects. In *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pages 261–265.

Watanabe, K., Kiguchi, K., Izumi, K., and Kunitake, Y. (1999). Path planning for an omnidirectional mobile manipulator by evolutionary computation. In *1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No.99TH8410)*, pages 135–140.

Webb, C. O. (2000). Exploring the phylogenetic structure of ecological communities: an example for rain forest trees. *The American Naturalist*, 156(2):145–155.

Weise, T., Chen, Y., Li, X., and Wu, Z. (2020). Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms. *Applied Soft Computing*, 92:106269.

Weise, T., Chiong, R., and Tang, K. (2012). Evolutionary optimization: Pitfalls and booby traps. *Journal of Computer Science and Technology*, 27(5):907–936.

Weise, T., Niemczyk, S., Skubch, H., Reichle, R., and Geihs, K. (2008). A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, page 795802, New York, NY, USA. Association for Computing Machinery.

Weise, T. and Wu, Z. (2018). Difficult features of combinatorial optimization problems and the tunable w-model benchmark problem for simulating them. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, page 17691776, New York, NY, USA. Association for Computing Machinery.

Whitley, D., Rana, S., Dzubera, J., and Mathias, K. E. (1996). Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1):245–276.

Whitley, D., Rana, S., and Heckendorn, R. B. (1999). The island model genetic algorithm: On separability, population size and convergence. *Journal of computing and information technology*, 7(1):33–47.

Wickham, H. (2007). Reshaping data with the reshape package.

Wickham, H. (2016a). *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York.

Wickham, H. (2016b). *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York.

Wickham, H. (2019). *tidyverse: Easily Install and Load the Tidyverse.* R package version 1.3.0.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., and Dunnington, D. (2021). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics.* R package version 3.3.4.

Wickham, H., François, R., Henry, L., and Müller, K. (2020). *dplyr: A Grammar of Data Manipulation.* R package version 1.0.2.

Wilke, C. O. (2020). *cowplot: Streamlined Plot Theme and Plot Annotations for ggplot2.* R package version 1.1.0.

Wineberg, M. and Chen, J. (2004). The shifting balance genetic algorithm as more than just another island model ga. In Deb, K., editor, *Genetic and Evolutionary Computation – GECCO 2004*, pages 318–329, Berlin, Heidelberg. Springer Berlin Heidelberg.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Wright, S. (1943). Isolation by distance. *Genetics*, 28(2):114–138.

Xie, Y. (2020a). *bookdown: Authoring Books and Technical Documents with R Markdown.* R package version 0.21.

Xie, Y. (2020b). *knitr: A General-Purpose Package for Dynamic Report Generation in R.* R package version 1.30.

Yao, X. and Liu, Y. (1997). Fast evolution strategies. In Angeline, P. J., Reynolds, R. G., McDonnell, J. R., and Eberhart, R., editors, *Evolutionary Programming VI*, pages 149–161, Berlin, Heidelberg. Springer Berlin Heidelberg.

Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102.

Yuan, Y., Xu, H., and Wang, B. (2014). An improved nsga-iii procedure for evolutionary many-objective optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 661–668, New York, NY, USA. Association for Computing Machinery.

Zhang, Q., Barri, K., Jiao, P., Salehi, H., and Alavi, A. H. (2021). Genetic programming in civil engineering: advent, applications and future trends. *Artificial Intelligence Review*, 54(3):1863–1885.

Zhao, X., Gao, X.-S., and Hu, Z.-C. (2007). Evolutionary programming based on non-uniform mutation. *Applied Mathematics and Computation*, 192(1):1–11.

Zhou, A., Qu, B.-Y., Li, H., Zhao, S.-Z., Suganthan, P. N., and Zhang, Q. (2011). Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49.

Zhu, H. (2021). *kableExtra: Construct Complex Table with kable and Pipe Syntax.* R package version 1.3.4.